From Factory Floors to Code: Designing a Digital Escape Room for Introductory Programming

Gijs Nieuwkoop

2026

Abstract

Introductory programming is difficult for many secondary-school students because concepts such as variables, types, casting, arithmetic, and conditionals require learners to connect abstract code structures to dynamic program behaviour. This study investigates how a digital educational escape room based on scaffolding and the semantic wave can contribute to the process of learning fundamental programming concepts. A digital escape room called *The Factory* was designed and tested with 13 Dutch VWO students in six groups. In the game, students configured factory production lines in which boxes represented variables and values, machines represented operations, and routing behaviour represented conditionals. The game consisted of 24 sequential levels that introduced concepts through concrete factory mechanics before gradually shifting toward more abstract programming-like representations.

The study used a mixed-methods descriptive case-study approach. Data were collected through in-game logs, audio and video recordings of student interaction, and researcher observation notes. The analysis focused on process markers rather than formal learning gain, including completion time, attempts, registered mistakes, support use, collaboration, and student movement between concrete game mechanics and abstract programming concepts.

The results show that the escape room created observable opportunities for students to reason between concrete factory representations and programming concepts. Students used terms such as *string*, *int*, *var*, *if*, and *else* while discussing boxes, colours, machines, routes, and values. Difficulty increased especially at levels that required an upward shift toward more programmatic representations or the combination of earlier concepts. Several scaffolded level sequences showed reduced completion times and fewer attempts after an introductory level, particularly in the conditional sequence. However, the casting sequence showed that scaffolding was not equally effective throughout the game, as the step from introduction to practice was too large for some groups.

The findings indicate that a digital educational escape room can support the learning process of introductory programming by making abstract concepts visible, discussable, and connected to concrete actions. However, the results should be interpreted as process evidence rather than proof of learning gain. Future versions should refine the pacing of abstraction, improve hint integration, reduce interface-related difficulty, and include the debriefing phase more explicitly in the research design.

1 Introduction

Programming is a core component of *Digital Literacy* in Dutch secondary education [1]. Beyond its disciplinary value, it fosters computational thinking, a prerequisite for many modern professions. The national computer-science syllabus explicitly requires that students can read and write code [2]. Despite this mandate, learners often struggle with basic concepts such as sequencing, conditionals and loops. Common misconceptions include executing both branches of an *if–else*, misunderstanding loop termination and treating assignment as reversible. These difficulties stem from abstract syntax, weak mental models and limited problem-solving strategies [3], [4].

Research suggests that visual programming languages lower syntactic load and improve concept acquisition [5]. However, motivation and sustained engagement remain problematic. Game-based learning, and more specifically educational escape rooms, address this issue by embedding content in interactive, goal-oriented contexts. Prior studies report increases in motivation and perceived learning when game elements align with educational objectives [6], [7]. Escape rooms add time pressure and collaboration, creating a setting where learners must apply knowledge to progress the narrative of an Escape Room, which can make abstract ideas tangible.

1.1 Research objective

The present study investigates how a digital escape room designed for fundamental programming can support concept learning for secondary school students. The escape room will connect puzzle mechanics directly to sequencing, conditionals and loops, employ scaffolding that fades with learner competence and implement semantic-wave shifts between concrete tasks and abstract principles. Effectiveness will be measured through in-game analytics and student recordings.

By integrating established challenges in learning programming with the motivational benefits of game-based environments, this research aims to provide empirical guidance on the design of digital escape rooms that foster an accurate understanding of introductory programming concepts.

1.2 Basic Concepts in Programming

Programming, in the narrow sense, is defined by SLO as expressing an algorithm in a language that can be executed by a computer. In the programming domain of the Dutch computer science curriculum, SLO focuses mainly on imperative programming languages. In these languages, programs consist of step-by-step instructions that are executed in a prescribed order. SLO identifies the basic building blocks of imperative programming as data objects, assignment, control structures, and abstraction mechanisms such as procedures and functions [2]. These concepts form the basis of the programming content used in this study.

Variables and types Variables are named data objects used to store values in a program. Assignment is the operation by which a value is given to a variable. A variable can then be used later in the program to refer to the stored value. Values can have different types, such as integers or strings. Types determine what kind of data a value represents and which operations can be performed on it. Changing the type of a variable is called casting the variable.

Sequencing Sequencing, or *opeenvolging*, is the execution of instructions in a fixed order. In an imperative program, the order of instructions is important because each instruction can affect the state of the program. For example, assigning a value to a variable before using it produces a different result than using the variable before the assignment has taken place.

Conditionals Conditionals are control structures for choice, or *keuze*. They allow a program to select between different instructions based on whether a condition is true or false. A common example is an `if`-statement, where one block of code is executed only if a given condition holds.

Loops Loops are control structures for repetition, or *herhaling*. They allow a program to execute the same instructions multiple times. The repetition can depend on a condition, as in a `while`-loop, or on a fixed range or collection, as in a `for`-loop.

Functions and parameters Functions are abstraction mechanisms that describe a piece of program separately and allow it to be called elsewhere in the program. According to SLO, functions are used for pieces of program that produce a value, while procedures or methods are used for pieces of program

that perform an action. Functions can use parameters, which are values passed into the function when it is called. This allows the same function to be reused with different input values.

1.3 Challenges

Students face various challenges when learning to program, including understanding abstract concepts like algorithm design and particularly loops. Students commonly struggle with misconceptions related to syntactic, conceptual, and strategic knowledge [3]. These include unfamiliarity with syntax, inaccurate mental models, and lack of effective problem-solving strategies, as seen in table 1. Issues such as confusion over the purpose and function of loops, difficulties understanding sequentiality, and misconceptions about variables are also prevalent [4]. Moreover, students often exhibit a flawed understanding of how control structures work, believing, for instance, that both branches of an 'if-else' statement can be executed simultaneously [3]. In addition, factors like lack of problem-solving abilities, inadequate instructional materials, and students' negative perceptions about programming contribute significantly to these difficulties [8]. The dynamic nature of programming concepts is also challenging to explain through static teaching materials, leading to further confusion. These challenges highlight the importance of using effective teaching methods that cater to students' learning needs and focus on both syntax and problem solving skills.

