CTE Workshop Proceedings, 2025, Vol. **12**, pp. 19–62 https://doi.org/10.55056/cte.926

Game simulators as educational tools for developing algorithmic thinking skills in computer science education

Maksym S. Kovtaniuk[1], Svitlana V. Shokaliuk[2] and Alexander N. Stepanyuk[2]

[1]*Pavlo Tychyna Uman State Pedagogical University, 2 Sadova Str., Uman, 20300, Ukraine*
[2]*Kryvyi Rih State Pedagogical University, 54 Universytetskyi Ave., Kryvyi Rih, 50086, Ukraine*

Abstract. This paper presents an analysis of game simulators as educational tools for developing algorithmic thinking skills in computer science education. As computational thinking becomes increasingly important in modern education, innovative approaches to teaching programming and algorithmic concepts are essential. Game simulators offer an engaging and interactive alternative to traditional teaching methods, particularly in developing algorithmic thinking – a fundamental skill in computer science. Through a synthesis of current research and pedagogical theories, this paper examines various game simulators including Blockly Games, Rabbids Coding, Kodu Game Lab, 7 Billion Humans, and Minecraft Education Edition. We analyze their features, implementation strategies, and effectiveness in different educational contexts while providing a theoretical framework connecting gamification principles with educational psychology. The paper also addresses practical challenges in implementation and suggests directions for future research. Our findings indicate that game simulators, when effectively integrated into educational curricula, can significantly enhance student engagement, motivation, and algorithmic thinking skills across various educational levels.

Keywords: game simulators, algorithmic thinking, computer science education, computational thinking, educational technology, gamification, block-based programming, simulation-based learning, educational games, programming education, learning analytics, Minecraft Education, Blockly Games, Kodu Game Lab, cognitive development, scaffolding, assessment strategies, implementation models, virtual learning environments

1. Introduction

The digital transformation of society has fundamentally changed the way we approach education, particularly in fields related to computer science and information technology. As computational devices become ubiquitous in everyday life, the ability to think algorithmically – to devise systematic, step-by-step solutions to complex problems – has become increasingly essential [17]. Algorithmic thinking, defined as the capacity to design, implement, and assess algorithms to solve problems [58], represents a core competency not only for future computer scientists but for students across disciplines.

However, teaching algorithmic thinking presents significant challenges. Traditional approaches often rely heavily on abstract theoretical concepts that students find difficult to grasp and apply in practical scenarios [26]. The abstract nature of algorithmic

0000-0001-7059-6784 (M. S. Kovtaniuk); 0000-0003-3774-1729 (S. V. Shokaliuk); 0000-0001-9088-2294 (A. N. Stepanyuk)
maksym-kovtanyuk@udpu.edu.ua (M. S. Kovtaniuk); shokalyuk@kdpu.edu.ua (S. V. Shokaliuk); alexanderstepanyuk@gmail.com (A. N. Stepanyuk)
https://sites.google.com/udpu.edu.ua/maksym-kovtaniuk (M. S. Kovtaniuk); https://kdpu.edu.ua/personal/svshokaliuk.html (S. V. Shokaliuk); https://kdpu.edu.ua/personal/omstepaniuk.html (A. N. Stepanyuk)

concepts, combined with the technical complexity of programming languages, can create substantial barriers to learning, particularly for novice students [44]. These challenges have led educators and researchers to explore innovative pedagogical approaches that can make algorithmic thinking more accessible and engaging.

Game simulators – interactive digital environments that combine elements of gaming with educational content – have emerged as a promising approach to addressing these challenges. By leveraging the intrinsic motivation and engagement associated with games, these simulators create immersive learning experiences where students can explore and apply algorithmic concepts in contextualized, meaningful ways [26, 43]. As Ekanayake et al. [19] note, game simulators can provide safe environments for experimentation, immediate feedback, and opportunities for iterative learning – all crucial elements for developing complex cognitive skills like algorithmic thinking.

The potential of game simulators extends beyond mere engagement. Research by Spieler et al. [49] suggests that game design activities support the development of various cognitive functions underlying computational thinking, including working memory, creativity, and arithmetic skills. Similarly, Pellas and Vosinakis [44] highlights how 3D simulation games can enhance computational problem-solving practices by providing students with opportunities to express alternative solutions and rationalize their decisions in coding.

Despite growing interest in this area, several questions remain regarding the effective implementation of game simulators in educational settings. How can these tools be integrated into existing curricula? Which simulators are most effective for specific learning objectives? What pedagogical approaches best support learning with game simulators? How can educators assess learning outcomes in these environments? These questions highlight the need for a comprehensive analysis of game simulators as educational tools for developing algorithmic thinking.

This paper aims to address these questions through a detailed examination of game simulators in computer science education. Drawing on current research, pedagogical theories, and practical applications, we analyze various game simulators, their features, and their effectiveness in developing algorithmic thinking skills.

The paper is organized as follows: section 2 establishes the theoretical framework connecting algorithmic thinking, gamification, and educational psychology. Section 3 reviews the literature on game simulators in computer science education. Section 4 provides an analysis of specific game simulators, including their features and pedagogical applications. Section 5 discusses implementation strategies for different educational contexts. Section 6 examines the implications of our findings for computer science education. Section 7 outlines directions for future research, and section 8 concludes with a summary of key insights.

2. Theoretical framework

2.1. Defining algorithmic thinking and its relationship to computational thinking

Algorithmic thinking constitutes a fundamental cognitive process within the broader domain of computational thinking. As defined by Thomas et al. [59], algorithmic thinking encompasses “the ability to design, implement, and assess the implementation of algorithms to solve a range of problems”. This definition highlights the practical, problem-solving orientation of algorithmic thinking, which involves identifying problems, articulating solutions in the form of algorithms, implementing those solutions, and evaluating their effectiveness.

While algorithmic thinking focuses specifically on the design and implementation of step-by-step procedures, computational thinking encompasses a wider range of

cognitive processes. Bratitsis et al. [6] characterizes computational thinking as "a specific set of mental skills used to formulate solutions to problems by converting given inputs to produce appropriate outputs while following detailed algorithms". This broader framework includes abstraction, algorithm design, decomposition, pattern recognition, and data representation as core components.

The relationship between these two concepts is hierarchical but interdependent. Algorithmic thinking serves as a subset of computational thinking, providing the specific methodological approach through which computational problems are solved. As Bacelo and Gómez-Chacón [4] notes in their university study of unplugged activities, algorithmic thinking establishes connections between mathematical and computational working spaces across semiotic, instrumental, and discursive dimensions. This relationship underscores the importance of algorithmic thinking not merely as a programming skill but as a fundamental cognitive process that bridges abstract mathematical concepts with practical computational applications.

In educational contexts, the distinction and connection between algorithmic thinking and computational thinking have important implications. Park and Jun [42] highlights that algorithmic thinking skills are often the first step in software implementation and essential for problem-solving in digital environments. By developing algorithmic thinking, educators lay the groundwork for broader computational thinking capabilities that students can apply across various domains and problems.

2.2. Cognitive learning theories supporting game-based learning

The effectiveness of game-based learning in developing algorithmic thinking is supported by several cognitive learning theories. Constructivist theory, which emphasizes active knowledge construction through experience and reflection, provides a strong foundation for understanding how game simulators facilitate learning. As Lei [35] explains, game-based learning environments allow students to construct knowledge by engaging with interactive content, testing hypotheses, and receiving immediate feedback on their actions – all core principles of constructivist learning.

Social cognitive theory, which emphasizes learning through observation, modeling, and social interaction, also provides insights into the value of game simulators. McGaghie and Harris [38] notes that simulation-based learning creates opportunities for students to observe models of effective problem-solving, practice skills in a social context, and receive feedback from peers and instructors. These social dimensions of learning are particularly important in developing algorithmic thinking, which often involves collaborative problem-solving and knowledge sharing.

Cognitive load theory offers another perspective on the benefits of game simulators. According to Faber et al. [20], adaptive scaffolding within game-based learning environments can optimize cognitive load, allowing students to focus on key concepts without becoming overwhelmed by complexity. This scaffolding, when properly designed, enables students to tackle increasingly complex algorithmic problems while maintaining engagement and motivation.

The ARCS motivation model (Attention, Relevance, Confidence, Satisfaction) developed by Keller [29] provides additional theoretical support for game-based learning. As discussed by Lei [35], game elements can capture and sustain attention, establish relevance through authentic contexts, build confidence through progressive challenges, and generate satisfaction through achievement and recognition. These motivational factors are particularly important in developing algorithmic thinking, which requires sustained engagement with challenging problems over time.

2.3. Gamification principles in educational contexts

Gamification – the application of game elements and mechanics to non-game contexts – has gained significant traction in educational settings. Dicheva et al. [18] conducted a systematic mapping study of gamification in education, identifying key game design elements that have been successfully applied in educational contexts, including points, badges, leaderboards, progress bars, and narrative elements. These elements leverage intrinsic and extrinsic motivation to enhance student engagement with educational content.

The effectiveness of gamification in education depends on thoughtful implementation aligned with pedagogical goals. As Adams and Du Preez [1] emphasizes, successful gamification requires an understanding of design principles that support student engagement while achieving learning outcomes. Their design-based research approach revealed that gamification elements must be contextually sensitive and iteratively refined based on student responses and learning outcomes.

In the specific context of algorithmic thinking education, gamification offers several advantages. Hsu and Wang [26] found that applying game mechanics to an online puzzle-based learning system significantly enhanced algorithmic thinking skills and puzzle-solving performance. Their study also revealed that combining game mechanics with student-generated questions further improved engagement and willingness to participate, suggesting that gamification can be particularly effective when it incorporates active, constructivist learning principles.

However, gamification is not without challenges. Castillo-Parra et al. [9] notes that gamification must consider the principles and appropriate elements of the game to solve problems and promote learning effectively. Poorly designed gamification can lead to superficial engagement, excessive focus on extrinsic rewards, or distraction from learning objectives. Klock et al. [31] also highlights the importance of evaluating gamification approaches to ensure they achieve desired educational outcomes – an area that requires further research and methodological development.

2.4. The pedagogical foundations of simulation-based learning

Simulation-based learning (SBL) provides a theoretical framework for understanding how game simulators support the development of algorithmic thinking. According to Ledger et al. [34], SBL has been embedded in many disciplines to practice complex skills before applying them in real-life situations. This approach allows students to engage with authentic problems in a controlled environment, receiving immediate feedback on their actions and decisions.

The pedagogical value of simulations stems from their ability to create what Ledger et al. [34] describes as a "third space" where theory can be practiced, rehearsed, and reviewed virtually before real-world application. This bridging of theory and practice is particularly important in algorithmic thinking, where abstract concepts must be translated into concrete procedures and applications.

Simulation-based learning is supported by theories of experiential learning, which emphasize the importance of concrete experience, reflective observation, abstract conceptualization, and active experimentation in the learning process. As Rutherford-Hemming [47] notes, simulations provide structured opportunities for students to engage in this experiential learning cycle, developing both conceptual understanding and practical skills through iterative practice and reflection.

The effectiveness of simulation-based learning also depends on the fidelity or authenticity of the simulation environment. Chien, Ho and Hou [11] emphasizes the importance of immersive scenes and interactive contextual clue scaffolding in simulation-based learning, suggesting that these elements enhance learning effectiveness and reduce anxiety. This is particularly relevant for algorithmic thinking education, where

the complexity of problems can induce cognitive overload and anxiety if not properly scaffolded.

2.5. Connecting game mechanics to algorithmic thinking development

The integration of game mechanics with algorithmic thinking development represents a synergistic approach that leverages the motivational aspects of games to enhance cognitive skill development. Theodosi and Papadimitriou [57] highlight this synergy in their study "The Synergy of Three: Incorporating Games, Multimedia and Programming in Order to Improve Algorithmic Skills", demonstrating how game mechanics can scaffold the development of algorithmic thinking through progressive challenges and immediate feedback.

Specific game mechanics that support algorithmic thinking development include:

- **Progressive challenge**: gGames typically present increasingly complex problems, mirroring the progression from simple to complex algorithms in computational thinking education.
- **Iterative feedback**: immediate feedback on actions allows students to refine their algorithmic solutions based on observed outcomes.
- **Rule-based systems**: games operate according to explicit rules, paralleling the rule-based nature of algorithms.
- **Goal-oriented tasks**: clear objectives in games provide motivation and context for algorithmic problem-solving.
- **Systematic thinking**: games that require planning and strategic thinking develop the systematic approach essential for algorithmic thinking.

Renganathan et al. [46] emphasizes how game-based learning can specifically enhance algorithmic reasoning through the integration of graph theory concepts. Their approach combines theoretical knowledge with practical application, creating an educational gaming experience that develops critical cognitive skills. This integration demonstrates how specific mathematical concepts can be embedded in game mechanics to develop algorithmic thinking in context-rich environments.

The connection between game mechanics and algorithmic thinking is further strengthened by what Horn et al. [25] describes as "strategy analysis" in educational games. By analyzing the strategies players develop to solve game-based challenges, educators can gain insights into students' algorithmic thinking processes and problem-solving approaches. This analysis provides feedback for both students and educators, supporting metacognitive development and instructional refinement.

3. Literature review

3.1. Historical development of game simulators in computer science education

The use of games and simulations in education has a rich history that predates digital technology. De Freitas [15] traces the evolution of games and simulations as educational tools, noting their early applications in military and business training before their adoption in academic settings. However, the specific application of game simulators to computer science education represents a more recent development, coinciding with advances in computing technology and changing perspectives on educational methodology.

Early digital games for programming education emerged in the 1980s and 1990s, with simple environments like Logo providing students with visual feedback on programming commands. These early systems, while limited by the technology of their time, established the principle that visual, interactive environments could make programming concepts more accessible and engaging for novice learners [53].

The early 2000s saw significant advances in educational game design, with researchers and educators recognizing the potential of games to develop critical thinking and problem-solving skills. Van Eck et al. [60] describes how this period saw increasing interest in developing instructional games to engage "21st century learners" and teach problem-solving in math and science. However, as they note, the process for developing games that were both engaging and instructionally effective remained elusive.

