Beyond Entertainment:
Academic Approaches to Gamedev Industry
ISBN 978-83-969444-4-3, AVANT Anthologies

Framework for Gamifying Production Management Principles in Education with Commercial-Off-The-Shelf Games

Cyryl Leszczyński
Poznan University of Economics and Business, Poland
cyryl.leszczynski@ue.poznan.pl

Received 2025-06-07; accepted 2025-12-18; published Online First 2025-12-31.

Abstract

This paper introduces a framework for integrating gamification principles into production management education and production via commercial-off-the-shelf (COTS) software. It synthesizes multidisciplinary insights from management science (logistics), psychology, and computer science to demonstrate how gamified interactions can effectively represent complex management concepts such as lean manufacturing, workflow optimization, TOC, and resource allocation. The framework emphasizes factors like motivation, engagement, and cognitive retention, as well as computational considerations, realistic simulation, adaptive feedback, and comparisons to mathematically optimal solutions. The goal of the study is to outline these games can be repurposed as effective, low-barrier and relatively low-cost tools for teaching complex production systems and developing practical decision-making skills in both academic and professional settings.

Keywords: interactive learning, production management, gamification, supply chain simulation

1. Introduction

As the complexity of production chains in the modern world continues to increase, so does the demand for production managers equipped with both a solid grasp of (Deterding et al., 2011) theory and practical problem-solving skills. These professionals need to navigate dynamic production environments, optimize resource flows and respond to uncertainty with informed decision-making, grounded in the theory behind production management principles, making traditional education and simulation key components in their training.

In this paper, commercial-off-the-shelf (COTS) video games, originally designed with entertainment as their primary focus, can be repurposed and implemented to support the education of future production managers. A structured framework is proposed, laying out the specifics of how a selection of production-oriented video games (*Factorio, Minecraft)* could be integrated into a class on production management, with a focus on *Minecraft* due to its extensive, community-driven mod support. Applying game-design elements in non-game context (like education) is an effective way of enhancing engagement with the material (Deterding et al., 2011), and has been shown to increase information retention and subject understanding (Clark et al., 2016; Deterding et al., 2011). Within this context commercial-off-the-shelf games present a unique opportunity— unlike custom-developed software, COTS games are readily available, cost-effective, and often feature sophisticated production chain simulations that mirror real-world production scenarios. Professional production management software, though offering a much greater degree of simulation accuracy, remains expensive, inaccessible, and generally much more complex.

Integrating COTS video games into production management curricula aligns with experiential learning theories, which emphasize learning through reflection on doing (Kolb, 1984). Immersing students in simulated environments facilitates the development of critical thinking and problem-solving skills relevant to production management, such as decision-making under uncertainty or job scheduling. Despite those potential benefits, the adoption of COTS games in production management education remains limited, and there is a lack of structured frameworks guiding their implementation, which is a gap this paper aims to address, especially for games which have been modified for this specific purpose. The goals of the presented framework are to:

- Analyze the applicability of selected COTS as a means to teach production management principles.
- Demonstrate an example structured framework for integrating those COTS games into a limited curriculum.
- Identify good practices, drawbacks and potential challenges in implementing (and expanding) gamification in production management curricula.