Concept	**Typical student difficulty**
Variables	
	Belief that a variable can remember multiple values successively
	Assuming that printing "X = 1" actually changes the value of `X`
	Treating `5 = A` as equivalent to `A = 5`; assignment seen as reversible
	Natural-language meaning of names dictate values (e.g. `largest` must be big)
	Thinking the variable exists only on-screen, not in memory
Loops	
	Misidentifying which lines are repeated inside a loop body
	Expecting a loop to stop as soon as the condition first becomes false, even mid-iteration
	Difficulty choosing between `for` and `while`
	Reading "while" literally and assuming endless execution without condition being re-tested
Conditionals	
	Expectation that both branches of an `if-else` execute
	Belief that the whole program halts if a lone `if` test is false
	Sequentiality & Later assignments presumed to retroactively update earlier results
	Input & Assuming the script continues instantly, or the computer supplies its own input

Table 1: Common misconceptions and challenges in introductory programming

2 Gaming and Education

The use of games in education is often categorized into three main areas: gamification, serious games, and game-based learning (GBL) [9].

2.1 Gamification

Gamification is the application of game design elements, such as points, badges, levels, and challenges, in non-game environments to foster user engagement and motivation. Unlike a true game, gamification refers to "gameful" experiences, fostering structured, goal-oriented interaction. It aims to make activities in areas such as education, productivity, health, and marketing more enjoyable and engaging by integrating aspects of what makes games compelling [10].

2.2 Serious Games

Serious games are games that are specifically designed for educational or training purposes, rather than only for entertainment. They offer a unique opportunity to simulate real-world scenarios, allowing learners to practice and develop skills in a controlled and immersive environment. For example, serious games can be used to teach programming by having students solve puzzles that require coding solutions, thus providing practical experience in a fun and engaging manner.

2.3 Game-Based Learning

Game-based learning refers to the use of actual games as tools for learning and entertainment [11]. GBL leverages the interactive nature of games to help students learn concepts and skills by doing, exploring, and experimenting. Games used in GBL often present challenges that require students to apply knowledge and think critically to solve problems. This hands-on approach can help deepen understanding and improve retention of complex topics such as programming [12].

2.3.1 Why Game-Based Learning

Game-based learning is effective because it taps into the core psychological needs that drive human motivation. According to Self-Determination Theory [13], individuals are motivated when they experience autonomy, competence, and relatedness. GBL provides learners with the autonomy to make choices within the game, the opportunity to build competence through challenges and feedback, and a sense of relatedness when collaborating or competing with others. These elements help to make learning more engaging and meaningful.

Furthermore, GBL helps students learn by doing, rather than passively consuming information. By actively participating in problem solving and experimentation, students can better internalize concepts and experience how they are applied in real world situations. The immersive nature of games also reduces the anxiety often associated with learning complex subjects, allowing students to make mistakes and learn from them in a safe environment.

2.3.2 GBL Design Building Blocks

Designing an effective game-based learning experience requires several key elements to ensure alignment between educational content and gameplay. The following building blocks contribute to making a good educational game [14]:

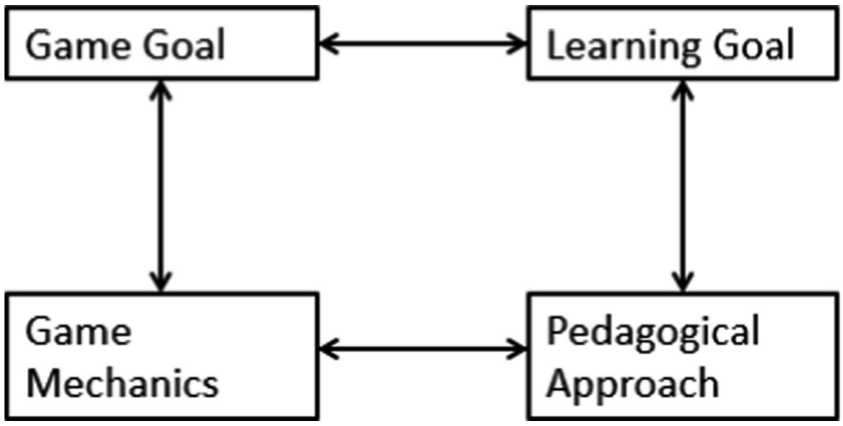

Figure 1: Alignment of design elements within an educational game [14]

1. **Alignment of Learning and Game Goals:** The learning goal should be intrinsically linked to the game goal. Players should only be able to achieve the game objective if they demonstrate the intended learning outcome. This alignment helps ensure that players remain focused on educational objectives while engaging with the game.

2. **Integration of Game Mechanics and Educational Content:** The game mechanics should reflect the educational content. Game mechanics must support and reinforce the learning process by enabling players to practice relevant skills or apply knowledge in a meaningful way. For example, if the learning goal involves understanding forces, then the game mechanics should require players to apply forces to objects within the game to see the resulting effects.

3. **Clear Pedagogical Approach:** The game should incorporate a clear pedagogical approach that guides the design of game mechanics. This approach could involve problem posing, scaffolding, or feedback mechanisms that help players confront and overcome misconceptions. The pedagogical framework should align with the desired learning outcomes.

4. **Player Experience and Engagement:** A good educational game keeps players in a state of flow, where the challenges presented match the players' skill levels. This balance is crucial for maintaining motivation and preventing frustration or boredom. The game should offer increasing levels of difficulty, providing an appropriate challenge as players' skills develop.

5. **Feedback and Reflection:** Immediate feedback is an essential part of learning in games. Players should be given information on their performance, allowing them to reflect on their actions and understand the consequences of their decisions. This process helps reinforce learning and encourages players to adjust their strategies.