The late 2000s marked a turning point with the development of block-based programming environments like Scratch (released in 2007) and Blockly (released in 2012). These environments, designed specifically for educational purposes, made programming more accessible by eliminating syntax errors and providing visual representations of programming concepts [39]. Concurrently, commercial games like Minecraft began to be adapted for educational purposes, leading to versions like Minecraft Education Edition that incorporated programming elements [41].

Recent years have seen a proliferation of game simulators specifically designed to develop computational and algorithmic thinking. Johnson et al. [28] describes how game development has become increasingly incorporated into computer science curricula, allowing students to learn programming concepts through the process of creating games. Similarly, Hernandez et al. [24] discusses the evolution of JGOMAS, a simulation game where students design and implement agents and strategies to practice technologies related to the multi-agent paradigm.

This historical trajectory reflects broader trends in educational technology and computer science education. As Guarda and Díaz-Nafría [22] notes, simulators have emerged as powerful knowledge transfer tools across multiple disciplines, with their educational applications expanding as technology advances. The integration of game design principles with educational objectives represents a synthesis of entertainment and learning that continues to evolve with technological capabilities and pedagogical understanding.

3.2. Current research trends in using game simulators for teaching programming

Current research on game simulators for teaching programming reveals several emerging trends that highlight both the potential and challenges of this approach. One significant trend is the focus on developing and evaluating game simulators that target specific programming concepts and skills. Hsu and Wang [26] examined the effects of using game mechanics and student-generated questions to promote algorithmic thinking skills through an online puzzle-based game learning system. Their findings indicated that game mechanics significantly enhanced algorithmic thinking skills and puzzle-solving performance, while the addition of student-generated questions further improved engagement and participation.

Another important trend is the exploration of different game types and formats for programming education. Pellas and Vosinakis [44] investigated the use of 3D simulation games for developing computational thinking through a game playing approach. Their research highlighted the value of immersive, three-dimensional environments in helping students express alternative solutions and rationalize their decisions in coding. Similarly, Taylor et al. [56] presented IntelliBlox, a toolkit for integrating block-based programming into game-based learning environments, demonstrating how block-based programming challenges can be embedded in immersive game environments.

Research is also increasingly examining the cognitive processes and skills developed through game-based programming education. Spieler et al. [49] investigated the relationship between computational thinking and cognitive skills in the context of game design activities. Their findings suggest that different concepts of computational

thinking applied in games correlate with separate cognitive measures, indicating that computational thinking is not a homogeneous skill but a set of subskills that load on different cognitive functions.

Assessment and evaluation of learning in game-based programming environments represent another significant research trend. Stahlbauer, Kreis and Fraser [50] introduced a formal testing framework for Scratch programs, allowing for automated and property-based testing of student work. This approach achieved an average of 95.25% code coverage in testing student programs, demonstrating the potential for automated assessment methods to support both students and teachers in programming education.

Finally, research is increasingly focused on accessibility and inclusivity in game-based programming education. Milne, Baker and Ladner [39] described Blocks4All, a blocks-based programming environment designed to be accessible for blind children. Their work addresses the limitations of visual block-based environments like Scratch and Blockly, which rely heavily on visual metaphors and interactions that can exclude visually impaired learners.

3.3. International perspectives and approaches

The adoption and implementation of game simulators for developing algorithmic thinking skills vary significantly across international contexts, reflecting cultural, educational, and technological differences. Research by Lean et al. [33] in the United Kingdom revealed significant levels of use of both computer and non-computer-based simulations and games in higher education. Their study identified the availability of resources as a perceived barrier to teaching with simulations, though further analysis suggested that views on the suitability of and risk attached to simulation-based learning methods were more influential factors in adoption decisions.

In Spain, Cossío-Silva, Vega-Vázquez and Revilla-Camacho [13] documented a growing tendency to use simulation games in universities, driven by educational reforms associated with the European Higher Education Area. Their study of the Quantum marketing simulator at Carlos III University in Madrid found that students' global assessment of simulators was conditioned by their prior motivation and perception of the simulator's impact on skill acquisition. This highlights the importance of student attitudes and perceptions in the successful implementation of game simulators.

Research from Ukraine by Chornous et al. [12] explored the implementation of ERP simulation games in economic education, proposing models for applying these games in the framework of courses related to quantitative methods of decision-making support. Their work demonstrates how simulation games can be adapted to specific disciplinary contexts and learning objectives, emphasizing the importance of context-sensitive implementation.

In Southeast Asia, Ng [40] investigated gender differences in the use of educational games for learning programming at a tertiary level. Their findings revealed that female participants spent more time in the simulator but achieved lower scores, suggesting the need for gender-sensitive approaches to game design and implementation. This research highlights the importance of considering diversity and inclusivity in the design and deployment of educational game simulators.

Cross-cultural studies like those conducted by Campos et al. [7] have examined simulation-based education involving online and on-campus models in different European universities. Their research emphasizes the role of e-learning practices in making educational experiences available to students from different geographical regions and universities, promoting international and inter-university cooperation in education.

These international perspectives reveal both commonalities and differences in ap-

proaches to using game simulators for algorithmic thinking development. Common themes include the recognition of simulations' potential to enhance engagement and practical skills, concerns about resource requirements and implementation challenges, and interest in evaluation and assessment methodologies. Differences emerge in the specific applications, institutional support, and integration with existing educational frameworks across different national contexts.

3.4. Identified benefits and challenges from existing research

Existing research has identified numerous benefits and challenges associated with using game simulators for developing algorithmic thinking skills. Understanding these factors is crucial for effective implementation and evaluation of game-based approaches in computer science education.

Benefits:

1. *Enhanced engagement and motivation*
   A consistently reported benefit of game simulators is their ability to enhance student engagement and motivation. Hsu and Wang [26] found that incorporating game mechanics into a puzzle-based learning system not only improved algorithmic thinking skills but also increased students' willingness to participate. Similarly, Jivani et al. [27] reported that gamification positively impacted student engagement and academic performance in undergraduate engineering education, with game elements like Kahoot and leaderboard systems effectively capturing students' attention and facilitating a dynamic learning environment.
2. *Development of specific cognitive skills*
   Game simulators have been shown to develop specific cognitive skills associated with algorithmic thinking. Spieler et al. [49] found correlations between different gaming and design elements in students' projects and specific cognitive measures. For instance, the number of design elements (shape, structure, sound, visual design) correlated with working memory and arithmetic skills, while game elements (interactivity, mechanics, dynamics, aesthetics) correlated with creativity. This suggests that different aspects of game design and implementation engage distinct cognitive processes.
3. *Safe environment for experimentation*
   Game simulators provide a safe environment for students to experiment with algorithmic concepts without fear of real-world consequences. As Ekanayake et al. [19] notes, this allows students to learn from mistakes and refine their understanding through iterative trial and error. This protected space for experimentation is particularly valuable for developing algorithmic thinking, which often requires multiple attempts and refinements to solve complex problems effectively.
4. *Immediate feedback and adaptive learning*
   Many game simulators provide immediate feedback on student actions, allowing for rapid iteration and learning. Faber et al. [20] investigated the effects of adaptive scaffolding on performance, cognitive load, and engagement in game-based learning, finding that tailored scaffolding improved speed, reduced self-regulation, and lowered cognitive load. This immediate feedback loop accelerates the learning process and helps students identify and correct misconceptions quickly.
5. *Transfer of skills to real-world contexts*
   Research by Guarda and Díaz-Nafría [22] suggests that simulators can effectively transfer knowledge across various domains, including medicine, aviation, and engineering. In the context of algorithmic thinking, this means that skills

developed through game simulators may transfer to real-world programming and problem-solving contexts, though the extent and conditions of this transfer require further investigation.

Challenges:

1. *Resource constraints*
   Resource limitations represent a significant challenge for implementing game simulators in educational settings. Lean et al. [33] identified the availability of resources as a commonly perceived barrier to teaching with simulations, including constraints related to time, technology, and expertise. These resource constraints can limit access to high-quality game simulators, particularly in under-resourced educational contexts.
2. *Technical and usability issues*
   Technical challenges and usability issues can impede the effective implementation of game simulators. Alhumairi et al. [3] reported mixed reactions to VR interfaces in computer science education, with some students finding the technology advanced but others criticizing it as unrealistic or non-interactive. Issues like motion sickness in VR environments or difficulties navigating virtual spaces can detract from the learning experience and create barriers to engagement.
3. *Assessment and evaluation*
   Assessing and evaluating student learning in game-based environments presents methodological challenges. While Stahlbauer, Kreis and Fraser [50] demonstrated the potential of automated testing frameworks for Scratch programs, measuring the development of algorithmic thinking skills through game simulators often requires nuanced approaches that go beyond traditional assessment methods. Fanfarelli [21] explored different assessment methods for computational thinking in serious games, highlighting the need for multiple methods including questionnaires, think-aloud testing, and automated data logging.
4. *Teacher preparation and support*
   The effective implementation of game simulators requires adequate teacher preparation and support. Alhumairi et al. [3] noted that while setting up VR equipment might be straightforward, instructors needed guidance on integrating the technology effectively into their teaching. Without appropriate professional development and ongoing support, teachers may struggle to implement game-based approaches effectively, limiting their potential impact on student learning.
5. *Balancing entertainment and educational value*
   Striking the right balance between entertainment and educational value represents a persistent challenge in educational game design. Johnson et al. [28] discussed the difficulty of developing games that are both engaging and instructionally effective, highlighting the need for interdisciplinary approaches drawing on game design, instructional design, and subject matter expertise. If games prioritize entertainment at the expense of learning objectives, or emphasize educational content at the cost of engagement, their effectiveness as learning tools may be compromised.

Understanding these benefits and challenges provides a foundation for developing more effective approaches to implementing game simulators in computer science education.

3.5. Gaps in current research

Despite the body of research on game simulators for developing algorithmic thinking skills, several significant gaps remain in our understanding of this field. Identifying

these gaps is essential for guiding future research efforts and improving educational practices.

1. *Long-term effectiveness*
   A notable gap in current research is the limited evidence regarding the long-term effectiveness of game simulators in developing algorithmic thinking skills. While many studies demonstrate immediate improvements in engagement, motivation, and specific skills, fewer investigations track the persistence of these gains over extended periods. As Gutiérrez, Dávila and Quintana [23] notes in their study on serious games for computational thinking, many studies use pre-experimental designs with limited experimentation periods, constraining the development of conclusions about long-term effectiveness. Longitudinal studies tracking skill development and retention over multiple years would provide valuable insights into the durability of learning through game simulators.
2. *Transfer to non-game contexts*
   Research on the transfer of algorithmic thinking skills from game environments to non-game contexts remains limited. While games may effectively teach concepts within their structured environments, questions persist about how well these skills transfer to traditional programming tasks, problem-solving in other domains, or real-world applications. Varghese and Renumol [61] highlights the need for more research on the assessment of computational thinking using video games, noting a lack of empirical evidence to prove their effectiveness. Studies examining skill transfer across diverse contexts would enhance our understanding of how game-based learning translates to broader competencies.
3. *Inclusive design and accessibility*
   Although some research addresses accessibility issues in game simulators, significant gaps remain in understanding how to design inclusive learning experiences that accommodate diverse needs. As Milne, Baker and Ladner [39] notes, many block-based programming environments rely heavily on visual metaphors and interactions, making them inaccessible for visually impaired learners. Research on designing accessible game simulators for algorithmic thinking development – considering visual, auditory, cognitive, and motor diversity – remains limited, highlighting a critical area for future investigation.
4. *Cultural and contextual factors*
   The influence of cultural and contextual factors on the effectiveness of game simulators is inadequately explored in current research. Educational approaches that succeed in one cultural context may not translate effectively to others due to differences in educational traditions, technological infrastructure, and cultural attitudes toward games and learning. Pellas [43] notes differences in game experiences and satisfaction among students using simulation games, but deeper analysis of how cultural and contextual factors influence these experiences is needed to develop culturally responsive approaches to game-based algorithmic thinking education.
5. *Integration with broader curricula*
   Research on effectively integrating game simulators into broader computer science curricula remains limited. While many studies examine the impact of specific games or simulations in isolation, fewer investigate how these tools can be systematically incorporated into comprehensive educational programs. Boulden et al. [5] discusses the integration of a Use-Modify-Create scaffolding framework into a game-based learning environment, but broader questions about curricular integration across different educational levels and settings require further investigation.

6. *Methodological diversity*
   Current research on game simulators for algorithmic thinking development often relies on similar methodological approaches, potentially limiting the comprehensiveness of our understanding. Fanfarelli [21] advocates for combining multiple assessment methods, including standardized questionnaires, think-aloud testing, and automated data logging, to achieve a more granular understanding of learning processes and outcomes. Greater methodological diversity – incorporating mixed methods, participatory research, design-based research, and learning analytics – would provide richer insights into the complex processes of learning through game simulators.

Addressing these gaps through focused research initiatives would significantly advance our understanding of how game simulators can effectively support the development of algorithmic thinking skills across diverse educational contexts.

4. Analysis of game simulators for algorithmic thinking

4.1. Block-Based Programming Environments

Block-based programming environments represent a significant innovation in computer science education, offering visual, intuitive interfaces that eliminate syntax errors and allow students to focus on algorithmic concepts. These environments use draggable, interlocking blocks that represent programming constructs, allowing students to create programs by assembling these blocks into logical sequences.

4.1.1. Blockly Games

Blockly Games, developed by Google, provides a series of educational games designed to teach programming concepts through a visual block-based interface. As described by Milne, Baker and Ladner [39], Blockly serves as the foundation for many block-based programming environments, including Google's own educational games and applications like MIT's App Inventor.

The platform features progressive educational games that introduce programming concepts incrementally, allowing students to build understanding through structured challenges. Each game focuses on specific concepts, from basic sequencing in "Puzzle" to loops and conditionals in "Maze" and "Bird", to more complex logical operations in "Turtle" and "Movie".

One of the key strengths of Blockly Games is its accessibility for novice programmers. Zinchenko [63] highlights how Blockly Games offers an interactive and practical approach to programming, making abstract concepts tangible through visual representation. The environment allows learners to transition from basic structures to more complex concepts as they progress through the games.