The relative popularity of production-chain oriented video games is a favorable factor in adapting them in educational settings. According to the data aggregated by SteamDB, *Factorio* reached a peak of 118 674 concurrent players during the launch of its latest DLC expansion *Space Age* in October of 2024, which shows a significant increase when compared to the base-game's launch in August 2020, which held a count of 34 700 concurrent players (SteamDB, 2025). *Satisfactory,* another production management simulator, saw a concurrent its peak concurrent player count in September of 2024, when the game left Steam's Early Access program and launched its 1.0 version (SteamDB, 2025). *Minecraft* is considered the single best-selling video game with over 350 million copies sold as of April 2025 (Mojang AB, 2025), though the game in its original, unmodified version offers limited capabilities for production management simulation; more extensive logistics features are added into *Minecraft* via player-made game modifications. Despite not having an official modding API, *Minecraft's* most popular mod-loader *Forge* (CurseForge, 2025) (which provides its own API to create game modifications) allows for implementing new and complex features into the base game of *Minecraft*[1]. The website *CurseForge* (and its accompanying launcher) lists over 10 000 different mods available for download. Out of those, nearly one thousand is under the category of "Energy, Fluid and Item Transport" which implement production-chain features. Game modifications such as *Applied Energistics 2* (175 million downloads), *Mekanism* (131 million downloads), *Logistics Pipes* (13 million downloads), *ComputerCraft* (66 million downloads), *Integrated Dynamics* (90 million downloads) all provide highly flexible and in-depth logistics features that can be used to model real-world production systems; in addition, the modifications are well-documented, open-source, and also highly popular (as indicated by their download statistics), which positively impacts their viability as teaching environments. Game modifications for *Minecraft* are also (mostly) not conflicting and designed to be used together; programmers often iterate on each other's code and design addons and compatibility features to allow for seamless integration with other mods. This approach has enabled the creation of "mod packs", which are pre-constructed instances of *Minecraft* that include many mods as a single executable, often tweaked and further modified to ensure compatibility and performance.

2. Literature Review

Abt's Serious Games (Abt, 1970) first argued that rule-based play could achieve non-trivial learning objectives and has laid out the conceptual foundation for the field. Digital game-based learning emphasizes the motivational pull of fast feedback loops provided by gameplay (Prensky, 2001). Since then, large scale syntheses have confirmed consistent gains in knowledge retention and new skill acquisition: a meta-analysis of over 50 controlled studies (Clark et al., 2016) found a statistically significant mean effect size in favor of digital games over

[1] An important note regarding *Minecraft* modding is that it relates only to the *Java Edition* of the game— on November 18, 2016 Microsoft (which has by that point acquired Mojang, the initial developer of *Minecraft)* released the *Bedrock Edition* of *Minecraft* which supported cross-platform play and was written in C++. Although the *Bedrock Edition* of the game provides limited support for game modifications in the form of "data packs", they provide less flexibility than Java-based modifications written for the *Java Edition* of the game— as such, the *Java Edition* is the version of *Minecraft* which is being described in this paper (unless stated otherwise).

conventional instruction (when conventionally viable). A systemic review of more than 40 studies shown links between gamified education elements and an increase in student engagement and assessment performance (Subhash & Cudney, 2018).

Production management teaching has also seen success when combined with game-based learning. The "Beer Distribution Game" (Sterman, 1989) remains a popular example of how even simple, tabletop simulations reveal intricate supply chain dynamics, such as the bullwhip effect. Customer-off-the-shelf games have also been used in teaching: Squire (Squire, 2004) has used *Sid Meier's Civilization* to frame the series as a set of "epistemic frames" that allowed learners to inhabit different roles. Despite the momentum within the field, studies focus mostly on qualitative impressions and systemic mappings of course outcomes onto in-game results remain sparse, motivating the simple structured alignment method proposed in Section 3.

The two games selected as a basis for a case study in this paper have both been used as teaching platforms: *Minecraft* includes mechanics that allow for the construction of logic gates and has been used to teach Boolean algebra, as well as digital circuit design (Prayaga et al., 2016). A 2025 systematic review of more than 100 studies found that *Minecraft* consistently enhances cognitive and motivational outcomes when aligned with clearly stated curricular goals (García-Álvarez & Acevedo-Borrega, 2025). *Minecraft*'s publisher, Microsoft, has tried to capitalize on the game's educational capabilities by releasing *Minecraft: Educational Edition* which hosts a number of lessons, including ones on logic gates (*Introduction to Logic Gates— Minecraft Education Lesson Library*, n.d.). *Factorio* has been used to demonstrate lean flow, kanban, bottleneck alleviation and other concepts that directly transfer to industrial engineering contexts (Boardman & Krejci, 2021).