6. **Immersive Context:** The game environment should be immersive, engaging players in a narrative or scenario that is relevant to the learning content. The context should make the educational material meaningful by situating it in a scenario that feels real or engaging, helping students connect abstract concepts to practical applications.

2.4 Game-Based Learning in computer science

Game-based learning (GBL) has gained increasing attention as an effective pedagogical approach in computer science education, especially for improving student motivation, engagement, and teamwork. Videnovik et al 2023 [7] highlights that GBL is most commonly applied to topics like programming

and computational thinking, with studies reporting improvements in problem-solving, creativity, and collaboration. Despite its broad adoption, there is no standardized game format or pedagogical approach, and most implementations rely on learning by playing rather than designing games. Hosseini et al 2019 [6] emphasizes the potential of non-digital, modular game activities to enhance students' perception of learning and engagement in higher education. Their study found that even brief game-based sessions can increase participation, particularly among students who typically remain passive in traditional settings. While GBL has demonstrated promise, both reviews note the need for rigorous assessment methods and careful alignment with learning objectives to avoid shifting focus away from core content.

3 Educational Escape Rooms

Educational escape rooms are increasingly being used in science education to foster engagement and improve learning outcomes [15]. Escape rooms are effective because they provide a hands-on, collaborative learning environment where students must apply their knowledge to solve puzzles and achieve a common goal. The time-constrained, immersive setting encourages active participation and promotes problem-solving and teamwork skills. Escape rooms also offer a unique opportunity for formative assessment, allowing teachers to observe students' understanding in real-time. The use of storytelling and puzzles helps create an enjoyable learning experience that enhances student motivation, making complex science topics more approachable and memorable.

3.1 Design Guidelines

According to Veldkamp et al's research on the use of escape rooms in secondary science education [15], creating a good educational escape room requires a balance of educational and game design principles. They stipulate the following design principles:

1. **'Dare to "leave" the room'** Setting up a single escape room for a whole class can be challenging. Creating an escape room that is not an actual room, but is more like multiple stations that students have to check out, or a single box that students have to try to open, can be preferable. Existing technology can be used for this and can be used to structure puzzle paths, validate answers, and give the students hints if they fall behind.

2. **'Create alignment'** To design effective educational escape rooms, align learning goals, game objectives, pedagogy, and game mechanics. Select pedagogical approaches that fit game elements like puzzle types, structure, and team size. For team-based or collaborative learning, use path-based or hybrid puzzle structures to promote interdependence among players. A hybrid structure with linear design helps students and simplifies monitoring progress. Additionally, individual puzzles can encourage mutual interdependence by requiring various skills or dividing information among team members.

3. **'Strive for high success rates'** A high success rate on puzzles tends to improve the students' learning experiences and helps them achieve the learning goals better. This can be done by adjusting and testing of an adequate level of difficulty in the puzzles.

4. **'Allow all groups to finish the escape game'** Teams should play against time, not against other students.

5. **'Design the role of the teacher'** A teacher can monitor and guide students better if they are in the same room. Being present and intervening can be disruptive for the feelings of immersion and autonomy the students experience. The presence and intervention itself does not seem to be the problem here, but more the way the teacher is incorporated into the narrative. It is recommended to give the teacher a role in the narrative so they can be questioned without disrupting the immersion.

6. **'Design specific for grading'** When the students are graded on their performance during gameplay, it is better to stick with small team sizes and sequential puzzle paths. What is graded should be dictated by the intended learning goals. Grading does not need to be used as a precautionary measure to active students. The medium itself already activates them enough.

7. **'Consider hybrid learning spaces'** Hybrid learning spaces can help students feel immersed, making it easier for them to switch from a school setting to a game environment. This works best when using real-world science-related scenarios.

4 Pedagogical Approaches

4.1 Scaffolding

Research in computer science education has increasingly adopted scaffolding techniques to support student learning. Scaffolding refers to the supportive interactions between a tutor and a student that enable the student to accomplish tasks they could not complete alone [16]. This support involves simplifying the task, maintaining the student's focus, highlighting important features, managing frustration, and demonstrating correct solutions when needed. As the student's competence grows, the tutor gradually reduces assistance, allowing the student to gain independence in problem-solving. This process fosters skill development by guiding learners through challenges just beyond their current capabilities.

Min et al 2014 [17] have integrated scaffolding into GBL environments to teach programming concepts to secondary school students. These environments provide adaptive guidance such as personalized hints and dynamic task selection, helping learners navigate problem-solving tasks at their own pace while avoiding frustration or disengagement. Scaffolding is also being applied in higher education contexts as demonstrated by Piotrowska et al 2020 [18], particularly through the integration of Content and Language Integrated Learning (CLIL). In these settings, scaffolding supports not only domain-specific content, such as data mining and computer simulation, but also the acquisition of relevant English terminology. This dual focus helps computer science students engage with authentic scientific texts and tools, gradually reducing support as their competence grows. These examples illustrate the versatility of scaffolding approaches in computer science education, ranging from adaptive feedback in intelligent tutoring systems to structured linguistic and conceptual support in higher education.

Scaffolding can be effectively integrated into educational escape rooms, firstly, through the use of hints. In-game hints function as adaptive scaffolds, offering targeted support as learners gain knowledge and skills. Secondly, the puzzle sequence itself can serve as a scaffold when each puzzle builds incrementally on the previous one, enabling students to apply prior solutions to more complex problems. This layered structure aligns the pedagogical approach with the game mechanics and the learning goals as outlined by Van der Linden et al 2019 [14], ensuring that learning support is integrated into the gameplay experience.

4.2 Semantic Wave

The concept of the Semantic Wave can help students understand complex material easier [19] and has proven to be an effective way of teaching in computer science [20]. It describes how knowledge is built by shifting between abstract concepts and specific examples. This movement is called a "wave" because of the way Semantic Gravity and Semantic Density interact with one another during the explanation of the material, much like waves in the ocean.