However, accessibility challenges remain for certain user groups. Caraco et al. [8] notes that programs built with the Blockly library are currently not accessible for people with visual impairments, as they rely heavily on visual metaphors and interactions. This limitation highlights the need for more inclusive design approaches in block-based programming environments.

From an algorithmic thinking perspective, Blockly Games excels at teaching fundamental concepts like sequencing, iteration, and conditional logic. The visual nature of the blocks helps students understand the structure and flow of algorithms without being distracted by syntax details. The immediate visual feedback provided by the games allows students to see the results of their algorithms in action, reinforcing the connection between code and behavior.

4.1.2. Scratch and related platforms

Scratch, developed by MIT's Media Lab, represents one of the most widely used block-based programming environments in educational settings. With more than 90 million registered users as reported by Deiner et al. [16], Scratch has established itself as a foundational platform for introducing programming concepts to young learners.

Unlike Blockly Games' structured approach, Scratch emphasizes creative expression and open-ended exploration. Students can create interactive stories, games, and animations by assembling blocks that control sprite characters on a stage. This open-ended nature encourages experimentation and personal expression, making programming accessible and engaging for diverse learners.

Scratch incorporates several features that specifically support algorithmic thinking development. The platform allows for event-driven programming, where blocks execute in response to specific triggers, helping students understand cause-and-effect relationships in code. The ability to create and control multiple sprites simultaneously introduces concepts of parallel execution and coordination between program elements.

Recent developments in Scratch assessment tools enhance its educational value. Stahlbauer, Kreis and Fraser [51] introduced an automated testing framework for Scratch programs that can identify functional errors and provide feedback on code quality. Their Whisker tool achieved an average of 95.25% code coverage in testing student programs, demonstrating the potential for automated assessment to support both students and teachers in identifying and addressing misconceptions.

Despite its advantages, Scratch shares similar accessibility limitations as Blockly Games. Milne, Baker and Ladner [39] notes that block-based environments like Scratch rely heavily on visual interactions that can exclude learners with visual impairments. Efforts to address these limitations include projects like "Blocks4All", which aims to make block-based programming accessible through touchscreen interfaces and screen readers.

The pedagogical approach of Scratch aligns well with constructionist learning theories, which emphasize learning through creating personally meaningful projects. This approach, as Da Silva and Falcão [14] suggests, can make algorithmic concepts more intuitive by embedding them in creative contexts that students find engaging and relevant.

4.2. Narrative-driven programming games

Narrative-driven programming games integrate algorithmic thinking development within compelling story frameworks, using narrative elements to contextualize programming challenges and motivate student engagement. These games leverage storytelling to make abstract computational concepts more accessible and meaningful.

4.2.1. Rabbids Coding

Rabbids Coding, developed by Ubisoft, represents an innovative approach to teaching programming concepts through narrative engagement. As described by Kovtaniuk [32], this educational game features the "Rabbids" characters from Ubisoft's franchise, introducing players to coding concepts through a series of puzzles and challenges embedded within a humorous narrative context.

The game employs a block-based coding interface that makes it accessible to beginners, allowing players to create sequences of instructions that guide the Rabbid characters through various game environments. Each level presents specific challenges that require players to apply different programming concepts, from basic sequencing to loops and conditional statements.

The narrative framework of Rabbids Coding serves several pedagogical functions. First, it provides context and motivation for solving programming challenges, as players

work to achieve specific goals within the game's storyline. Second, the humorous and playful nature of the Rabbids characters helps reduce anxiety around programming, creating a low-stress environment for experimentation and learning. Finally, the narrative progression creates a sense of accomplishment and forward momentum as players advance through increasingly complex challenges.

From an algorithmic thinking perspective, Rabbids Coding excels at helping students visualize the execution of code through the actions of the game characters. This visual representation makes abstract concepts like sequential execution, looping, and conditional branching more concrete and understandable. The immediate feedback provided when Rabbids follow (or fail to follow) the programmed instructions helps students develop debugging skills and algorithmic precision.

4.2.2. 7 Billion Humans

7 Billion Humans, developed by Tomorrow Corporation, offers a more sophisticated narrative-driven approach to algorithmic thinking development. As noted by Kovtaniuk [32], this puzzle game unfolds in a futuristic world where players program and manage office workers using a simplified programming language.

Unlike many educational programming games that target beginners, 7 Billion Humans presents increasingly complex algorithmic challenges that can engage even experienced programmers. The game requires players to write algorithms that control multiple workers simultaneously, introducing concepts like parallelism, synchronization, and optimization that are rarely addressed in introductory programming environments.

The dystopian narrative framework of 7 Billion Humans provides both context and motivation for solving increasingly difficult algorithmic puzzles. Players must optimize the performance of their human workforce through efficient algorithms, with each level introducing new constraints and objectives. This narrative framing transforms abstract optimization problems into concrete scenarios with visible consequences.

From a pedagogical perspective, 7 Billion Humans excels at developing advanced algorithmic thinking skills. The need to coordinate multiple actors simultaneously encourages systematic thinking about parallel execution and resource allocation. The optimization challenges, which often have multiple valid solutions of varying efficiency, develop algorithmic analysis skills and an appreciation for the trade-offs between different approaches.

As Chen, Dowling and Goetz [10] notes in their analysis of narrative-driven games, the rules and game mechanics in such environments can perform narrative semiotic functions, offering a ludic grammar of interactive storytelling. In 7 Billion Humans, the programming constraints and optimization challenges serve both narrative and educational functions, creating an immersive learning environment that develops sophisticated algorithmic thinking skills.

4.3. Open world programming environments

Open world programming environments represent a distinct approach to developing algorithmic thinking, offering expansive virtual spaces where learners can apply programming concepts in creative, self-directed ways. These environments combine the engagement of open-world gameplay with opportunities for authentic programming practice.

4.3.1. Minecraft Education Edition

Minecraft Education Edition has emerged as one of the most widely adopted open-world programming environments in educational settings. Based on the popular sandbox game Minecraft, this educational version incorporates specific features designed to support learning across various disciplines, including computer science and

algorithmic thinking.

As described by Nguyen, Nguyen and Nguyen [41], Minecraft Education Edition promotes learning of basic programming concepts in an immersive digital environment. While playing the game, students use coding to address various challenges in the Minecraft world, applying computational thinking skills in a familiar and engaging context. The educational value of this approach was demonstrated in their study of STEM-oriented teaching with Minecraft, which showed how the platform can promote creativity, collaboration, and problem-solving in an immersive environment.

Minecraft Education Edition supports multiple programming interfaces, accommodating diverse student needs and skill levels. Liang, Kang and Mendoza [36] notes that programming in the Minecraft world can be done using either block-based programming or text-based languages like Python. This flexibility allows for scaffolded learning experiences, where students can transition from visual block-based programming to text-based coding as their skills develop.

The open-world nature of Minecraft creates unique opportunities for algorithmic thinking development. Rather than presenting predefined challenges with single correct solutions, the environment allows students to define their own projects and goals, requiring them to decompose complex problems, design algorithmic solutions, and implement these solutions through code. This approach aligns with constructivist learning theories, emphasizing active knowledge construction through authentic problem-solving.

Research by Klimova, Sajben and Lovaszova [30] examined the effectiveness of Minecraft: Education Edition for programming education through an online programming contest for lower-secondary school students. Their study analyzed ten programming tasks covering standards of the Slovak Informatics curriculum, including problem analysis, sequence of commands, loops, debugging, and error correction. The results demonstrated that Minecraft can effectively support the development of these algorithmic thinking skills in educational contexts.

Sánchez-López, Roig-Vila and Pérez-Rodríguez [54] provides a deeper analysis of Minecraft's educational potential, describing it as a pioneering case of the metaverse in immersive digital learning. Their research highlights how Minecraft Education reinforces pre-existing aspects from the physical world while adding differential components like avatars, multimodal literacy, and game mechanics that boost creativity. These elements collectively create a rich environment for developing and applying algorithmic thinking skills in contextualized, meaningful ways.

4.3.2. Kodu Game Lab

Kodu Game Lab, developed by Microsoft Research, offers a distinct approach to open-world programming, focusing specifically on game design and development as a context for algorithmic thinking. As described by Stolee and Fristoe [52], Kodu uses an entirely event-driven programming model based on "when-do" clauses, providing a non-traditional but powerful approach to computational thinking development.

Unlike many educational programming environments that emphasize text-based or block-based syntax, Kodu employs a visual rule-based system where programming takes the form of condition-action pairs. This approach makes programming accessible to novice learners while still allowing for the expression of sophisticated computational concepts. Stolee and Fristoe [52] analyzed 346 Kodu programs created by users and found that most exhibited significant sophistication through complex control flow and boolean logic, demonstrating the environment's capacity to support advanced algorithmic thinking despite its simplified interface.

Kovtaniuk [32] highlights Kodu's accessibility for users with limited programming experience, noting that it serves as an excellent tool for introducing game design

and programming logic in educational contexts. The environment allows users to create characters, landscapes, and rules using a simple interface based on icons, encouraging creativity and personalization while developing algorithmic thinking skills.

From a pedagogical perspective, Kodu exemplifies what MacLaurin [37] describes as a programming language designed specifically for young children to learn through independent exploration. The environment seeks to lower barriers to entry for new programmers by presenting a radically simplified programming model that nevertheless has significant expressive power. This balance between accessibility and capability makes Kodu particularly effective for introducing algorithmic thinking concepts to young learners.

The open-world nature of Kodu encourages creative exploration and experimentation, allowing students to discover computational concepts through self-directed projects rather than structured tutorials. This approach aligns with constructionist learning theories, which emphasize learning through creating personally meaningful artifacts. By enabling students to design and implement their own games, Kodu provides authentic contexts for applying algorithmic thinking skills and seeing the concrete results of their programming decisions.

4.4. Comparative analysis

A comprehensive comparison of the various game simulators reveals distinct patterns in their features, capabilities, and pedagogical approaches. This analysis examines these environments across five key dimensions: features and capabilities, programming concepts covered, user interface and accessibility, assessment mechanisms, and alignment with curriculum standards.

Table 1
Comparative analysis of game simulators for algorithmic thinking development.

Feature	Block-based environments	Narrative-driven games	Minecraft Education	Kodu Game Lab
Core approach	Visual programming through connected blocks	Programming challenges embedded in narratives	Open-world creation with programming tools	Visual rule-based game creation
Programming paradigm	Procedural, event-driven	Procedural, imperative	Multiple paradigms (blocks and text)	Event-driven, rule-based
Target audience	Beginners to intermediate	Beginners to advanced	Diverse skill levels	Beginners to intermediate
Creative freedom	Moderate to high	Low to moderate	Very high	High
Structured guidance	High in Blockly Games, moderate in Scratch	High, progressive challenges	Low to moderate	Low
Collaborative features	Moderate	Low	High	Low to moderate
Assessment tools	Automated testing available	In-game performance metrics	Limited formal assessment	Limited formal assessment

4.4.1. Features and capabilities

Block-based environments like Blockly Games and Scratch excel in providing intuitive, visual representations of programming concepts that eliminate syntax barriers for beginners. Their drag-and-drop interfaces allow learners to focus on algorithmic logic rather than syntax details. Scratch in particular emphasizes creative expression and supports a wide range of project types, from animations to interactive games.

Narrative-driven games like Rabbids Coding and 7 Billion Humans leverage storytelling to contextualize programming challenges, providing clear goals and motivation through narrative progression. 7 Billion Humans stands out for its focus on algorithmic optimization and parallel processing, addressing more advanced concepts than many educational programming environments.

Minecraft Education Edition offers an expansive open world where programming becomes a tool for creative expression and problem-solving. Its support for both block-based and text-based programming allows for differentiated instruction and skill progression. The collaborative multiplayer environment enables team-based projects and peer learning experiences.

Kodu Game Lab's distinctive rule-based programming approach represents a unique alternative to traditional programming paradigms. Its focus on game design provides an authentic context for algorithmic thinking, while its simplified interface makes it accessible to young learners with limited technical experience.

4.4.2. Programming concepts covered

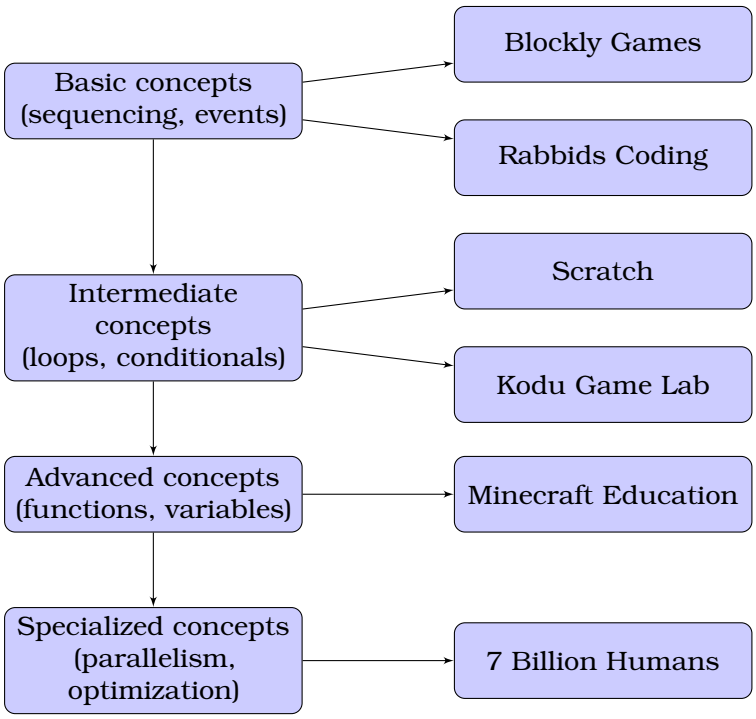

Figure 1: Progression of programming concepts across game simulators.

Each simulator covers a different range of programming concepts. Block-based environments like Blockly Games and Scratch effectively introduce fundamental concepts like sequencing, events, loops, and conditionals. Scratch extends to more advanced concepts including variables, functions, and simple data structures through list blocks.

Narrative-driven games vary in their conceptual coverage. Rabbids Coding focuses primarily on basic sequencing, loops, and conditionals, making it suitable for begin-

ners. 7 Billion Humans, however, introduces more specialized concepts like parallel processing, synchronization, and algorithm optimization, addressing areas rarely covered in introductory programming environments.