Across literature, arises a need for a simple, replicable, and theory-driven method that can help educators connect production management principles to specific mechanics that form the gameplay loop of COTS games, detailed in Section 3.

3. Methodology for Framework Development

A core idea of serious games and gaming in education is that the gameplay not only provides entertainment, but also purposefully aligns with the pedagogical goals of a given curriculum (here: key production management principles). The process of mapping curriculum concepts starts with the identification of said concepts; this is highly dependent on the course and class being taught (e.g., a general introductory 101 course on logistics would include only the broad fundamentals, while an advanced course could build upon prior knowledge and focus on highly specific topics). Afterwards, relevant game mechanics need to be extracted— relevant, here meaning capable of simulating (and preferably visualizing) the previously identified concepts. Bridging the two (game mechanics and their corresponding concepts) is done by designing a learning task (scenario) that requires the players (students) to engage with the mechanic in a controlled way, where successfully completing it necessitates an understanding of the corresponding concept. This "triangulated" approach to concept-game mapping reinforces principles of constructive alignment (Biggs, 1996), to ensure that learning outcomes, in-game tasks and assessments are integrated in a cohesive way. This section includes several concept-game mappings that teach production management principles with gameplay mechanics of *Factorio* and *Minecraft*. Though the presented framework could be used by a wider variety of games, *Factorio* and *Minecraft* have been chosen as examples due to two factors: their popularity (discussed in Section 1) and availability of tools that allow for their modification ('moddability'). *Factorio* is an example of a very popular (estimated by player count; see Section 1) automation-focused game that implements a number of features that relate to production management by default, while also offering ways to further customize the user experience, which is very useful for designing lesson scenarios. To help with modifying the game, *Factorio* provides an official, extensively documented modding API which uses Lua— a relatively beginner-friendly programming language often used in game development (Ierusalimschy et al., 2007). Though its modding community is not as large as *Minecraft's*, game modifications have been an important part of *Factorio's* development, with its lead developer quoting *IndustrialCraft 2* (a *Minecraft* mod) and other mods similar to it as a direct source of inspiration when designing *Factorio* (Kovařík, 2013). Since *Minecraft* is a sandbox and exploration-based game, rather than one with automation and production management at the core of its gameplay loop (like *Factorio*), it requires multiple game modifications running in parallel to be used effectively as a teaching tool under the proposed framework. This means a potential educator would have to go through additional steps when setting up a lesson scenario— due to this increase in complexity, the case using *Minecraft* has been discussed in more detail.

3.1 Case 1: Bottleneck Identification and Resolution in Factorio

A bottleneck (constraint) is any point in a production system where the capacity of a resource or operation is less than the demand placed upon it, thereby limiting the throughput of the entire system. Commonly referred to as "the weakest link", it consists of a single unit (e.g., a machine with a relatively high processing time, an inefficient policy, a worker experiencing a disproportionate workload) that causes work-in-process inventory to build up and an increase in production cycle times, leading to a lowered throughput. Resolving bottlenecks requires their identification— methods include value stream mapping (Liker & Meier, 2005), process cycle-time analysis, and queuing-analysis metrics, which help in spotting and quantifying constraints. Resolving bottlenecks include capacity elevation, buffering upstream work, and preventative measures. The theoretical foundation is laid out in Goldratt's Theory of Constraints (Şimşit et al., 2014) where constraints are the system's primary improvement lever, as well as in the waste-minimizing principles of Lean Manufacturing (Nicholas, 2018).