- **Semantic Gravity (SG)** means how closely a piece of information is tied to its context. If something has strong semantic gravity (SG+), it makes sense only in a specific situation or example. For instance, saying "Water boils at 100°C at sea level" is very specific and only works in the context of Earth. On the other hand, weak semantic gravity (SG-) applies to ideas that are more general and can be used in many contexts. An example would be the concept of "Boiling is the process where a liquid turns into a gas" which is can be applied to any environment.

- **Semantic Density (SD)** is about how much meaning or information is packed into a term or idea. Strong semantic density (SD+) means the term holds a lot of meaning and might need explanation. For example, the word "photosynthesis" includes complex ideas about how plants turn sunlight into food. Weak semantic density (SD-) means the idea is simple and easier to understand, like saying "plants make food using sunlight."

When teaching or learning, semantic waves occur as the lessons move between examples and general ideas. This process has two main movements:

- **Downward Shift (SG+, SD-)**: This happens when a complex idea is broken down into simpler examples. For example, a history teacher might explain "democracy" by talking about voting in a school election.
- **Upward Shift (SG-, SD+)**: This occurs when learners take specific examples and connect them to a larger idea. After talking about voting in a school, the teacher might explain how democracy works in countries around the world.

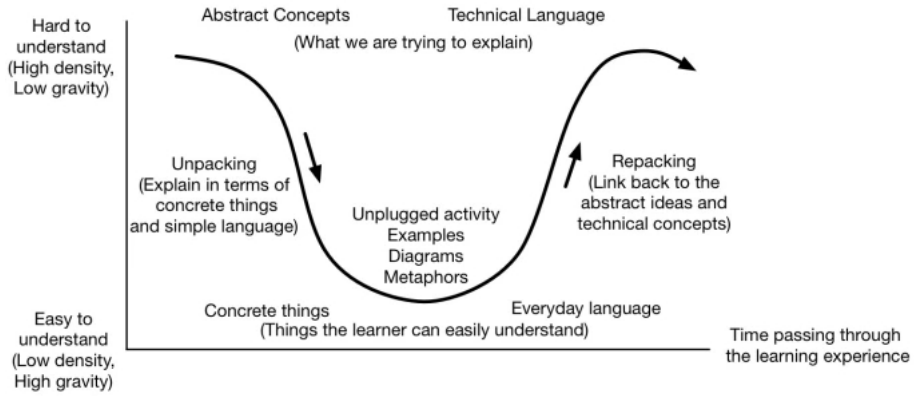

Figure 2: The Semantic Wave by Maton et al 2013 [19]

Using semantic waves helps students understand both simple examples and the related larger concepts. When lessons focus only on abstract ideas, students might get confused. If we stick to simple examples, students might not learn how to apply what they know in new situations. Teachers can create effective lessons by balancing both movements, helping students build a deeper and broader understanding of what they are learning.

This approach has proven an effective pedagogical approach in computer science education, where abstract concepts such as algorithms, variables, and assignment often pose significant challenges to learners. [20] applied the Semantic Wave framework to analyze and improve unplugged computing activities, lessons that teach computational thinking without the use of computers. By mapping how lessons move between abstract technical terms and concrete, everyday experiences, they showed that activities that include clear unpacking (moving from abstract to concrete) and repacking (moving back from concrete to abstract) help students form a deeper conceptual understanding. For example, in their "Teleporting Robot" and "Box Variables" case studies, improvements to teaching activities based on semantic profiling led to more meaningful learner engagement with key computing concepts. This demonstrates how the Semantic Wave can be a practical and valuable tool for designing and refining computer science lessons that promote both accessibility and conceptual mastery.

The pedagogical approach of Semantic Waves reinforce the alignment framework proposed by [14] by organizing how learners move between abstract and concrete knowledge within an educational escape room. Each puzzle sequence can begin with a downward shift, where abstract programming

concepts are introduced through concrete, context-rich challenges. As students progress, successive puzzles guide them through an upward shift, gradually connecting these contextual experiences back to the underlying programming principles. During the debriefing the upward shift ends with discussing the actual abstract programming concepts. This progression ensures that game mechanics support the applied pedagogical approach.

5 Research

5.1 Research Question

The previous sections show that introductory programming is difficult for many students because it requires them to work with abstract concepts such as variables, sequencing, conditionals, and loops. These difficulties are not limited to writing correct syntax, as students also need to develop accurate mental models of how programs are executed and how programming concepts relate to one another.

The literature also suggests that game-based learning and educational escape rooms can create active and engaging learning environments. However, the educational value of such environments depends on more than motivation alone. The game structure, puzzle design, and pedagogical approach need to support the learning process in a way that helps students connect concrete tasks to the underlying subject matter.

This leads to the following research question:

How does an educational escape room based on scaffolding and the semantic wave contribute to the process of learning fundamental programming concepts for secondary students?

5.2 Method

This study used a mixed-methods descriptive case-study approach to investigate how a digital educational escape room could support the learning process of fundamental programming concepts among secondary-school students. The study focused on process indicators rather than on measuring learning gain through a formal pre-test and post-test design. The intervention was analysed through in-game log data, recorded student interaction, and researcher observation notes.

The intervention was designed according to principles from educational escape room design, game-based learning, scaffolding, and the semantic wave. The game was then tested with student groups in a classroom setting. The analysis focused on how students progressed through the levels, where difficulty points occurred, how students used available support, and whether the recordings contained markers of students moving between concrete game representations and abstract programming concepts.

5.2.1 Participants and prior experience

This study was conducted with 13 students aged 16 to 18 in the 4th and 5th year of VWO in dutch education, all from the Kalsbeek College in Woerden where I also teach. The students played the escape room in six groups. Five groups consisted of two students, and one group consisted of three students. Prior programming experience differed between the groups as seen in table 2. The level of prior experience was recorded informally as none, little, or a lot.