Minecraft Education Edition provides comprehensive coverage across basic to advanced programming concepts. Through its support for both block-based and text-based programming (Python), it can address concepts from simple sequencing to complex data structures and algorithms. The open-world environment allows for application of these concepts in diverse contexts, from simple automation to complex simulations.

Kodu Game Lab emphasizes event-driven programming through its "when-do" rule structure. While its syntax is simplified, Stolee and Fristoe [52] found that it effectively supports concepts like boolean logic, conditionals, and even complex control flow patterns. Its focus on game design introduces domain-specific concepts like collision detection, scoring systems, and game state management.

4.4.3. User interface and accessibility

The user interfaces of these simulators vary significantly in their design approaches and accessibility considerations. Block-based environments provide visual, drag-and-drop interfaces that eliminate syntax errors but rely heavily on visual interactions that may exclude visually impaired users. Recent efforts by Caraco et al. [8] and Milne, Baker and Ladner [39] seek to address these limitations through touchscreen interfaces and screen reader compatibility.

Narrative-driven games typically feature polished, game-like interfaces designed to engage users through visual appeal and narrative elements. These interfaces may prioritize engagement over accessibility, potentially creating barriers for users with visual, motor, or cognitive disabilities.

Minecraft Education Edition offers a complex 3D interface that can present a steeper learning curve than other environments but provides immersive spatial interactions once mastered. The environment's support for collaborative multiplayer experiences creates opportunities for peer support and differentiated instruction.

Kodu Game Lab features a simplified 3D interface with icon-based programming, reducing textual requirements but maintaining visual dependencies. Its controller-friendly design (originally developed for Xbox) can improve accessibility for users with certain motor limitations but may present challenges for others.

4.4.4. Assessment mechanisms

Assessment capabilities vary widely across these simulators. Block-based environments, particularly Scratch, benefit from emerging assessment tools like Whisker [51], which enables automated testing and evaluation of student programs. These tools can identify functional errors, measure code coverage, and provide structured feedback on program quality.

Narrative-driven games typically include built-in assessment through game progression and performance metrics. Success in completing challenges serves as a proxy for skill development, though these measures may not align perfectly with educational objectives beyond the game context.

Minecraft Education Edition offers limited formal assessment tools, though Klimova, Sajben and Lovaszova [30] demonstrates how structured challenges and contests can be used to assess specific programming skills. The open-ended nature of the environment means that assessment often relies on project-based evaluation rather than standardized metrics.

Kodu Game Lab similarly lacks formal assessment mechanisms, with evaluation typically based on the quality and complexity of student-created games. This ap-

proach aligns with constructionist learning theories but may present challenges for standardized assessment and skill certification.

4.4.5. Alignment with curriculum standards

The alignment of these simulators with curriculum standards varies by educational context and implementation approach. Block-based environments like Blockly Games and Scratch align well with introductory programming standards across various curricula, particularly those emphasizing computational thinking development. Milne, Baker and Ladner [39] notes that Blockly is increasingly used in formal and informal curriculum contexts, including Code.org's Hour of Code projects.

Narrative-driven games may require more intentional alignment with curriculum standards, as their design prioritizes engagement and specific skill development over comprehensive curriculum coverage. However, as Hsu and Wang [26] demonstrates, these environments can effectively develop algorithmic thinking skills that align with broader computational thinking objectives.

Minecraft Education Edition has been explicitly designed with curricular alignment in mind. Klimova, Sajben and Lovaszova [30] demonstrates how Minecraft can address specific standards in informatics curricula, including programming constructs, problem analysis, and debugging. The environment's flexibility allows for adaptation to various national and regional curriculum frameworks.

Kodu Game Lab's alignment with curriculum standards depends largely on implementation approach. While not designed specifically for curricular alignment, Stolee and Fristoe [52] shows how it can effectively develop computational concepts that align with computer science education objectives. Its focus on game design may also support interdisciplinary learning across art, design, and computer science standards.

This comparative analysis reveals both commonalities and differences across these game simulators. Each environment offers distinct advantages for developing algorithmic thinking skills, with their effectiveness depending on factors including student age, prior experience, learning objectives, and implementation context.

5. Implementation strategies

Effective implementation of game simulators for algorithmic thinking development requires thoughtful planning, appropriate scaffolding, and alignment with educational objectives. This section explores strategies for integrating these tools into various educational contexts, considering different educational levels, diverse learner needs, assessment approaches, and practical implementation challenges.

5.1. Integration models for different educational levels

The integration of game simulators into computer science education requires differentiated approaches for different educational levels, considering the cognitive development, prior knowledge, and learning objectives characteristic of each stage.

5.1.1. Primary education

At the primary education level (ages 6-11), implementation strategies should emphasize playful exploration, concrete representations of abstract concepts, and short, engaging activities that maintain student interest. Pellas [43] found that incorporating interactive environments for game-based instruction has significant potential to support the development of computational thinking and programming skills in primary education students.

For this age group, block-based environments like Blockly Games and narrative-driven games like Rabbids Coding are particularly appropriate. These environments provide visual, concrete representations of programming concepts that align with

the concrete operational stage of cognitive development typical of primary students. Implementation strategies might include:

- Short, focused sessions (15-30 minutes) that introduce single concepts through guided exploration.
- Pair programming approaches where students collaborate to solve simple challenges.
- Teacher-led demonstrations followed by supervised independent exploration.
- Connection of programming activities to familiar contexts and experiences.
- Celebration of creative expression and multiple solution paths.

Takbiri, Bastanfard and Amini [55] demonstrated the effectiveness of a gamified approach for improving the learning performance of K-6 students, finding significant improvement in learning skills performance after comparing initial and final sessions. Their model incorporated easter egg elements to trigger curiosity and encourage exploration, an approach that aligns well with the developmental characteristics of primary students.

5.1.2. Secondary education

At the secondary education level (ages 12-18), implementation strategies can build on students' developing capacity for abstract thinking, increased autonomy, and interest in real-world applications. For this age group, more complex environments like Minecraft Education Edition, Kodu Game Lab, and 7 Billion Humans can provide appropriate challenges and opportunities for creative expression.

Theodosi and Papadimitriou [57] demonstrated an effective approach for secondary students, incorporating games, multimedia, and programming to improve algorithmic skills. Their method involved two types of projects: reconstructing existing games and creating new games based on specified rules and behaviors. This approach engaged students in both analytical and creative thinking, developing algorithmic skills through authentic design challenges.

Implementation strategies for secondary education might include:

- Project-based learning approaches where students design and implement solutions to complex problems.
- Integration of programming activities with content from other subject areas (science, mathematics, social studies).
- Competitions and challenges that motivate engagement and provide opportunities for peer learning.
- Progression from structured guidance to independent exploration as skills develop.
- Focus on developing metacognitive skills through reflection on problem-solving strategies.

Pellas and Vosinakis [44] found that junior high school students working with a 3D simulation game had a greater range of expressing alternative solutions in blended instruction. The instructor's feedback and guidance facilitated students' ability to rationalize decisions related to computational practices in coding, highlighting the importance of teacher guidance even as students develop greater autonomy.

5.1.3. Higher education

At the higher education level, implementation strategies can focus on advanced algorithmic concepts, professional applications, and interdisciplinary connections.

While game simulators are less commonly associated with university-level computer science education, they can provide valuable learning experiences, particularly for introductory courses and non-computer science majors.

Hernandez et al. [24] describes the implementation of JGOMAS 2.0, a capture-the-flag game using Jason agents, in higher education computer science curricula. This environment allows students to practice different technologies related to the multi-agent paradigm, including coordination, cooperation, and decision-making. Students design and implement different types of agents and strategies to win the game, developing both technical skills and strategic thinking.

Implementation strategies for higher education might include:

- Integration of game-based learning with traditional programming instruction, using games to illustrate complex concepts.
- Adaptation of commercial games for educational purposes, analyzing and modifying existing game systems.
- Student-led game development projects that apply programming concepts to create educational games for younger learners.
- Critical analysis of game mechanics and algorithms, examining how computational principles are applied in game design.
- Research projects investigating the design and effectiveness of educational game simulators.

Cossío-Silva, Vega-Vázquez and Revilla-Camacho [13] found that the use of simulation games in higher education can effectively develop capacities and skills related to entrepreneurial spirit, teamwork, and competitiveness. Their study of the Quantum marketing simulator revealed that students' global valuation of the experience was conditioned by prior motivation and their perception of the simulator's impact on skill acquisition, highlighting the importance of aligning game-based approaches with students' professional goals and interests.

5.2. Scaffolding approaches for diverse learner needs

Effective implementation of game simulators requires appropriate scaffolding to support diverse learner needs, including differences in prior knowledge, learning styles, cognitive abilities, and physical or sensory capabilities. Thoughtful scaffolding can make game-based learning more accessible and effective for all students.

5.2.1. Cognitive scaffolding

Cognitive scaffolding supports students in developing the mental models and strategies needed for algorithmic thinking. Chien, Ho and Hou [11] demonstrated the effectiveness of integrating immersive scenes and interactive contextual clue scaffolding into decision-making training games. Their approach allowed learners to experience a high level of game fidelity while reducing learning anxiety, maintaining motivation, and improving learning effectiveness.

Effective cognitive scaffolding strategies include:

- Decomposing complex tasks into manageable steps with clear progression.
- Providing worked examples that demonstrate problem-solving processes.
- Offering hints and prompts that guide students toward solutions without providing answers.
- Visualizing algorithm execution through animations or simulations.
- Gradually removing scaffolds as students develop greater competence and confidence.

Faber et al. [20] investigated adaptive scaffolding based on interaction trace data, finding that tailored scaffolding improved speed, reduced self-regulation demands, and lowered cognitive load compared to non-tailored approaches. This suggests that adaptive systems that adjust scaffolding based on student performance can effectively support diverse cognitive needs.

5.2.2. Technical scaffolding

Technical scaffolding addresses barriers related to using the game simulators themselves, ensuring that technical challenges don't interfere with learning algorithmic concepts. This form of scaffolding is particularly important when introducing new environments or working with students with limited technical experience.

Effective technical scaffolding strategies include:

- Providing clear tutorials and reference materials for simulator interfaces.
- Starting with simplified versions of the environment before introducing all features.
- Creating classroom routines for common technical tasks (saving work, sharing projects, troubleshooting).
- Leveraging peer support through "tech expert" roles within student groups.
- Ensuring appropriate hardware and software configurations before beginning activities.

Alhumairi et al. [3] highlighted both successes and challenges in implementing VR simulation for computer science education. While some students praised the technology for being cutting-edge, others criticized it as unrealistic or non-interactive, with some experiencing motion sickness. These findings emphasize the importance of thorough technical scaffolding when implementing novel technologies.

5.2.3. Accessibility scaffolding

Accessibility scaffolding addresses the specific needs of students with disabilities, ensuring that game simulators are usable by all learners regardless of visual, auditory, motor, or cognitive differences. This area represents a significant challenge, as many game simulators rely heavily on visual interfaces that may exclude students with visual impairments.

Milne, Baker and Ladner [39] described Blocks4All, a blocks-based programming environment designed specifically for blind children. This environment uses touchscreen interfaces and screen readers to make block-based programming accessible to visually impaired learners, demonstrating how thoughtful design can address accessibility challenges.

Effective accessibility scaffolding strategies include:

- Providing alternative input methods for students with motor limitations.
- Ensuring compatibility with assistive technologies like screen readers.
- Offering multiple representation modes (visual, auditory, tactile) for key concepts.
- Adjusting timing and pacing for students who need additional processing time.
- Creating collaborative structures where students with different abilities can support each other.

5.2.4. Social and emotional scaffolding

Social and emotional scaffolding addresses the affective dimensions of learning with game simulators, supporting student motivation, confidence, and collaborative skills. This form of scaffolding is particularly important for maintaining engagement and persistence when students encounter challenges.

Jivani et al. [27] found that gamification positively impacted student engagement and academic performance in undergraduate engineering education. Their implementation incorporated game elements like Kahoot and leaderboard systems, with public recognition of achievements serving as a powerful motivator for continued engagement.

Effective social and emotional scaffolding strategies include:

- Creating a supportive classroom culture that normalizes mistakes as learning opportunities.
- Highlighting progress and growth rather than focusing solely on achievement.
- Incorporating collaborative structures that leverage diverse student strengths.
- Providing timely, specific feedback that guides improvement.
- Celebrating creative approaches and innovative solutions.

The Use-Modify-Create (UMC) framework described by Boulden et al. [5] represents a comprehensive scaffolding approach that integrates cognitive, technical, and social-emotional dimensions. Their study found that this progressive framework effectively supported students regardless of prior programming experience, with the "Use" phase reducing cognitive load for novices while the "Modify" and "Create" phases provided appropriate challenges for more experienced students.

5.3. Assessment strategies and learning analytics

Effective assessment of algorithmic thinking development through game simulators requires approaches that capture both the process and products of student learning. Traditional assessment methods may not adequately reflect the complex, multifaceted nature of learning in game-based environments, necessitating innovative assessment strategies.

5.3.1. Automated assessment tools

Automated assessment tools provide timely, objective feedback on student programming within game simulators. Stahlbauer, Kreis and Fraser [50] introduced a formal testing framework for Scratch programs, enabling automated and property-based testing that achieved 95.25% code coverage. Their Whisker tool can automatically reproduce and replace manually produced grading efforts, opening new possibilities for supporting learners.

Advantages of automated assessment include:

- Immediate feedback that allows students to quickly identify and correct errors.
- Consistent evaluation criteria applied across all student work.
- Scalability to support large numbers of students without increasing teacher workload.
- Detailed analytics that identify common misconceptions and learning patterns.
- Objective measures of code quality and functional correctness.

However, automated assessment also presents challenges, including the need for well-designed test cases, the potential for false positives or negatives, and limitations in evaluating creative aspects of student work. Effective implementation requires balancing automated assessment with human judgment, especially for open-ended tasks.

5.3.2. Performance-based assessment

Performance-based assessment evaluates students' ability to apply algorithmic thinking skills in authentic contexts. Game simulators provide natural opportunities for performance assessment through gameplay challenges, project creation, and problem-solving tasks.