The continuous recognition and elimination of bottlenecks is a core part of the gameplay loop in production-management oriented video games, and as such, can be used to demonstrate it in an educational setting. *Factorio* includes UI elements that can be used to see production graphs, and other statistics that can be used to identify bottlenecks in complex systems. Though the game itself provides top-down 2D visuals, with a sufficiently large production change, identification of constraints at a glance becomes difficult, which is accurate to the real world— utilizing data visualization to extract valuable information from a system in order to eliminate bottlenecks is key in Value Stream Mapping. In less elaborate systems, bottlenecks in *Factorio* may be quickly visually identified as clogged-up conveyor belts or under-supplied machines with a blinking indicator. *Factorio* offers several ways to alleviate bottlenecks after they have been spotted, and which ones are available to students should depend on the course contents. Designing a task that effectively teaches the student course material and assesses their understanding of it involves preparing: the initial game state, the desired game state, and a scenario description outlining clear goals.

Unlike during usual gameplay, where manually gathering resources to kickstart a production line is a necessary, and often lengthy step, a well-designed task should present students with a pre-constructed environment, where player interaction is limited to mechanics that are relevant to the lesson. In *Factorio* this is represented as a save file (alternatively, a game server). Removing certain gameplay elements, despite restricting the player's freedom from a gameplay point of view, is almost certainly necessary when adapting COTS games to educational settings. Features that normally make gameplay more varied, unpredictable and entertaining can detract from the educational aspect. It is important to note, that there is no "one size fits all" set of guidelines on which gameplay features should be limited in teaching: for example, during a module intended to teach capacity planning and line balancing, PvE (player versus environment, i.e. combatting naturally occurring in-game enemies) mechanics do not add any value to the lesson, and are likely to only distract and frustrate students. The same set of mechanics, however, may be useful in another module that aims to teach decision-making under uncertainty and calculated loss management— in *Factorio* enemies are denizens of the planet that the player character is turning into a factory, and they react aggressively the pollution generated by the numerous processing lines the player builds over the course of the game. Finding an optimal set-up that minimizes generated pollution while also maximizing production throughput, as well as accounting for potential loss due to enemy interference are a mapping of several important production management and operations research principles that are possible only when the lesson scenario involves game mechanics that may initially seem counterproductive to learning. All gameplay elements should be carefully considered when designing the initial game state.

The desired game state is the state of the game world that fulfills all the task criteria presented during the lesson. Where the initial game state given to a student is a literal one (a save file, world generation seed value or a server), the desired game state can be represented as a well-structured list of requirements that constitute a successfully executed task. An example of trivial desired game state could be "a throughput of at least 25 gears per second[2] on the top-most assembly line", which provides a goal for the student when paired with the initial game state. The desired game state should be seen as a frame of reference through which the teacher can assess a student's familiarity with the lesson contents. The previously mentioned example is binary (the student has either managed to achieve the goal or not) but could be expanded into a point range using different throughput values as references (e.g., 25 gears per second is 5 points, 20 per second is 4, and so on). When deciding on what constitutes a satisfactory solution, a bottom-up approach is recommended— this means finding the optimal (or sufficient) solution should be done by playing the game from its initial state until one is reached, rather than adjusting the initial state to fit a specific value. The desired game state can also be represented more literally (just like the initial state), with a save file that fulfills all task criteria.

Getting from the initial state to the desired state within the scope of play (allowed set of gameplay interactions) is the goal of a single unit lesson. In Case 1, the player uses UI elements and in-game analytics to accomplish the goal: those systems are already present as a part of the base game, and do not have to be implemented by the teacher manually.

[2] Due to hardware differences, production-management focused games like *Factorio*, *Satisfactory* and especially *Minecraft* can run at different speeds. To keep game mechanics consistent, the aforementioned games use a tick system, where the game states progress in discrete steps called "ticks", which occur multiple times a second. For example, in *Minecraft* ticks are programmed to occur at a fixed rate of 20 per second (Minecraft Wiki, 2025), but may occur less frequently on less powerful hardware— this means that the number of seconds a process takes up can vary from computer to computer, but the amount of executed ticks remains consistent. As such, game ticks are the fairest unit by which to measure timespan. Under the assumption that lessons are taught on similar hardware, and for the sake of simplicity, the framework described here uses seconds as the base unit.