Table 2: Student groups and prior programming experience

Group	Students	Prior experience
1	Student 1, student 2	Little, none
2	Student 3, student 4	None, a lot
3	Student 5, student 6	Little, little
4	Student 7, student 8	None, little
5	Student 9, student 10	Little, little
6	Student 11, student 12, student 13	None, none, none

Groups 1 and 5 did not reach levels 20 to 24 within the available time. As a result, the averages for the later levels are based on fewer groups than the averages for the earlier levels.

5.2.2 Digital Educational Escape Room Design

The intervention, titled *The Factory*, is a digital educational escape room built by me in which students acted as engineers configuring production lines in a factory setting. Students solved levels by placing and configuring machines so that boxes moved through the factory correctly. Each completed level unlocked the next level.

The game consisted of 24 sequential levels. These levels either introduced a new concept, practiced earlier concepts, or were used as an upward shift of the semantic wave, followed by a final cumulative level. The sequential structure was chosen to reduce path variability, to make progression easier to monitor, and to have previous levels be able to function as a scaffold for new levels. The levels introduced programming concepts through concrete factory mechanics before gradually moving toward more programmatic representations of programming. Each level was designed to be short and iterate slightly on the previous one, giving students a lot of success experiences. As the game is meant to be played in a classroom setting, the length of the game has to be taken into consideration. The game will therefore focus on the basic concepts of variables and conditionals.

The game mechanics were designed to align with the learning goals. Variables were represented as boxes with values, types were represented through box properties such as color, operations were represented through machines, and conditionals were represented through routing behavior. Later levels increasingly required students to use code-like syntax and to combine earlier concepts.

Figure 3: Screenshots from level 2 of The Factory

Scaffolding was built into the level sequence in three ways. First, new concepts were introduced in simpler levels before being reused in more complex levels. Second, interaction patterns were kept visually and mechanically similar within related levels, so that students could focus on the programming concept rather than relearning the interface. Third, support was available through information screens, hints, and teacher support when needed.

The semantic-wave design was implemented through shifts between concrete and abstract representations. Earlier levels used factory objects and visual behavior to introduce concepts. Later levels required students to connect these concrete experiences to more abstract programming notation. The intended movement was therefore from concrete, context-rich representations toward more abstract programming principles. An overview per level can be seen in table 3.

The levels were designed to move between concrete factory-based representations and more abstract programming-like representations. Earlier levels introduced variables, casting, and arithmetic through concrete game mechanics. Later levels increasingly required students to use more abstract or programmatic representations. Conditionals were introduced later in the sequence and were then combined with earlier concepts.

Table 3: Overview of level content

Level	Concept	Focus
1	Variables	Boxes and value assignment
2	Checks	Checking box values
3	Types	Casting between strings and integers
4	Arithmetic	Adding integers
5	Arithmetic	Practice of level 4
6	Variables	Programmatic declaration
7	Casting	Programmatic casting
8	Casting	Practice with added complexity
9	Casting and arithmetic	Combined use
10	Casting and arithmetic	Practice with added complexity
11	Arithmetic and types	Strings and integers
12	Arithmetic and types	Practice of level 11
13	Casting, arithmetic and types	Combined use
14	Conditionals	Introduction
15	Conditionals	Practice
16	Conditionals	Further practice
17	Conditionals	Programmatic form
18	Conditionals	Practice of level 17
19	Conditionals	Practice with added complexity
20	Conditionals	Closer to regular programming
21	Casting and conditionals	Combined use
22	Arithmetic, casting and conditionals	Combined use
23	Variables and conditionals	Reusing variable values
24	Cumulative	Final combined level

Hints and information screens Each level could contain an information screen and one or more hints. Information screens provided the concept or task information and were always available without cost. Hints were level-specific and could be unlocked when students needed additional support. Students could also ask the teacher for additional help as well. Teacher interventions were kept limited where possible, so that the game itself remained the main source of guidance.

Debriefing After the play session, the teacher could connect the game mechanics back to programming concepts. The debriefing was intended to complete the upward movement of the semantic wave by explicitly linking factory actions, such as routing boxes or changing types, to programming concepts such as variables, casting, arithmetic, and conditionals. The present results focus on the gameplay data and recording markers from the session itself.

5.3 Implementation

The digital escape room was implemented in Godot 4.5.1. Godot was chosen because it is an open-source game engine and because the project did not require graphically intensive features or complex development tools. The game mainly consisted of simple interactive objects, level logic, user input, and logging functionality. Godot was therefore sufficient for building and testing the factory-based puzzle environment, while also allowing the development process to remain accessible and adaptable.

5.3.1 Data Collection

The study used three main data sources.

- **In-game analytics:** The digital platform logged quantitative data for each group and level, including time spent, number of attempts, registered mistakes, hints used, information screens opened, previous-level jumps, and level completion. For groups that did not finish the game, periodic logs were used to recover progression data up to the final reached level.
- **Audio and video:** Student interaction during gameplay was recorded using the webcam of the laptop and a separate audio recorder. For one group both recordings were corrupted and could not be used for analysis, and for another group both the webcam recording and audio recorder crashed shortly after the start, with the students noticing after finishing the third level and restarting the recording. The recordings were used to identify process markers such as collaborative reasoning, references to earlier levels, use of programming vocabulary, interface confusion, and moments where students connected factory mechanics to programming concepts. These recordings were automatically transcribed using a digital tool, with manual touch ups in seemingly important segments.
- **Observation notes:** Researcher observations were used to contextualize log and recording data, especially when interpreting unusual completion times, support use, or moments where students appeared to be stuck.

5.3.2 Data Analysis

The analysis combined descriptive quantitative analysis of the log data with qualitative analysis of the recordings.

For the quantitative analysis, level-level statistics were compared across the 24 levels. The main indicators were completion time, number of attempts, registered mistakes, hint use, information-screen use, previous-level jumps, and completion status. Particular attention was given to levels that introduced new representations, levels that reused earlier mechanics, and levels that combined multiple concepts.