Ekanayake et al. [19] proposed an automated, formula-driven quantitative evaluation method for assessing performance competence in serious training games. Their method achieved up to 90.25% accuracy compared to qualitative assessment, suggesting that quantitative performance metrics can effectively measure learning outcomes in game-based environments.

Effective performance-based assessment strategies include:

- Challenge scenarios that require application of specific algorithmic concepts.
- Project-based assessments where students design and implement solutions to complex problems.
- Competitions where students apply algorithmic thinking to optimize performance.
- Debugging tasks where students identify and correct errors in existing code.
- Portfolio assessments that document progression of skills over time.

5.3.3. Learning analytics

Learning analytics leverage data generated during game-based learning to provide insights into student learning processes and outcomes. Slimani et al. [48] discusses learning analytics through serious games, using data mining algorithms like EM and K-Means to analyze player experience data and evaluate learning outcomes.

Game simulators can generate rich data about student interactions, including:

- Time spent on different activities or challenges.
- Sequences of actions and solution strategies.
- Patterns of errors and corrections.
- Frequency and type of help requests.
- Collaboration and communication patterns in multiplayer environments.

Qasrawi, Amro and Jayousi [45] presents an automatic analytics model for learning skills analysis using game player data and robotic process automation. Their approach automatically processes game data to assess the effect on students' learning skills, providing an efficient pre-posttest assessment tool. Results showed significant improvement in learning skills performance over multiple sessions, demonstrating the value of learning analytics for tracking progress over time.

Effective learning analytics approaches require careful consideration of privacy, data interpretation, and educational relevance. When thoughtfully implemented, they can provide valuable insights for both teachers and students, supporting adaptive instruction and personalized learning pathways.

5.3.4. Multimodal assessment

Multimodal assessment combines multiple assessment approaches to develop a comprehensive understanding of student learning. Fanfarelli [21] recommends combining standardized questionnaires, think-aloud testing, and automated data logging for evaluating computational thinking in serious games. This approach provides a granular and actionable understanding of student learning processes and outcomes.

Effective multimodal assessment strategies include:

- Triangulating data from multiple sources to validate findings.

- Combining quantitative metrics with qualitative insights.
- Balancing process-oriented and product-oriented assessment.
- Incorporating both self-assessment and external evaluation.
- Adapting assessment approaches based on learning objectives and student characteristics.

Horn et al. [25] demonstrated the value of multimodal assessment through their analysis of player strategies in an educational game designed to support algorithmic thinking. By combining quantitative and qualitative game analysis techniques – including hierarchical player clustering, game progression visualizations, playtraces, and think-aloud data – they gained insights into level progression and learning strategies that might have been overlooked with single-method approaches.

5.4. Sample lesson plans and activities

To illustrate how game simulators can be effectively integrated into computer science education, this section presents sample lesson plans and activities for different educational levels and contexts. These examples demonstrate how theoretical principles can be translated into practical classroom implementations.

5.4.1. Primary education: Introduction to sequencing with Blockly Games

Learning objectives:

- Understand the concept of sequencing in programming.
- Create simple programs using sequential commands.
- Debug simple sequence errors.

Materials:

- Computers with internet access.
- Blockly Games "Maze" activity.
- Printed maze grids and direction cards for unplugged activity.

Lesson flow:

1. *Unplugged introduction (10 minutes):* Begin with an unplugged activity where students physically navigate a classroom "maze" by following sequence cards with directional instructions. Students take turns being the "programmer" (giving instructions) and the "computer" (following instructions).
2. *Concept introduction (5 minutes):* Discuss how computers follow instructions in sequence, emphasizing the importance of order. Connect to the unplugged activity, highlighting how the "computer" followed instructions step by step.
3. *Guided exploration (10 minutes):* Introduce Blockly Games' Maze activity, demonstrating how to drag and connect blocks to create a sequence of movements. Complete the first two levels as a class, modeling the thinking process.
4. *Independent practice (13 minutes):* Students work individually or in pairs to progress through Maze levels, applying sequencing concepts. Teacher circulates to provide support and ask probing questions.
5. *Reflection and sharing (7 minutes):* Gather students to discuss challenges faced and strategies developed. Have volunteers share successful approaches or interesting discoveries.

Assessment:

- Formative: observation of student progress through maze levels, noting common misconceptions.

- Summative: completion of a specified maze level with optimal solution (fewest blocks).

Differentiation:

- Support: provide visual reference cards for block functions; pair struggling students with peers.
- Extension: challenge advanced students to solve mazes with constraints (e.g., limited number of blocks, alternative pathways).

5.4.2. Secondary education: Algorithmic problem-solving with 7 Billion Humans

Learning objectives:

- Apply algorithmic thinking to solve complex problems.
- Analyze and optimize code for efficiency.
- Develop solutions that manage multiple actors simultaneously.

Materials:

- Computers with 7 Billion Humans installed.
- Algorithm pseudocode worksheets.
- Optimization challenge cards.

Two-lesson flow:

1. *Problem analysis (15 minutes):* Present a scenario from 7 Billion Humans that requires coordinating multiple workers. Guide students in analyzing the problem, identifying constraints, and recognizing patterns.
2. *Algorithm design (20 minutes):* Students work in pairs to develop pseudocode for solving the problem, considering edge cases and potential optimizations. Share selected approaches with the class for feedback.
3. *Implementation (30 minutes):* Students implement their algorithms in 7 Billion Humans, testing and refining their solutions through iterative debugging.
4. *Optimization challenge (20 minutes):* Introduce optimization metrics (steps, memory usage) and challenge students to improve their solutions, creating a class leaderboard for different optimization priorities.
5. *Analysis and reflection (15 minutes):* Conduct a structured comparison of different approaches, analyzing trade-offs between optimization metrics and solution readability/maintainability.

Assessment:

- Formative: review of pseudocode designs, checking for algorithmic understanding.
- Summative: solution effectiveness (correctness, optimization metrics), written analysis of optimization strategies.

Differentiation:

- Support: provide algorithm templates with key structures; allow pair programming.
- Extension: challenge students to develop solutions optimized for multiple metrics simultaneously; introduce additional constraints.

5.4.3. Higher education: Computational thinking through Minecraft world design

Learning objectives:

- Design complex algorithmic solutions to interdisciplinary problems.
- Implement automated systems using programming in Minecraft.
- Evaluate and communicate the effectiveness of computational approaches.

Materials:

- Computers with Minecraft Education Edition.
- Shared Minecraft server for collaborative work.
- Design brief templates and rubrics.

Project overview: This multi-week project involves students working in interdisciplinary teams to design and implement a Minecraft world that demonstrates a complex system from another discipline (environmental science, urban planning, economics, etc.). The project unfolds in phases:

1. *Research phase (1 week):* Teams research their chosen system, identifying key components, relationships, and behaviors to model in Minecraft.
2. *Design phase (1 week):* Teams develop specifications for their Minecraft implementation, including world design, automated systems, and user interactions. They create algorithm designs for all programmed elements.
3. *Implementation phase (2 weeks):* Teams build their Minecraft worlds, implementing automated systems using redstone circuits and programmed agents. Regular check-ins ensure progress and allow for feedback.
4. *Testing and refinement (1 week):* Teams test their worlds with users from outside the course, gathering feedback and refining their implementations based on observed interactions.
5. *Presentation and evaluation (1 week):* Teams present their worlds to the class, demonstrating key features and explaining the algorithmic approaches used. Peer evaluation contributes to final assessment.

Assessment:

- Formative: weekly progress reports, design document reviews, implementation check-ins.
- Summative: final project demonstration, technical documentation, reflective analysis of algorithmic approaches.

Differentiation:

- Support: provide templates for algorithm design; offer specialized roles within teams based on student strengths.
- Extension: challenge advanced teams to implement API connections to external data sources or develop custom mods.

5.5. Addressing technical and logistical challenges

The implementation of game simulators in educational settings presents various technical and logistical challenges that must be addressed for effective integration. This section explores common challenges and practical strategies for overcoming them.

5.5.1. Technical infrastructure requirements

Game simulators often have specific hardware and software requirements that may exceed available resources in educational settings. Lean et al. [33] identified resource availability as a commonly perceived barrier to implementing simulation-based teaching approaches in higher education.

Strategies for addressing infrastructure challenges include:

- Conducting comprehensive technical audits before implementation to identify potential bottlenecks.
- Selecting simulators with flexible system requirements that can run on available hardware.
- Implementing rotation systems where limited resources are shared among student groups.
- Using cloud-based solutions that reduce local hardware requirements.
- Developing contingency plans for technical failures, including offline alternatives.

Alhumairi et al. [3] described both successes and challenges in implementing VR simulation for computer science education, noting that while setup was straightforward, reliability issues and motion sickness affected some participants. These experiences highlight the importance of testing technologies thoroughly before classroom implementation and having backup plans for technical limitations.

5.5.2. Time and scheduling constraints

Integrating game simulators into existing curricula often requires balancing time constraints with learning objectives. Game-based learning can be time-intensive, particularly during initial implementation phases when both teachers and students are learning new systems.

Strategies for addressing time and scheduling challenges include:

- Starting with small, focused implementations that target specific learning objectives.
- Developing clear lesson structures with time allocations for different activities.
- Creating self-guided tutorials that students can complete independently.
- Extending learning beyond classroom time through homework assignments or optional activities.
- Integrating game simulator activities with existing curricular content rather than treating them as separate units.

Theodosi and Papadimitriou [57] demonstrated an effective approach by incorporating game design activities into existing multimedia courses in high school, aligning game development with established curriculum objectives. This integration allowed for development of algorithmic skills without requiring additional course time.

5.5.3. Teacher preparation and support

Teacher comfort and competence with game simulators significantly influence implementation success. Many teachers may lack experience with gaming technologies or be uncertain about how to effectively integrate them into their teaching practices.

Strategies for addressing teacher preparation challenges include:

- Providing comprehensive professional development that addresses both technical and pedagogical aspects.
- Creating teacher communities of practice for ongoing peer support and idea sharing.

- Developing detailed teacher guides with lesson plans, troubleshooting tips, and assessment strategies.
- Implementing gradual adoption models where teachers master basic features before adding complexity.
- Offering in-classroom support during initial implementation phases.

Guarda and Díaz-Nafría [22] emphasizes the importance of knowledge transfer in simulator implementation, suggesting that successful adoption requires not just technical training but also conceptual understanding of how simulators support learning objectives. This dual focus helps teachers develop both the skills and confidence needed for effective implementation.

5.5.4. Assessment and evaluation challenges

Assessing student learning in game-based environments presents unique challenges, particularly when traditional assessment methods may not capture the complex, multifaceted nature of learning through games.

Strategies for addressing assessment challenges include:

- Developing clear learning objectives and assessment criteria before implementation.
- Creating rubrics that evaluate both process (problem-solving approaches) and product (final solutions).
- Incorporating multiple assessment methods, including automated metrics, observations, and student reflections.
- Using learning analytics to track student progress and identify areas needing additional support.
- Balancing formative and summative assessment approaches.

Qasrawi, Amro and Jayousi [45] demonstrated the potential of automated analytics for learning skills analysis using game player data, showing significant improvement in learning skills performance across multiple sessions. Their approach highlights how game-generated data can be leveraged for assessment purposes, reducing reliance on traditional testing methods.

5.5.5. Accessibility and inclusion considerations

Ensuring that game simulators are accessible to all students, regardless of abilities or backgrounds, represents a critical implementation challenge. Many game environments rely heavily on visual interfaces and may present barriers for students with disabilities.

Strategies for addressing accessibility challenges include:

- Selecting simulators with built-in accessibility features when possible.
- Providing alternative access methods for students with specific needs.
- Creating collaborative structures where students with different abilities work together.
- Consulting with accessibility specialists to identify and address potential barriers.
- Involving students in identifying and solving accessibility challenges.

Milne, Baker and Ladner [39] described efforts to make block-based programming environments accessible via touchscreen for visually impaired students. Their work demonstrates how thoughtful design modifications can address accessibility challenges, though significant work remains to make most game simulators fully inclusive.

6. Discussion

The analysis of game simulators as educational tools for developing algorithmic thinking skills reveals several key insights regarding their effectiveness, implementation considerations, and broader implications for computer science education. This section synthesizes these findings, examines their implications, and considers limitations of current approaches.

6.1. Synthesis of findings

The review of literature and analysis of specific game simulators reveals a complex but promising landscape for algorithmic thinking development through game-based approaches. Several patterns emerge across the various simulators and implementation contexts examined.

First, game simulators demonstrate significant potential for enhancing student engagement and motivation in computer science education. From the narrative-driven challenges of Rabbids Coding to the creative open worlds of Minecraft Education Edition, these environments leverage game mechanics to create compelling learning experiences that can sustain student interest in algorithmic concepts. As Jivani et al. [27] found, game-based approaches can positively impact both engagement and academic performance, creating a dynamic learning environment that captures student attention and encourages active participation.

Second, different simulator types appear to support different aspects of algorithmic thinking development. Block-based environments like Blockly Games and Scratch excel at introducing fundamental concepts through visual, concrete representations, making them particularly effective for novice programmers. Narrative-driven games like 7 Billion Humans provide structured challenges that develop specific algorithmic skills, particularly optimization and efficiency. Open-world environments like Minecraft and Kodu Game Lab encourage creative application of algorithmic thinking in authentic contexts, supporting transfer to real-world problem-solving scenarios.

Third, the effectiveness of game simulators depends significantly on implementation factors, including teacher preparation, scaffolding approaches, and assessment strategies. As Boulden et al. [5] demonstrated with their Use-Modify-Create framework, thoughtful scaffolding can support learners regardless of prior experience, with scaffolded progression from structured guidance to creative independence. Similarly, Pellas and Vosinakis [44] found that instructor feedback and guidance helped students rationalize their decisions in computational practices, highlighting the continued importance of teacher facilitation even in game-based learning environments.

Fourth, game simulators appear to support the development of diverse cognitive skills beyond programming syntax. Spieler et al. [49] found correlations between different game design elements and specific cognitive measures, including working memory, creativity, and arithmetic skills. This suggests that game-based approaches may develop a broader range of cognitive capabilities than traditional programming instruction, potentially supporting transfer to other domains and problem-solving contexts.