3.2 Case 2: Just-in-Time production in modded Minecraft

Just-in-time (abbreviated to JIT) is a lean scheduling philosophy that aims to produce or deliver each unit in a production chain at the exact moment it is needed, reducing in-process inventory and consequently, the cost of carrying stock (Nicholas, 2018). JIT has formalized inside Toyota manufacturing plants between the late-1940s and 1970s (Nicholas, 2018), as a way to synchronize individual operations with the actual downstream consumptions. This approach helps alleviate waste and shortens lead time. Reducing work-in-process inventory helps slash costs in real-life manufacturing and is also a critical gameplay component of many production management simulation games. Modded instances of *Minecraft* offer a very customizable environment that could be used to demonstrate and teach JIT and its related methodologies— during a normal, non-educational play session the player is likely to "discover" JIT principles by themselves by noticing that creating unnecessary buffers for products often incurs some sort of cost. For example, storage compartments in *Minecraft* require the player to first collect necessary resources and then craft the storage containers (which uses up said resources in the process). Larger buffers may require more resources, which may be better spent elsewhere by the player.

Using modded *Minecraft* as the educational game base requires more input from the teacher than a game that revolves around optimizing production chains by default. In addition to the steps described in Case 1 (designing an initial and desired game state, as well as task specification), it also necessitates a careful selection of gameplay modifications to expand on the production simulation features natively offered by *Minecraft*. This process is similar to how a more education-friendly environment has been created in *Factorio* by removing certain features (e.g., player-versus-world interactions), but additive in its nature, rather than subtractive. This way, *Minecraft* becomes almost like a game engine that can be used to design very elaborate and specific lessons. The flexibility offered by *Minecaft's* numerous modding APIs makes it an attractive teaching platform. The freedom and customizability offered by modifying *Minecraft* to better suit teaching needs, however, presents additional challenges: selecting gameplay elements that are meant to be added, rather than removed, requires in-depth knowledge of how many (often very complex) game modifications work. It also requires a greater amount of technical know-how: though bundling game modifications together into one instance is relatively simple, they may not always work in tandem "out of the box" and could require additional customization (e.g., due to versioning issues, compatibility or obtuse interactions). Using modified game instances as teaching environments also opens up the possibility of designing said modifications for the specific purposes of a curriculum from scratch— this requires programming knowledge (Java) but enables even more control over how individual gameplay elements work. Case 2 uses only three gameplay mods and a single programming utility mod bundled together on the 1.21 version of *Minecraft* to simulate

JIT principles: *Mekanism, Applied Energistics 2* and *ComputerCraft* as well as *KubeJS* (enabling small compatibility tweaks to be implemented into the game with a much simpler, less advanced framework that relies on JavaScript).

The initial game state, much like in Case 1, should be a literal save (or server) for *Minecraft*. The student is presented with a small, pre-built and pre-powered[3] production line: a sink (chest), some number of always-on processing machines that convert raw materials (e.g., iron ore) into intermediate components (e.g., iron ingots) to deposit them into buffer storage, an automation network (automating product requests) and culminates in a single processor that assembles the final product (e.g., iron pickaxes). Pre-programmed display panels show current work-in-process inventory (large number of components) and a calculated average order lead-time (high, despite the sizable inventory). The student should see and identify a push system with high inventory and unsatisfactory service times. Though adding gameplay features is what makes modded *Minecraft* so appealing as a learning environment, some gameplay elements may also be removed to keep the player away from distractions, just like with *Factorio*. For example, in Case 2 the player should not be able to leave the boundaries of their factory and explore the world to obtain additional raw-material sources or processing machines.

This scenario asks the student to convert the small factory into a JIT (pull) oriented system that builds products only when an order is placed, while also maintaining a slim buffer of components in a simulation that lasts for a set amount of time with no stock-outs. The desired game state could be presented as a simple rubric, demonstrated below, including example starting metrics.