For the semantic-wave analysis, levels were examined for markers of movement between concrete factory representations and abstract programming representations. Increased time, attempts, or mistakes on upward-shift levels were treated as indicators of difficulty during abstraction. Recording fragments were then used to clarify whether students discussed both the concrete game behavior and the corresponding programming concepts.

For the scaffolding analysis, related level pairs were compared. A decrease in time, attempts, or registered mistakes after an introductory level was treated as a possible marker that the earlier level supported the later one. Hint use, information-screen use, and previous-level jumps were analyzed as additional support markers. Recording fragments were used to examine whether students referred back to earlier levels or used provided information during problem solving.

The analysis does not treat log patterns as direct proof of learning. Instead, the results describe markers of how students interacted with the designed learning environment.

Table 4: Operationalization of process markers used in the analysis

Analytical focus	**Quantitative markers**	**Recording markers**	**Interpretation in this study**
Semantic-wave movement	Increased completion time, attempts, or registered mistakes on levels that introduced a more abstract or programmatic representation.	Students connected concrete factory mechanics to programming concepts, or discussed the difference between game actions and code-like notation.	Treated as evidence that students were moving between concrete representations and more abstract programming concepts.
Scaffolding	Decreased completion time, attempts, or registered mistakes on levels that reused or extended a previously introduced concept. Hint use, information-screen use, and previous-level jumps were also considered.	Students referred back to earlier levels, reused earlier strategies, used provided information, or helped each other apply a previously introduced concept.	Treated as possible evidence that earlier levels or available support helped students approach later tasks.
Interface or representation difficulty	High completion time or repeated attempts that did not clearly correspond to a new programming concept.	Students discussed controls, placement, input format, or game mechanics rather than the intended programming concept.	Treated as a possible indication that difficulty came from the interface or representation rather than from the programming concept itself.
Engagement and collaboration	Continued progression after mistakes, repeated attempts, and voluntary use of available support.	Students persisted after failure, discussed possible solutions, showed positive reactions, or divided problem-solving work.	Treated as process-level evidence of participation and collaboration, not as a direct measure of motivation or learning gain.

5.3.3 Ethical Considerations

This research adhered to strict ethical guidelines. Informed consent was obtained from the schools, and the student participants themselves. As the participants are at least 16 years or older, no consent was needed from parental figures, in compliance with the EU General Data Protection Regulation. Participants were informed that their participation is voluntary and that they could withdraw at any time without penalty. All collected data (recordings and logs) was anonymized to protect participant privacy. Data is stored securely and used solely for research purposes only. It was made clear that

participation or the lack there of would have no impact on any grades the students will receive in their future.

6 Results

The main analysis reports selected level statistics in Table 5. The full group-level graphs for time, attempts, mistakes, hint use, information-screen use, and previous-level jumps are included in Appendix B.

6.1 Overall level progression

The average time to complete each level varied strongly across the game. Several levels showed a large increase in time to complete compared with surrounding levels. The largest average times were found at level 7, level 8, level 9, level 17, level 19, and level 24, as can be seen in table 5.

Table 5: Selected level statistics relevant to semantic-wave and scaffolding markers

Level	Design role	Pairs	Avg. time (s)	Avg. attempts
7	Programmatic casting	6	321.02	9.17
8	Casting practice with added complexity	6	368.41	10.17
9	Casting and arithmetic	6	297.44	5.83
10	Practice of level 9	6	166.83	2.00
17	Programmatic conditionals	6	555.73	14.83
18	Practice of level 17	6	186.01	3.00
19	Programmatic conditionals with added complexity	6	403.13	6.50
24	Final cumulative level	4	488.46	6.75

These levels correspond with upward shifts, combinations of earlier skills, or increases in complexity. Level 7 introduced programmatic casting. Level 8 practiced this with increased complexity. Level 9 combined casting and arithmetic. Level 17 introduced programmatic conditionals. Level 19 further practiced programmatic conditionals with increased complexity. Level 24 combined several earlier skills in a final cumulative level.

Some levels following a difficult level showed lower average time, attempts, and registered mistakes. For example, level 9 had an average completion time of 297.44 seconds, while level 10 had an average completion time of 166.83 seconds. Level 11 had an average completion time of 128.69 seconds, while level 12 had an average completion time of 63.97 seconds. Level 17 had an average completion time of 555.73 seconds, while level 18 had an average completion time of 186.01 seconds.

6.2 Semantic wave markers

Several levels were designed as upward shifts from concrete game mechanics toward more abstract programming representations. These upward-shift levels often coincided with increases in time to complete and attempts.

Level 6 introduced programmatic variable declaration after earlier concrete work with boxes and values. The average completion time increased from 62.64 seconds on level 5 to 97.38 seconds on level 6. The average number of attempts increased from 1.17 to 3.33.

Level 7 introduced programmatic casting after casting had first been introduced through a more concrete representation. This level showed a large increase in time to complete, attempts, and registered mistakes. The average completion time increased from 97.38 seconds on level 6 to 321.02 seconds on level 7. The average number of attempts increased from 3.33 to 9.17, and the average number of registered mistakes increased from 0.67 to 5.83.

The recordings provide additional detail about the nature of this upward shift. During the casting levels, students discussed the relation between the visual representation and the programming notation. They referred to the color system of the game while also using programming terms such as *string*, *int*, and *var*. In some cases, students explicitly connected a color to a type, for example by identifying that strings and integers corresponded with different colored boxes T1, approx. 00:14:40–00:16:40, T3 approx. 00:18:45–00:20:55, T6, approx. 00:07:30–00:08:05.

At the same time, the recordings show that this transition was difficult. Students sometimes confused the name of a variable, the value stored in the variable, and the type of that value. Several fragments show students debating whether *int* or *string* referred to the type, the value, or the name of the box. Students also discussed whether type conversion required brackets and whether a variable still referred to the same value after conversion T1, approx. 00:14:40–00:16:40, T3, approx. 00:05:50–00:07:40, T6, approx. 00:07:30–00:10:20. These observations match the high completion times and attempts on levels 7 and 8.