Finally, the field shows significant evolution in assessment approaches, moving from subjective evaluation to more sophisticated methods incorporating automated testing, learning analytics, and multimodal assessment. Tools like Whisker for Scratch testing [51] demonstrate how automated assessment can provide objective feedback on program quality while reducing teacher workload. Similarly, analytical approaches like those proposed by Qasrawi, Amro and Jayousi [45] show how game-generated data can be leveraged to track learning progression and identify areas for intervention.

These findings suggest that game simulators represent a valuable addition to the computer science education toolkit, offering engaging, interactive environments for

developing algorithmic thinking skills across various educational levels and contexts. However, their effectiveness depends on thoughtful selection, implementation, and integration with broader educational objectives and approaches.

6.2. Implications for computer science education

The findings regarding game simulators for algorithmic thinking development have several important implications for computer science education practice, policy, and research.

For educational practice, these findings suggest that game simulators can effectively complement traditional programming instruction, providing engaging contexts for concept application and skill development. Rather than replacing traditional approaches, game simulators offer alternative pathways that may reach students who struggle with conventional programming instruction. This suggests a blended approach where game-based and traditional methods are strategically combined to leverage the strengths of each.

The observed relationship between game design elements and specific cognitive skills has implications for instructional design in computer science education. As Spieler et al. [49] found, different game elements correlate with distinct cognitive measures, suggesting that game simulators could be tailored to develop specific skills based on educational objectives. This more nuanced understanding of how game elements influence learning could inform the design of both educational games and broader computer science curricula.

For educational policy, these findings highlight the potential value of investing in game-based approaches to computer science education, particularly at primary and secondary levels. The demonstrated engagement benefits could help address persistent challenges in computer science education, including student retention and diversity. However, realizing this potential requires policy support for teacher professional development, technical infrastructure, and curriculum integration – areas that may require significant investment and institutional commitment.

The varied accessibility considerations across different simulators also have policy implications, particularly regarding educational equity and inclusion. As Milne, Baker and Ladner [39] demonstrated, many popular programming environments present significant barriers for students with disabilities. Policy initiatives supporting accessible design in educational technology could help ensure that game-based approaches to computer science education are available to all students, regardless of abilities or backgrounds.

For research in computer science education, these findings underscore the need for more rigorous, longitudinal studies examining the long-term effects of game-based approaches on algorithmic thinking development. While existing research demonstrates short-term benefits, questions remain about skill retention, transfer, and comparative effectiveness across different simulators and implementation approaches. Addressing these questions requires sustained research efforts with robust methodologies and diverse participant populations.

The emergent assessment approaches identified in this analysis also have implications for computer science education research. Tools like automated testing frameworks [51] and learning analytics models [45] offer new methodologies for measuring learning processes and outcomes in programming education. These approaches could inform broader assessment practices in computer science education, potentially addressing persistent challenges in evaluating complex problem-solving skills.

6.3. Recommendations for educators and curriculum designers

Based on the analysis of game simulators and their implementation contexts, several recommendations emerge for educators and curriculum designers seeking to incorporate these tools into computer science education:

1. *Strategic simulator selection*
   Rather than adopting game simulators based on popularity or novelty, educators should strategically select environments that align with specific learning objectives, student characteristics, and available resources. The comparative analysis in section 4.4 provides a framework for evaluating simulators across dimensions including features, programming concepts, user interface, assessment capabilities, and curriculum alignment. This evaluation should inform selection decisions, ensuring that chosen simulators effectively support educational goals.
2. *Scaffolded implementation*
   The evidence suggests that scaffolded implementation approaches are particularly effective for integrating game simulators into computer science education. Frameworks like Use-Modify-Create [5] provide structured progressions that support diverse learners while maintaining appropriate challenges. Educators should design implementation plans that include explicit scaffolding for both technical and conceptual aspects of game-based learning, with gradual release of responsibility as students develop greater competence and confidence.
3. *Multimodal assessment*
   The limitations of traditional assessment approaches in capturing learning through game simulators suggest the need for multimodal assessment strategies. Educators should combine multiple assessment methods, including automated metrics, observations, artifacts, and reflections, to develop comprehensive understanding of student learning. As Fanfarelli [21] recommends, combining standardized questionnaires, think-aloud testing, and automated data logging can provide granular insights into learning processes and outcomes.
4. *Integration with broader curriculum*
   Rather than treating game simulators as isolated activities, educators should integrate them thoughtfully with broader curriculum objectives and learning progressions. This integration might involve connecting game-based activities to concepts from other subject areas, using games to illustrate principles introduced through traditional instruction, or designing game-based projects that synthesize learning across multiple topics. This integrated approach supports transfer of learning and helps students recognize the relevance of algorithmic thinking across diverse contexts.
5. *Teacher professional development*
   The significant influence of teacher facilitation on the effectiveness of game simulators highlights the importance of comprehensive professional development. Beyond technical training, this development should address pedagogical approaches, assessment strategies, and classroom management in game-based learning environments. Creating communities of practice where teachers can share experiences, resources, and strategies can support ongoing professional growth and implementation refinement.
6. *Inclusive design considerations*
   The accessibility limitations of many game simulators underscore the need for deliberate attention to inclusive design. Educators and curriculum designers should evaluate simulators for accessibility features, develop alternative access methods for students with disabilities, and create collaborative structures that leverage diverse student strengths. Consulting with accessibility specialists

and involving students in identifying and addressing barriers can support more inclusive implementation of game-based approaches.

7. *Balanced approach to gamification*
   While game elements can enhance engagement and motivation, educators should maintain a balanced approach to gamification that emphasizes intrinsic motivation and meaningful learning rather than extrinsic rewards. As Adams and Du Preez [1] found in their design-based research on gamification, effective implementation requires contextually sensitive design and iterative refinement based on student responses. Overemphasis on competition or reward structures may undermine deeper learning objectives and create inequitable learning environments.

6.4. Limitations of current approaches

While game simulators show significant promise for developing algorithmic thinking skills, current approaches have several limitations that merit consideration. Understanding these limitations is essential for realistic implementation planning and continued improvement of game-based approaches to computer science education:

1. *Empirical evidence gaps*
   Despite growing research interest in game-based approaches to algorithmic thinking development, significant gaps remain in the empirical evidence base. As Varghese and Renumol [61] notes in their systematic literature review of video games for assessing computational thinking, "there is a lack of empirical evidence to prove that video games are effective to assess CT skills". Many studies rely on small sample sizes, short intervention periods, or pre-experimental designs that limit generalizability and causal inference. Longitudinal studies examining sustained impact on algorithmic thinking development are particularly scarce, limiting our understanding of long-term effectiveness.
2. *Transfer challenges*
   Questions persist about how effectively skills developed in game environments transfer to other programming contexts and problem-solving domains. The engaging, scaffolded nature of many game simulators may not prepare students for the less structured challenges of real-world programming. Gutiérrez, Dávila and Quintana [23] suggests that future studies should prolong experimentation and increase experimental group sizes to better evaluate the effectiveness of serious games in developing computational thinking. Without stronger evidence regarding transfer, the broader educational value of game simulators remains somewhat uncertain.
3. *Accessibility barriers*
   Many game simulators rely heavily on visual interfaces and interactions that present significant barriers for students with disabilities. While some efforts have been made to develop accessible alternatives, such as the touchscreen-based blocks environment described by Milne, Baker and Ladner [39], most popular simulators remain inaccessible to students with visual impairments. Similar barriers may exist for students with motor, cognitive, or other disabilities, potentially excluding these students from the benefits of game-based learning approaches.
4. *Resource requirements*
   Implementation of game simulators often requires substantial resources, including hardware, software, technical support, and teacher training. As Lean et al. [33] found, resource availability represents a commonly perceived barrier to simulation-based teaching. These resource requirements may exacerbate

educational inequities if more privileged schools can implement game-based approaches while under-resourced schools cannot. The technical challenges reported by Alhumairi et al. [3] in their VR simulation implementation highlight how resource limitations can impact the educational experience even in relatively well-equipped settings.

5. *Assessment complexity*
   Assessing learning in game-based environments presents significant challenges, particularly when traditional assessment methods may not capture the complex, multifaceted nature of learning through games. While innovative assessment approaches are emerging, such as the automated testing framework described by Stahlbauer, Kreis and Fraser [51], these approaches require technical expertise and resources that may not be available in all educational contexts. The lack of standardized, validated assessment tools for algorithmic thinking complicates both implementation and evaluation of game-based approaches.
6. *Balancing engagement and learning*
   Finding the right balance between entertainment and educational value remains a persistent challenge in educational game design. Games that prioritize engagement over learning may provide enjoyable experiences that develop limited algorithmic thinking skills. Conversely, games that emphasize educational content at the expense of engaging gameplay may fail to maintain student interest and motivation. Achieving this balance requires interdisciplinary expertise in game design, computer science education, and learning sciences – a combination that is not readily available in many educational contexts.
7. *Cultural and contextual factors*
   The effectiveness of game simulators may be influenced by cultural and contextual factors that are not well understood. Educational approaches that succeed in one cultural context may not translate effectively to others due to differences in educational traditions, gaming familiarity, and cultural attitudes toward play and learning. The limited international research on game simulators for algorithmic thinking development makes it difficult to evaluate how these tools function across diverse cultural contexts.

These limitations do not negate the potential value of game simulators for algorithmic thinking development but highlight areas requiring attention from researchers, educators, and educational technology developers. Addressing these limitations through thoughtful design, implementation, and research will be essential for realizing the full potential of game-based approaches in computer science education.

7. Future research directions

The analysis of game simulators for algorithmic thinking development reveals several promising areas for future research. This section outlines key research directions that could address current limitations, extend our understanding of game-based approaches, and improve educational practices in this field.

7.1. Methodological approaches for empirical studies

Future research would benefit from more robust methodological approaches that address limitations in the current evidence base. Several specific methodological directions merit exploration:

1. To address questions about long-term effectiveness and skill retention, *longitudinal studies* tracking student development over extended periods are needed. These studies should examine how skills developed through game simulators

persist, evolve, and transfer to other contexts over time. Gutiérrez, Dávila and Quintana [23] emphasizes the need for prolonged experimentation to develop more conclusive evidence about the effectiveness of serious games for computational thinking development. Longitudinal designs with multiple measurement points could provide insights into learning trajectories and factors influencing sustained development of algorithmic thinking skills.

2. More rigorous *comparative studies* examining different simulator types, implementation approaches, and educational contexts would enhance our understanding of what works, for whom, and under what conditions. These studies should employ robust research designs with appropriate control or comparison groups, adequate sample sizes, and valid, reliable measures of algorithmic thinking. Pellas [43] conducted a quasi-experimental study comparing OpenSimulator with Scratch4SL to Scratch alone, finding differences in game experience and satisfaction but no significant difference in learning performance. More studies of this type, with larger samples and longer intervention periods, could clarify the relative effectiveness of different approaches.
3. The complex, multifaceted nature of learning through game simulators suggests the value of mixed *methods research designs* that combine quantitative measures of learning outcomes with qualitative insights into learning processes. Horn et al. [25] demonstrated the value of this approach in their analysis of player strategies in an educational game for algorithmic thinking, combining quantitative clustering techniques with qualitative analysis of player behaviors and think-aloud data. Expanded mixed methods approaches could provide richer understanding of how game simulators influence algorithmic thinking development.
4. *Design-based research approaches*, which involve iterative cycles of design, implementation, and refinement based on systematic investigation of educational practices, offer promising frameworks for developing and evaluating game simulators. Adams and Du Preez [1] employed this approach in their study of gamification for student engagement, developing design principles through iterative implementation and qualitative data collection. Similar approaches focused specifically on algorithmic thinking development could generate both practical design guidance and theoretical insights into learning processes.
5. Involving students, teachers, and other stakeholders as *active participants in research processes* could enhance both the relevance and validity of findings regarding game simulators. Participatory approaches might involve students in game design and evaluation, teachers in implementation planning and assessment development, or community members in defining relevant problem contexts. This collaborative approach could address limitations in current research while producing more contextually appropriate and culturally responsive game-based learning environments.

7.2. Key research questions

Several key research questions emerge from the analysis of current literature and practice regarding game simulators for algorithmic thinking development:

1. *Transfer and generalization:* How effectively do algorithmic thinking skills developed through game simulators transfer to other programming contexts and problem-solving domains? What factors influence this transfer, and how can game designs and implementation approaches enhance skill generalization? Varghese and Renumol [61] highlights the need for research examining whether game-based assessment of computational thinking predicts performance in other contexts, a question that could be extended to examine transfer of learning as well as assessment.

2. *Differential effectiveness:* How do different types of game simulators support the development of different aspects of algorithmic thinking? Are certain simulators more effective for specific concepts, skills, or student populations? Spieler et al. [49] found correlations between specific gaming elements and distinct cognitive measures, suggesting differentiated effects that merit further investigation through targeted studies of simulator characteristics and learning outcomes.
3. *Implementation factors:* What implementation factors most significantly influence the effectiveness of game simulators for algorithmic thinking development? How do teacher facilitation approaches, scaffolding strategies, assessment methods, and integration with broader curriculum impact learning outcomes? Pellas and Vosinakis [44] found that instructor feedback and guidance facilitated students' ability to rationalize decisions in computational practices, highlighting the importance of teacher roles in game-based learning environments – a topic requiring deeper investigation.
4. *Accessibility and inclusion:* How can game simulators be designed and implemented to support algorithmic thinking development for diverse learners, including students with disabilities and those from underrepresented groups in computer science? What design principles and adaptation strategies promote inclusive game-based learning environments? Milne, Baker and Ladner [39] demonstrated the possibility of making block-based programming accessible for visually impaired students, but much work remains to develop fully inclusive approaches to game-based algorithmic thinking development.
5. *Assessment validation:* How can we validly and reliably assess algorithmic thinking development through game simulators? What assessment approaches best capture the complex, multifaceted nature of learning in game-based environments? Fanfarelli [21] recommends combining multiple assessment methods for comprehensive evaluation, but questions remain about the validity, reliability, and practical feasibility of these approaches in diverse educational contexts.
6. *Long-term impact:* What are the long-term effects of game-based approaches to algorithmic thinking development on student learning trajectories, educational and career choices, and broader problem-solving capabilities? How do these effects compare to those of traditional programming instruction? Yusuf and Noor [62] examined algorithmic thinking growth trajectories in different programming environments, finding evidence of both Matthew effects (where strong students get stronger) and compensatory effects (where weaker students catch up) in different contexts – a model that could be extended to examine long-term development through game simulators.