Assuming initial values of 1000 WIP components, 15 s request to delivery time, 4 processors

Grade points	**Component WIP**	**Order lead-time**
5	≤ 128 components	≤ 5 s
4	≤ 256 components	≤ 7 s
3	≤ 384 components	≤ 10 s
2	≤ 512 components	≤ 12 s
1	Any reduction from initial	Any reduction from initial

Table 1 Example desired game state rubric. Source: own work

[3] Most processors in modded *Minecraft* and *Factorio* are usually hard coded to require a constant power supply. During usual gameplay, managing the power grid is an additional gameplay challenge, but may add undesired complexity to a teaching scenario, and its inclusion in lessons should be carefully considered. For both Case 1 and Case 2, we don't take power consumption into account.

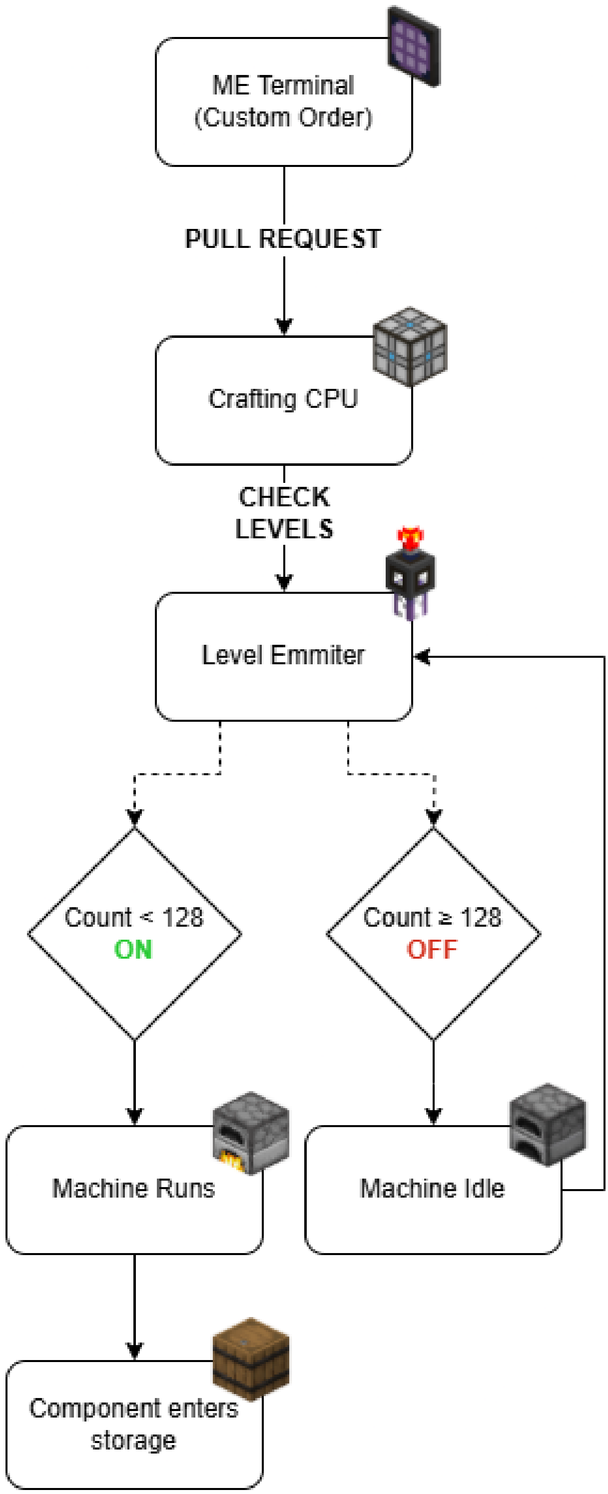

Figure 1. Viable solution chart for Case 2

The minimal viable solution requires the player to automate the assembly of the final component and installing a “kanban card” for the component storage that allows the machines to operate only when it drops below a set threshold of items. After setting up on-demand crafting, the student should notice how components only get processed as needed and everything stops afterwards. Doing this requires the student to have an understanding of pull systems and specific modded Minecraft mechanics, both of which should be supplemented by the instructor. Figure 1. Viable solution chart for Case 2 is a flowchart that demonstrates the kind of system a student should be able to achieve by the end of the lesson, working under the same initial value assumptions as the ones presented in Table 1 (example values only— could be adjusted by instructor). A similar flowchart could be a part of the scenario description, helping the students reach the desired game state, while also not providing exact answers.