Level 17 introduced conditionals in a programmatic form after earlier concrete conditional levels. This was the level with the highest average completion time, the highest average number of attempts, and the highest average number of registered mistakes. The average completion time was 555.73 seconds, with 14.83 attempts and 11.17 registered mistakes.

Recordings show that the difficulty of level 17 was connected to the shift from concrete routing behavior to programmatic conditional notation. Students discussed whether boxes should move up, down, left, or right after a condition was checked, and compared this visible movement to the code they had written. In several cases, students appeared to understand part of the conditional structure, but still struggled to connect the result of the condition to the spatial route taken by the box T1, approx. 01:08:50–01:10:25, T2, approx. 00:35:35–00:37:05, T6, approx. 00:39:00–00:43:00.

The recordings also show syntax-related uncertainty in the conditional levels. Students discussed whether code should contain one or two instances of *is*, whether capitalization mattered, and how conditional commands should be written T5, approx. 00:02:45–00:03:25, T1, approx. 01:11:20–01:12:55, T2, approx. 00:53:35–00:54:55.

Later upward-shift levels showed lower values in the log data. Levels 20 to 23 were designed as further upward shifts and combinations of earlier concepts. These levels had lower average completion times and few registered mistakes compared with levels 7, 8, 17, 19, and 24.

These later levels were only reached by four of the six groups. Groups 1 and 5 did not reach these levels, so these averages are based on the groups that progressed furthest.

The recordings contain some indications that students were able to move between the concrete and abstract representations during later levels. Students sometimes used programming terms while still referring to the visible factory behavior. For example, they discussed *if* and *else* statements in relation to the direction of box movement, or used variable and type terminology while reasoning about boxes and machines T1, approx. 00:45:40–00:47:00, T2, approx. 00:53:35–00:55:00. These moments are relevant as semantic-wave markers because they show students moving between the game context and the abstract programming concepts during problem solving.

6.3 Scaffolding markers

The level sequence included repeated practice after concept introduction. Several pairs of related levels show a pattern in which a difficult level is followed by a related level with lower average completion time, fewer attempts, and fewer registered mistakes.

Level 9 combined casting and arithmetic. The average completion time was 297.44 seconds, with 5.83 attempts and 1.00 registered mistake. Level 10 practiced the same skill area with increased complexity. The average completion time decreased to 166.83 seconds, with 2.00 attempts and 0.00 registered mistakes.

Level 11 introduced arithmetic with both strings and integers. The average completion time was 128.69 seconds, with 3.50 attempts and 0.33 registered mistakes. Level 12 practiced the skills from level

11. The average completion time decreased to 63.97 seconds, with 1.50 attempts and 0.00 registered mistakes.

Level 17 introduced programmatic conditionals. The average completion time was 555.73 seconds, with 14.83 attempts and 11.17 registered mistakes. Level 18 practiced programmatic conditionals. The average completion time decreased to 186.01 seconds, with 3.00 attempts and 1.50 registered mistakes.

The recordings add qualitative detail to these scaffolding patterns. Students sometimes referred back to earlier mechanics when solving later levels. In these cases, earlier levels appeared to function as reference points for interpreting a new task. Students compared new puzzles to previous addition, casting, or routing behavior, and used these similarities to decide what to try next T1 approx. 00:17:55–00:18:15, T3, approx. 00:07:00–00:07:40, T6, approx. 00:06:20–00:07:15. This supports the log-based observation that several practice levels were completed more quickly after an earlier introduction level.

The recording data also shows that students frequently used collaborative reasoning while progressing through the scaffolded sequence. Students read instructions aloud, proposed possible solutions, corrected each other, and tested different interpretations. When an attempt failed, students often discussed whether the problem was caused by the order of machines, the type of a box, the syntax of the code, or the direction in which a box moved T1, approx. 00:14:40–00:16:40, T2, approx. 00:35:35–00:37:05, T6, approx. 00:39:00–00:43:00. This means that repeated attempts in the log data often corresponded with active revision of hypotheses rather than only with random trial and error.

The game provided three main forms of built-in support: hints, information screens, and jumps to previous levels. These were used differently by the student groups.

Hint use was low throughout the game. The average number of hints was 0 for most levels. The highest average hint use occurred at level 19 and level 24. Level 17 also showed some hint use, but this remained low compared with the number of attempts and registered mistakes on that level.

The recording data suggest that low hint use should not automatically be interpreted as evidence that the students did not need support. Group 6 explicitly expressed reluctance to use hints early in the game T6, approx. 00:01:40–00:01:50. This indicates that hint use may partly reflect student attitude or strategy, rather than only the level of difficulty.

Information screens were used more consistently than hints. Levels 6 and 17 both had an average information-screen use of 1.00, and information screens were also used on several later levels. The recordings show that students sometimes read information screens aloud and used the information directly while reasoning about a puzzle T6, approx. 00:07:30–00:08:05, T3, approx. 00:03:20–00:04:05.

Level 7 and level 8 showed a different pattern from the other scaffolding level pairs. Level 7 introduced programmatic casting and had an average completion time of 321.02 seconds, with 9.17 attempts and 5.83 registered mistakes. Level 8 practiced the same skill area with increased complexity, but the average completion time increased to 368.41 seconds, with 10.17 attempts and 7.00 registered mistakes. Two groups, group 3 and group 5, spent a relatively long time on level 8. Group 3 spent 856.11 seconds on the level, and group 5 spent 598.67 seconds.

The conditional sequence showed a clearer scaffolding pattern in the log data. Level 17 was difficult for all groups, but level 18 was completed more quickly afterwards. The recordings show that students continued to reason with the same conditional and routing ideas during these levels T5, approx. 00:02:45–00:04:15, T2, approx. 00:35:35–00:37:05, T1, approx. 00:45:40–00:47:00. This supports the interpretation that level 18 reused the structure of level 17 in a way that allowed students to apply the same logic with less time and fewer attempts.