7.3. Emerging technologies and their potential impacts

Several emerging technologies show potential for enhancing game simulators and their application in algorithmic thinking development. Research exploring these technologies and their educational applications represents an important direction for future investigation:

1. Advanced *VR and AR technologies* offer possibilities for more immersive, interactive programming environments that blend physical and virtual learning experiences. Alhumairi et al. [3] investigated VR simulation for computer science education, finding mixed results that highlight both opportunities and challenges. Future research could explore how these technologies might support spatial reasoning in algorithmic thinking, provide novel visualization approaches for abstract concepts, or create more engaging collaborative programming environments.

2. *AI technologies*, particularly machine learning and natural language processing, could enhance game simulators through adaptive scaffolding, personalized feedback, and intelligent tutoring capabilities. Faber et al. [20] found that adaptive scaffolding based on interaction trace data improved speed and reduced cognitive load in game-based learning, suggesting potential benefits from more sophisticated adaptive systems. Research exploring how AI can support differentiated learning pathways and provide targeted assistance could significantly enhance the effectiveness of game simulators for diverse learners.
3. The integration of *physical computing elements* with digital game environments offers possibilities for embodied learning experiences that may enhance conceptual understanding and engagement. Aggarwal, Gardner-Mccune and Touretzky [2] evaluated the effect of using physical manipulatives to foster computational thinking in elementary school, finding that physical tiles and flashcards supported rule construction in programming activities. Research exploring hybrid physical-digital game simulators could expand our understanding of how embodied interactions influence algorithmic thinking development.
4. Advanced *learning analytics* approaches could transform assessment and adaptation in game-based learning environments by providing real-time insights into student learning processes and outcomes. Qasrawi, Amro and Jayousi [45] demonstrated the potential of automated analytics for learning skills analysis using game player data, showing how data mining techniques could identify patterns and track progression. Research developing more sophisticated analytics approaches specific to algorithmic thinking development could enhance both assessment and adaptive instruction in game simulators.
5. *Cloud-based game simulators and collaborative platforms* could address resource limitations while enabling new forms of collaborative learning and community participation. Klimova, Sajben and Lovaszova [30] described an online Minecraft: Education Edition Programming Contest that engaged students across geographic locations, demonstrating the potential of cloud-based approaches. Research exploring how these technologies can support broader access to game-based learning, facilitate collaborative programming projects, and connect learners across diverse contexts could expand the reach and impact of game simulators for algorithmic thinking development.

7.4. Interdisciplinary connections

Future research on game simulators for algorithmic thinking development would benefit from stronger interdisciplinary connections that bring diverse perspectives and methodologies to bear on complex educational challenges:

1. Deeper integration with *learning sciences and cognitive psychology* could enhance our understanding of how game simulators influence cognitive processes related to algorithmic thinking. Spieler et al. [49] found connections between game design activities and specific cognitive skills, suggesting fertile ground for interdisciplinary investigation. Research drawing on cognitive theories of skill acquisition, problem-solving, and knowledge transfer could inform both game design and implementation approaches, enhancing alignment with cognitive development principles.
2. Stronger connections with *game design and user experience* fields could lead to more engaging, effective educational games that balance entertainment and learning objectives. Johnson et al. [28] describes game development for computer science education as requiring interdisciplinary approaches drawing on game software design, instructional design, and teacher education perspectives. Research

partnerships bringing together game designers, computer science educators, and learning scientists could produce innovative approaches to algorithmic thinking development that leverage game design principles while maintaining educational rigor.

3. Collaboration with *educational technology and human-computer interaction* researchers could address usability, accessibility, and technical implementation challenges in game simulators. Caraco et al. [8] describes efforts to make the Blockly library accessible via touchscreen, demonstrating how HCI perspectives can enhance accessibility. Research examining interface design, user interaction patterns, and technology integration could lead to more usable, accessible game simulators appropriate for diverse educational contexts.
4. Engagement with *equity, diversity, and inclusion* research could help ensure that game simulators support algorithmic thinking development for all students, regardless of background or ability. Ng [40] found gender differences in educational game use for learning programming, highlighting the importance of equity considerations in game design and implementation. Research examining how game simulators might address persistent disparities in computer science participation and achievement could contribute to more equitable educational practices and outcomes.
5. Connections with *implementation science and educational change* research could enhance understanding of how game simulators can be effectively integrated into educational systems and sustained over time. Lean et al. [33] found that perceptions of risk and suitability influenced simulation adoption more than resource availability, highlighting organizational and cultural factors in implementation. Research examining how educational institutions adopt, adapt, and sustain game-based approaches to algorithmic thinking development could identify strategies for overcoming implementation barriers and supporting effective educational change.

These interdisciplinary connections would not only enrich research on game simulators but also enhance the practical relevance and theoretical grounding of this work. By drawing on diverse disciplinary perspectives and methodologies, future research can develop more comprehensive understanding of how game simulators can effectively support algorithmic thinking development across diverse educational contexts.

8. Conclusion

This paper has provided a comprehensive analysis of game simulators as educational tools for developing algorithmic thinking skills in computer science education. Through examination of theoretical foundations, current research, specific simulators, implementation strategies, and future directions, several key insights emerge regarding the potential and limitations of these approaches.

Game simulators offer promising approaches to algorithmic thinking development, leveraging the engaging, interactive nature of games to make abstract computational concepts more accessible and meaningful for diverse learners. The various simulators examined – from block-based environments like Blockly Games and Scratch to narrative-driven games like Rabbids Coding and 7 Billion Humans to open-world environments like Minecraft Education Edition and Kodu Game Lab – each provide distinct approaches to supporting algorithmic thinking development, with different strengths, limitations, and appropriate applications.

The effectiveness of game simulators depends significantly on implementation factors, including thoughtful selection aligned with learning objectives, appropriate

scaffolding for diverse learner needs, innovative assessment approaches that capture complex learning processes, and integration with broader educational contexts. When these factors are addressed through systematic implementation planning and ongoing refinement, game simulators can enhance student engagement, develop specific algorithmic thinking skills, and support transfer to other problem-solving contexts.

However, significant challenges remain in realizing the full potential of game simulators for algorithmic thinking development. These include technical and logistical barriers related to resource requirements and teacher preparation, accessibility limitations that may exclude students with disabilities, assessment complexities that complicate evaluation of learning outcomes, and limited evidence regarding long-term effectiveness and skill transfer. Addressing these challenges requires continued research, development, and collaboration across disciplines and stakeholder groups.

The analysis of game simulators for algorithmic thinking development has broader implications for educational technology and computer science education. It demonstrates how technology-enhanced learning environments can provide engaging, interactive contexts for developing complex cognitive skills, offering alternatives to traditional instructional approaches that may better serve diverse learners and learning objectives. The success factors and challenges identified in this analysis may inform development and implementation of other educational technologies beyond game simulators.

For computer science education specifically, this analysis highlights the value of diversifying instructional approaches to reach broader student populations and develop more holistic understanding of computational concepts. By complementing traditional programming instruction with game-based approaches, educators may enhance both engagement and effectiveness, particularly for students who might struggle with conventional methods. The emphasis on algorithmic thinking rather than programming syntax aligns with growing recognition of computational thinking as a foundational skill applicable across disciplines beyond computer science.

The implementation challenges identified in this analysis also reveal broader issues in educational technology integration, including resource inequities, teacher preparation gaps, and assessment limitations. Addressing these challenges requires systemic approaches that consider not just technological tools but also the educational ecosystems in which they operate. By attending to these broader contextual factors, educators and researchers can enhance the effectiveness and equity of technology-enhanced learning across domains.

Looking forward, game simulators for algorithmic thinking development represent a rapidly evolving field with significant potential for innovation and impact. Emerging technologies like virtual reality, artificial intelligence, and learning analytics offer possibilities for more immersive, adaptive, and insightful game-based learning experiences. Interdisciplinary collaborations across learning sciences, game design, human-computer interaction, and implementation science could drive development of more effective, accessible, and sustainable approaches to game-based algorithmic thinking education.

Realizing this potential will require continued research addressing the questions and gaps identified in this analysis, particularly regarding long-term effectiveness, skill transfer, accessibility, and implementation factors. It will also require thoughtful practice that balances technological innovation with sound pedagogical principles, ensuring that game simulators serve meaningful educational objectives rather than novelty for its own sake. Perhaps most importantly, it will require commitment to educational equity, ensuring that game-based approaches to algorithmic thinking development are available and effective for all students, regardless of background, ability, or resources.

Since we are learning in a digital age, the use of information and communication

technologies, including game simulators and gamification technology in general, to expand the capabilities of teachers ensures that the next generation of computer science professionals will acquire the skills necessary for their future professional activities. By thoughtfully designing, implementing, and researching game simulators for algorithmic thinking development, educators and researchers can contribute to this broader goal of preparing all students for successful participation in our increasingly computational world.

Declaration on generative AI: During the preparation of this work, the authors used Claude 3.7 Sonnet to enhance content and improve writing style. After using this tool, the authors reviewed and edited the content as needed and took full responsibility for the publication's content.

References

[1] Adams, S.P. and Du Preez, R., 2022. Supporting Student Engagement Through the Gamification of Learning Activities: A Design-Based Research Approach. *Technology, Knowledge and Learning*, 27(1), pp.119–138. Available from: https://doi.org/10.1007/s10758-021-09500-x.

[2] Aggarwal, A., Gardner-Mccune, C. and Touretzky, D.S., 2017. Evaluating the effect of using physical manipulatives to foster computational thinking in elementary school. *Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE*. pp.9–14. Available from: https://doi.org/10.1145/3017680.3017791.

[3] Alhumairi, A., Ebrahimi, R., Sahli, N. and Fakhrulddin, A., 2024. VR Simulation: Advancing Practical Skills in Computer Science Education. *Proceedings of the European Conference on Games-based Learning*, 18(1), pp.22–30. Available from: https://doi.org/10.34190/ecgbl.18.1.2819.

[4] Bacelo, A. and Gómez-Chacón, I.M., 2023. Characterising algorithmic thinking: A university study of unplugged activities. *Thinking Skills and Creativity*, 48, p.101284. Available from: https://doi.org/10.1016/j.tsc.2023.101284.

[5] Boulden, D.C., Rachmatullah, A., Hinckle, M., Bounajim, D., Mott, B., Boyer, K.E., Lester, J. and Wiebe, E., 2021. Supporting Students' Computer Science Learning with a Game-based Learning Environment that Integrates a Use-Modify-Create Scaffolding Framework. *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*. pp.129–135. Available from: https://doi.org/10.1145/3430665.3456349.

[6] Bratitsis, T., Tsapara, M., Melliou, K., Busuttil, L., Vassallo, D., Callus, J., Meireles, G., Koliakou, I., Kojok, N.T. and Sousa, S., 2024. Cultivating Computational Thinking in Early Years Through Board Games. The Cthink.it Approach. *Lecture Notes in Networks and Systems*, 937 LNNS, pp.78–89. Available from: https://doi.org/10.1007/978-3-031-56075-0_8.

[7] Campos, N., Nogal, M., Caliz, C. and Juan, A.A., 2020. Simulation-based education involving online and on-campus models in different European universities. *International Journal of Educational Technology in Higher Education*, 17(1), p.8. Available from: https://doi.org/10.1186/s41239-020-0181-y.

[8] Caraco, L.B., Deibel, S., Ma, Y. and Milne, L.R., 2019. Making the blockly library accessible via touchscreen. *ASSETS 2019 - 21st International ACM SIGACCESS Conference on Computers and Accessibility*. pp.648–650. Available from: https://doi.org/10.1145/3308561.3354589.

[9] Castillo-Parra, B., Hidalgo-Cajo, B.G., Vásconez-Barrera, M. and Oleas-López, J., 2022. Gamification in higher education: A review of the literature. *World Journal on Educational Technology: Current Issues*, 14(3), pp.797–816. Available from: https://doi.org/10.18844/wjet.v14i3.7341.

[10] Chen, L., Dowling, D. and Goetz, C., 2023. At the nexus of ludology and narratology: Advances in reality-based story-driven games. *F1000Research*, 12, p.45. Available from: https://doi.org/10.12688/f1000research.129113.1.

[11] Chien, C.C., Ho, Y.T. and Hou, H.T., 2024. Integrating Immersive Scenes and Interactive Contextual Clue Scaffolding Into Decision-Making Analysis Ability Training Game. *Journal of Educational Computing Research*, 62(1), pp.376–405. Available from: https://doi.org/10.1177/07356331231205058.

[12] Chornous, G., Banna, O., Fedorenko, I. and Didenko, I., 2022. Implementing ERP Simulation Games in Economic Education: Ukrainian Dimension. *Communications in Computer and Information Science*, 1635 CCIS, pp.112–132. Available from: https://doi.org/10.1007/978-3-031-14841-5_8.

[13] Cossío-Silva, F.J., Vega-Vázquez, M. and Revilla-Camacho, M.Á., 2015. Simulation Games and the Development of Competences. Empirical Evidence in Marketing. In: M. Peris-Ortiz and J.M. Merigó Lindahl, eds. *Sustainable Learning in Higher Education: Developing Competencies for the Global Marketplace*. Cham: Springer International Publishing, Innovation, Technology and Knowledge Management, pp.103–111. Available from: https://doi.org/10.1007/978-3-319-10804-9_8.

[14] Da Silva, J.P. and Falcão, T.P., 2017. Children's games and computational thinking: Looking for a set of design guidelines [Jogos infantis e pensamento computacional: Em busca de um conjunto de diretrizes de design]. *CEUR Workshop Proceedings*, 1877, pp.345–356.

[15] De Freitas, S.I., 2006. Using games and simulations for supporting learning. *Learning, Media and Technology*, 31(4), pp.343–358. Available from: https://doi.org/10.1080/17439880601021967.