4. Discussion, Future Work and Conclusions

The framework presented in this paper illustrates that COTS games can be aligned with well-established production management constructs through a structured *initial-state to scenario to desired state* task design. The two demonstrated case studies utilize popular, critically acclaimed video games as teaching platforms, and demonstrate how constraints, as well as modded features, can channel learners towards a single concept at a time, while also providing a transparent grading scheme for the teacher. From a pedagogical perspective, this approach helps address student engagement, visualization of highly abstract ideas, and encourages safe experimentation in a more constructive learning environment. Kolb’s experiential-learning cycle (Kolb, 1984) predicts that iterative problem solving inside a low-consequence “sandbox” type of environment helps facilitate the conceptual transfer from theory to practice. (Akella, 2010) COTS software can deliver many affordances that were traditionally reserved for costly and complex discrete-event simulators at a lesser organizational overhead and in a more entertaining way.

Adopting COTS games in this way is not without its caveats, however. Hardware requirements and differences may deter teachers from adopting this approach. Additionally, unless the instructors are already familiar with the games used, the teacher workload may be much greater than when using well-documented DES systems. Designing balanced save files or ensuring mod cross-compatibility also requires a degree of technical know-how. Lastly, there is the issue of scope creep: due to the open world nature of most production-centered COTS games, instructors will need to spend time carefully selecting which features to keep, and which to discard, so that the scenarios assess the students understanding of the curriculum material, rather than their prior gaming skills.

Further developing the framework and designing a set of teaching materials would require additional work, the most important of which are empirical efficacy studies conducted in controlled classroom environments. Comparisons of COTS games as teaching aid to traditional lectures or spreadsheet simulations could quantify the gain differentials and long-term knowledge retention, helping to justify (or reject) the workload required to prepare learning scenarios and materials.

The framework described in this paper is mostly theoretical, and as such elaborate technical solutions have not been discussed in detail— however, COTS games include many features that could be integrated into e-learning systems (e.g., Moodle) and help with save file distribution, telemetric data capture, automate rubric scoring and significantly reduce the required instructor overhead, while also providing rich analytics. Both cases described in the study were focused on individual production chains, assuming a distribution of one per student, but this does not have to be the case: the games used in the Methodology section have robust multiplayer features and allow hosting servers. This could be used to create a multiplayer environment where students can work in groups, rather than individually, and better illustrate how complex production chains rely on many smaller ones. Another potential avenue for further development is the co-creation of specific, bespoke game modifications between educators: with the numerous modding APIs available for games like *Minecraft* and *Factorio*, teachers could work together with mod developers to create purpose-built mechanics (e.g., visual Andon boards, SMED mini-games, etc.) that more faithfully mirror real industrial practice, while also maintaining the motivational and entertainment-related pull of mainstream game titles. Lastly, used COTS titles should be considered through the lens of accessibility— since games require a degree of mechanical skills to pilot (small as it may be), the ones used in teaching should include a set of accessibility features that ensure that a student's performance is not hindered by gameplay elements outside of their control: for example, mechanics that may unfairly disadvantage students with limited motor skills or color vision deficiency, but also those without access to high-spec hardware.

References

Abt, C. C. (1970). *Serious Games*. Viking.