[16] Deiner, A., Feldmeier, P., Fraser, G., Schweikl, S. and Wang, W., 2023. Automated test generation for Scratch programs. *Empirical Software Engineering*, 28(3), p.79. Available from: https://doi.org/10.1007/s10664-022-10255-x.

[17] Denning, P.J., 2009. The profession of IT: Beyond computational thinking. *Communications of the ACM*, 52(6), pp.28–30. Available from: https://doi.org/10.1145/1516046.1516054.

[18] Dicheva, D., Dichev, C., Agre, G. and Angelova, G., 2015. Gamification in education: A systematic mapping study. *Educational Technology and Society*, 18(3), pp.75–88.

[19] Ekanayake, H., Backlund, P., Ziemke, T., Ramberg, R. and Hewagamage, K., 2011. Assessing performance competence in training games. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6975 LNCS(PART 2), pp.518–527. Available from: https://doi.org/10.1007/978-3-642-24571-8_65.

[20] Faber, T.J.E., Dankbaar, M.E.W., Broek, W.W. van den, Bruinink, L.J., Hogeveen, M. and Merriënboer, J.J.G. van, 2024. Effects of adaptive scaffolding on performance, cognitive load and engagement in game-based learning: a randomized controlled trial. *BMC Medical Education*, 24(1), p.943. Available from: https://doi.org/10.1186/s12909-024-05698-3.

[21] Fanfarelli, J.R., 2021. Assessing Computational Thinking Pedagogy in Serious Games Through Questionnaires, Think-aloud Testing, and Automated Data Logging. *Proceedings - 2021 IEEE/ACIS 21st International Fall Conference on Computer and Information Science, ICIS 2021-Fall*. pp.149–152. Available from: https://doi.org/10.1109/ICISFall51598.2021.9627365.

[22] Guarda, T. and Díaz-Nafría, J.M., 2024. Use of Simulators as a Digital Resource for Knowledge Transference. *Communications in Computer and Information Science*, 1937 CCIS, pp.116–127. Available from: https://doi.org/10.1007/

978-3-031-48930-3_9.

[23] Gutiérrez, S.A.C., Dávila, G.A. and Quintana, H., 2023. Implementation of a Serious Game to Develop Computational Thinking Skills. *Studies in Systems, Decision and Control*, 497, pp.165–182. Available from: https://doi.org/10.1007/978-3-031-40710-9_9.

[24] Hernandez, L., Esparcia, S., Julian, V. and Carrascosa, C., 2016. JGOMAS 2.0: A capture-the-flag game using jason agents and human interaction. *Communications in Computer and Information Science*, 616, pp.173–184. Available from: https://doi.org/10.1007/978-3-319-39387-2_15.

[25] Horn, B., Folajimi, Y., Hoover, A.K., Smith, G., Barnes, J. and Harteveld, C., 2016. Opening the black box of play: Strategy analysis of an educational game. *CHI PLAY 2016 - Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play*. pp.142–153. Available from: https://doi.org/10.1145/2967934.2968109.

[26] Hsu, C.C. and Wang, T.I., 2018. Applying game mechanics and student-generated questions to an online puzzle-based game learning system to promote algorithmic thinking skills. *Computers and Education*, 121, pp.73–88. Available from: https://doi.org/10.1016/j.compedu.2018.02.002.

[27] Jivani, S.R., Chetehouna, M., Hafeez, S. and Adjali, M.H., 2024. Effects of Game-Based Learning on Engagement and Academic Performance for Undergraduate Science and Engineering Students. *International Journal of Engineering Education*, 40(1), pp.16–22.

[28] Johnson, C., Mcgill, M., Bouchard, D., Bradshaw, M.K., Bucheli, V.A., Merkle, L.D., Scott, M.J., Sweedyk, Z., Ángel, J., Xiao, Z. and Zhang, M., 2016. Game development for computer science education. *Proceedings of the 2016 ITiCSE Working Group Reports, ITiCSE 2016*. pp.23–44. Available from: https://doi.org/10.1145/3024906.3024908.

[29] Keller, J.M., 2012. ARCS Model of Motivation. In: N.M. Seel, ed. *Encyclopedia of the Sciences of Learning*. Boston, MA: Springer US, pp.304–305. Available from: https://doi.org/10.1007/978-1-4419-1428-6_217.

[30] Klimova, N., Sajben, J. and Lovaszova, G., 2021. Online game-based learning through minecraft: Education edition programming contest. *IEEE Global Engineering Education Conference, EDUCON*. vol. 2021-April, pp.1660–1668. Available from: https://doi.org/10.1109/EDUCON46332.2021.9453953.

[31] Klock, A.C.T., Ogawa, A.N., Gasparini, I. and Pimenta, M.S., 2018. Does gamification matter? A systematic mapping about the evaluation of gamification in educational environments. *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. New York, NY, USA: Association for Computing Machinery, SAC '18, p.2006–2012. Available from: https://doi.org/10.1145/3167132.3167347.

[32] Kovtaniuk, M.S., 2023. Perevahy vykorystannia mobilnykh ihrovykh symuliatoriv pid chas vyvchennia osnov prohramuvannia. *Nauka. Osvita. Molod : materialy XVI Vseukr. nauk. konf. studentiv ta molodykh naukovtsiv, m. Uman, 11 travnia 2023 r.* Uman, pp.152–153. Available from: https://dspace.udpu.edu.ua/handle/123456789/15597.

[33] Lean, J., Moizer, J., Towler, M. and Abbey, C., 2006. Simulations and games: Use and barriers in higher education. *Active Learning in Higher Education*, 7(3), pp.227–242. Available from: https://doi.org/10.1177/1469787406069056.

[34] Ledger, S., Mailizar, M., Gregory, S., Tanti, M., Gibson, D. and Kruse, S., 2025. Learning to teach with simulation: historical insights. *Journal of Computers in Education*, 12(1), pp.339–366. Available from: https://doi.org/10.1007/s40692-024-00313-2.

[35] Lei, Y., 2021. A Study on Online Game-based Teaching Design. *Proceedings - 2021 2nd International Conference on Education, Knowledge and Information*

Management, ICEKIM 2021. pp.184–190. Available from: https://doi.org/10.1109/ICEKIM52309.2021.00047.

[36] Liang, L.R., Kang, R. and Mendoza, C.S., 2024. Developing Lafayette Park Minecraft World to Broaden Participation in Computing. *Asee annual conference and exposition, conference proceedings*.

[37] MacLaurin, M., 2011. The design of kodu: A tiny visual programming language for children on the Xbox 360. *ACM SIGPLAN Notices*, 46(1), pp.241–245. Available from: https://doi.org/10.1145/1925844.1926413.

[38] McGaghie, W.C. and Harris, I.B., 2018. Learning Theory Foundations of Simulation-Based Mastery Learning. *Simulation in Healthcare*, 13(3 S), pp.S15–S20. Available from: https://doi.org/10.1097/SIH.0000000000000279.

[39] Milne, L.R., Baker, C.M. and Ladner, R.E., 2017. Blocks4All demonstration: A blocks-based programming environment for blind children. *ASSETS 2017 - Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility*. pp.313–314. Available from: https://doi.org/10.1145/3132525.3134774.

[40] Ng, E.M.W., 2010. Using an educational game to learn - are there any gender differences between pre-service teachers? *ASCILITE 2010 - The Australasian Society for Computers in Learning in Tertiary Education*. pp.684–689.

[41] Nguyen, T.H., Nguyen, T.L. and Nguyen, T.B., 2023. Design and evaluate a stem-oriented education teaching plan by exploiting the strength of minecraft education game-based platform. *AIP Conference Proceedings*, 2685, p.030014. Available from: https://doi.org/10.1063/5.0112025.

[42] Park, H. and Jun, W., 2023. A Study of Development of Algorithm Thinking Evaluation Standards. *International Journal of Applied Engineering and Technology*, 5(2), pp.53–58.

[43] Pellas, N., 2024. Effects of Simulation Games on students' Computational Thinking and Game Experience for Programming Courses in Primary School. *Computers in the Schools*, 41(1), pp.23–50. Available from: https://doi.org/10.1080/07380569.2023.2206825.

[44] Pellas, N. and Vosinakis, S., 2018. Learning to think and practice computationally via a 3D simulation game. *Advances in Intelligent Systems and Computing*, 725, pp.550–562. Available from: https://doi.org/10.1007/978-3-319-75175-7_54.

[45] Qasrawi, R., Amro, M. and Jayousi, R., 2020. Automatic analytics model for learning skills analysis using game player data and robotic process automation in a serious game for education. *Proceedings - 2020 International Conference on Promising Electronic Technologies, ICPET 2020*. pp.94–98. Available from: https://doi.org/10.1109/ICPET51420.2020.00026.

[46] Renganathan, K.K., Karuppiah, J., Lakshminarayanan, J. and Pathinathan, M., 2024. Enhancing algorithmic reasoning and critical thinking through game-based learning: A graph theory approach. *Multidisciplinary Reviews*, 7(10), p.2024233. Available from: https://doi.org/10.31893/multirev.2024233.

[47] Rutherford-Hemming, T., 2012. Simulation Methodology in Nursing Education and Adult Learning Theory. *Adult Learning*, 23(3), pp.129–137. Available from: https://doi.org/10.1177/1045159512452848.

[48] Slimani, A., Elouaai, F., Elaachak, L., Yedri, O.B. and Bouhorma, M., 2018. Learning analytics through serious games: Data mining algorithms for performance measurement and improvement purposes. *International Journal of Emerging Technologies in Learning*, 13(1), pp.46–64. Available from: https://doi.org/10.3991/ijet.v13i01.7518.

[49] Spieler, B., Kemény, F., Landerl, K., Binder, B. and Slany, W., 2020. The learning value of game design activities: association between computational thinking and

cognitive skills. *Proceedings of the 15th Workshop on Primary and Secondary Computing Education*. New York, NY, USA: Association for Computing Machinery, WiPSCE '20, p.19. Available from: https://doi.org/10.1145/3421590.3421607.

[50] Stahlbauer, A., Kreis, M. and Fraser, G., 2019. Testing scratch programs automatically. *ESEC/FSE 2019 - Proceedings of the 2019 27th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. pp.165–175. Available from: https://doi.org/10.1145/3338906.3338910.

[51] Stahlbauer, A., Kreis, M. and Fraser, G., 2020. Analyzing Scratch Programs with Automated Tests. *Lecture Notes in Informatics (LNI), Proceedings - Series of the Gesellschaft fur Informatik (GI)*. vol. P-300, pp.139–140. Available from: https://doi.org/10.18420/SE2020_42.

[52] Stolee, K.T. and Fristoe, T., 2011. Expressing computer science concepts through kodu game lab. *SIGCSE'11 - Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*. pp.99–104. Available from: https://doi.org/10.1145/1953163.1953197.

[53] Sá Silva, P., Pedrosa, D., Trigo, A. and Varajão, J., 2011. Simulation, games and challenges: From schools to enterprises. *Lecture Notes in Business Information Processing*, 88 LNBIP, pp.63–73. Available from: https://doi.org/10.1007/978-3-642-24175-8_5.

[54] Sánchez-López, I., Roig-Vila, R. and Pérez-Rodríguez, A., 2022. Metaverse and education: the pioneering case of minecraft in immersive digital learning. *Profesional de la Informacion*, 31(6). Available from: https://doi.org/10.3145/epi.2022.nov.10.

[55] Takbiri, Y., Bastanfard, A. and Amini, A., 2023. A gamified approach for improving the learning performance of K-6 students using Easter eggs. *Multimedia Tools and Applications*, 82(13), pp.20683–20701. Available from: https://doi.org/10.1007/s11042-023-14356-7.

[56] Taylor, S., Min, W., Mott, B., Emerson, A., Smith, A., Wiebe, E. and Lester, J., 2019. Position: IntelliBlox: A Toolkit for Integrating Block-Based Programming into Game-Based Learning Environments. *Proceedings - 2019 IEEE Blocks and Beyond Workshop, B and B 2019*. pp.55–58. Available from: https://doi.org/10.1109/BB48857.2019.8941222.

[57] Theodosi, A. and Papadimitriou, V., 2011. The synergy of three: Incorporating games, multimedia and programming in order to improve algorithmic skills. *Proceedings of the European Conference on Games-based Learning*. vol. 2011-January, pp.582–594.

[58] Thomas, J.O., 2015. Supporting Computational Algorithmic Thinking (SCAT): Exploring the difficulties African-American middle school girls face while enacting computational algorithmic thinking. *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*. vol. 2015-June, pp.69–74. Available from: https://doi.org/10.1145/2729094.2742605.

[59] Thomas, J.O., Rankin, Y., Minor, R. and Sun, L., 2017. Exploring the Difficulties African-American Middle School Girls Face Enacting Computational Algorithmic Thinking over three Years while Designing Games for Social Change. *Computer Supported Cooperative Work: CSCW: An International Journal*, 26(4-6), pp.389–421. Available from: https://doi.org/10.1007/s10606-017-9292-y.

[60] Van Eck, R., Hung, W., Bowman, F. and Love, S., 2009. 21st century game design: A model and prototype for promoting scientific problem solving. *Proceedings of the 12th IASTED International Conference on Computers and Advanced Technology in Education, CATE 2009*. pp.219–227.

[61] Varghese, V.V.V. and Renumol, V., 2024. Video games for assessing computational

thinking: a systematic literature review. *Journal of Computers in Education*, 11(3), pp.921–966. Available from: https://doi.org/10.1007/s40692-023-00284-w.

[62] Yusuf, A. and Noor, N.M., 2024. Modeling students' algorithmic thinking growth trajectories in different programming environments: an experimental test of the Matthew and compensatory hypothesis. *Smart Learning Environments*, 11(1), p.38. Available from: https://doi.org/10.1186/s40561-024-00324-7.

[63] Zinchenko, Y.M., 2023. Vykorystannia Blockly Games pid chas vyvchennia Prohramuvannia. *Suchasni informatsiini tekhnolohii v osviti i nautsi : materialy KhIV Vseukr. nauk.-prakt. konf. dlia molodykh uchenykh ta zdobuvachiv osvity, m. Uman, 16–17 bereznia 2023 r.* Uman, pp.34–35.