Akella, D. (2010). Learning together: Kolb's experiential theory and its application. *Journal of Management & Organization - J MANAG ORGAN*, *16*, 100–112. https://doi.org/10.5172/jmo.16.1.100

Biggs, J. B. (1996). Enhancing Teaching Through Constructive Alignment. *Higher Education*, *32*(3), 347–364. https://doi.org/10.1007/BF00138871

Boardman, B. S., & Krejci, C. C. (2021). Simulation of Production and Inventory Control Using the Computer Game Factorio. *ASEE Gulf-Southwest Section Annual Conference*.

Clark, D. B., Tanner-Smith, E. E., & Killingsworth, S. S. (2016). Digital Games, Design, and Learning: A Systematic Review and Meta-Analysis. *Review of Educational Research*, *86*(1), 79–125. https://doi.org/10.3102/0034654315582065

Deterding, S., Dixon, D., Khaled, R., & Nacke, L. (2011). From Game Design Elements to Gamefulness: Defining "Gamification." *Proceedings of the 15th International Academic MindTrek Conference*, 9–15.

García-Álvarez, J., & Acevedo-Borrega, J. (2025). Minecraft as a Pedagogical Tool: A Systematic Literature Review (2014–2024). *Games and Culture*, 15554120251-341034. https://doi.org/10.1177/15554120251341034

Ierusalimschy, R., Figueiredo, L., & Celes, W. (2007). *The evolution of Lua*. https:-//doi.org/10.1145/1238844.1238846

Introduction to Logic Gates - Minecraft Education Lesson Library. (n.d.). https://education.minecraft.net/en-us/resources/introduction-logic-gates

Kolb, D. A. (1984). *Experiential Learning: Experience as the Source of Learning and Development*. Prentice-Hall.

Kovařík, M. (2013, February 1). *"Factorio" Forum thread linking Factorio's crowdfunding page*. IC^2 Forum. https://forum.industrial-craft.net/thread/8845-factorio/

Liker, J. K., & Meier, D. (2005). *The Toyota Way Fieldbook*. McGraw-Hill.

Nicholas, J. (2018). *Lean Production for Competitive Advantage*. CRC Press.

Prayaga, L., Davis, J., Whiteside, A., & Riffle, A. (2016). An Exploration in the Use of Minecraft to Teach Digital Logic to Secondary School Students. *International Journal of Computer Science, Information Technology and Management 2*(2).

Prensky, M. (2001). *Digital Game-Based Learning*. McGraw-Hill.

Şimşit, Z. T., Günay, N. S., & Vayvay, Ö. (2014). Theory of Constraints: A Literature Review. *Procedia - Social and Behavioral Sciences*, *150*, 930–936. https://doi.org-/10.1016/j.sbspro.2014.09.104

Squire, K. (2004). Replaying History: Learning World History Through Simulation Games. *International Conference of the Learning Sciences (ICLS)*. https://doi.dx-.org/10.22318/icls2004.505

Sterman, J. D. (1989). Modeling Managerial Behavior: Misperceptions of Feedback in a Dynamic Decision-Making Experiment. *Management Science*, *35*(3), 321–339. https://doi.org/10.1287/mnsc.35.3.321

Subhash, S., & Cudney, E. A. (2018). Gamified Learning in Higher Education: A Systematic Review of the Literature. *Computers in Human Behavior*, *87*, 192–206. https://doi.org/10.1016/j.chb.2018.05.028

Tick. (2025, June 7). Minecraft Wiki. https://minecraft.fandom.com/wiki/Tick

CurseForge. (2025). *Minecraft Forge Documentation.* Retrieved from the Forge API documentation: https://docs.minecraftforge.net/en/latest/

Mojang, AB. (2025). *Minecraft Annual 2026.* Farshore.

Steam, DB. (2025, 06). *Steam Database*. Retrieved from URL: https://steamdb.info/

Cyryl Leszczyński is a researcher and teacher at the Department of Information Technology, Institute of Informatics and Quantitative Economics of the Poznań University of Economics and Business. Research interests include virtual and augmented reality, artificial intelligence, and educational technology, with a focus on the design and evaluation of interactive learning systems and applied digital learning tools.