6.4 Additional recording markers

Some recording markers did not fit directly into the semantic wave or scaffolding categories, but still provide relevant context for interpreting the results.

6.4.1 Interface-related difficulty

The recordings show that some difficulty, especially in the early levels, came from understanding the interface rather than from the programming content itself. Students were sometimes unsure where to place machines, whether an object could be dragged, where text could be entered, and when they had to press the run button. In one group, students initially confused the hint and information buttons T1, approx. 00:01:40–00:03:50, T6, approx. 00:01:10–00:02:20. The early levels introduced both the programming metaphor and the interaction conventions of the game.

Interface issues also appeared later in the game. Some students commented on small clickable areas, difficulty selecting or dragging stations, and uncertainty about whether input had been accepted by the game T3, approx. 00:00:35–00:00:50, T2, approx. 00:06:25–00:06:35.

6.4.2 Engagement during play

Students remained engaged during the activity, including during difficult levels. Students made positive comments about the game, described some levels as enjoyable, and continued to discuss possible solutions after failed attempts T1, approx. 00:04:40–00:05:45, T5, approx. 00:00:40–00:01:35, T6, approx. 00:06:15–00:06:25. This was visible even in levels where the log data showed high completion times or many attempts.

6.4.3 Group-level variation

The final and periodic JSON logs show substantial variation between groups. Some groups completed all 24 levels, while others did not reach the final levels within the available time. The groups also differed strongly in total attempts, registered mistakes, hint use, and information screen use.

This variation was already visible in the level averages, but the group-level logs provide a clearer indication that the same level sequence did not function equally for all groups. Some groups progressed through later levels with relatively few attempts, while others spent a large amount of time on specific difficulty points, especially around levels 7, 8, 17, 19, and 24.

6.5 Differences between groups

The groups showed different progression patterns. Group 2, which included one student with a lot of prior programming experience and one with no experience, completed all 24 levels. Groups 3, 4, and 6 also completed all 24 levels. Groups 1 and 5 did not reach level 20 within the available time.

Several groups showed large individual peaks at specific levels. Group 3 spent 856.11 seconds on level 8. Group 5 spent 598.67 seconds on level 8. Group 1 spent 1061.45 seconds on level 17. Group 4 spent 747.32 seconds on level 17. Group 5 spent 760.09 seconds on level 17. Group 6 spent 890.53 seconds on level 24. These peaks occurred mainly on levels involving programmatic representation, combination of earlier concepts, or cumulative application.

The recordings also show that collaboration differed between groups. In group 2, which consisted of student 3 and student 4, the collaboration appeared asymmetrical. Student 4 more often took the lead in proposing solutions, entering or dictating code, and deciding what to try next. Student 3 still contributed by checking interpretations, correcting mistakes, and occasionally suggesting alternatives, but was less consistently involved in driving the problem-solving process T2, approx. 00:05:50–00:10:20; 00:16:40–00:17:45; 00:53:15–00:55:00. This means that the group still showed collaboration, but the cognitive and operational work was not evenly distributed throughout the session.

For groups that did not complete the game, the periodic logs remain useful. These logs show the last reached level, the level statistics up to that point, and the support use before the end of the session. They therefore allow these groups to be included in the analysis of earlier and middle levels, even though they do not provide any data for levels 20 to 24.

6.6 Final cumulative level

Level 24 functioned as the final cumulative level, where all skills and concepts came together as a regular programming assignment. Four groups reached this level. The average completion time was 488.46 seconds, with 6.75 attempts and 0.00 registered mistakes.

The absence of registered mistakes on level 24 does not mean that no errors occurred during problem solving. Mistakes were only counted when the game registered them at specific checkpoints. The recordings and observation data show that students could still discuss incorrect approaches, revise their reasoning, or test partial solutions without these actions necessarily appearing as registered mistakes in the log data. They would restart a level before a wrong box reached the end of a conveyor belt, thus not counting for registered mistakes T1, approx. 01:08:50–01:12:55, T2, approx. 00:53:15–00:55:00.

The completion times for level 24 varied between the four groups that reached it. This variation suggests that the final level functioned differently for different groups. For some groups it served as a relatively manageable cumulative task, while for others it created a new difficulty point despite the absence of registered mistakes.

7 Discussion

This study investigated how a digital educational escape room based on scaffolding and the semantic wave contributed to the process of learning fundamental programming concepts for secondary-school students. The results show that the escape room created observable moments in which students moved between concrete factory mechanics and more abstract programming representations. They also show that the scaffolded level sequence supported students in some parts of the game, although this support was not equally strong for all concepts or all groups.

7.1 Interpretation of the main findings

The log data showed clear difficulty points at levels where students had to move from concrete representations toward more programmatic forms, or where they had to combine several earlier concepts. This was especially visible in levels 7, 8, 17, 19, and 24. These levels required students to coordinate the visible factory logic with more abstract programming ideas, such as type conversion, variable use, conditional notation, and cumulative program structure.

This is relevant to the semantic-wave design of the intervention. The game presented programming concepts in an abstract form, but also embedded them in concrete mechanics such as boxes, colors, values, machines, and routing paths. The recordings show students sometimes reasoned with both levels of meaning at the same time. For example, they discussed programming terms such as *string*, *int*, *var*, *if*, and *else*, while also referring to box colors, movement direction, and factory behavior. These fragments seem to indicate that the game created opportunities for students to link concrete contexts to abstract programming concepts. This is consistent with the intended role of the semantic wave, where learning is supported by movement between context-rich examples and more general conceptual understanding.

At the same time, the most difficult levels show that these upward shifts were sometimes challenging. The transition from concrete type representations to programmatic casting was particularly demanding. Students had to understand that values had types, and also had to distinguish between variable names, stored values, value types, color labels, and conversion notation. The high time and attempt values on levels 7 and 8 suggest that this concept may have been introduced with too large a cognitive step. The semantic-wave movement was present, but the upward shift may have needed more intermediate support. The recordings help explain why level 8 did not show the same decrease as other practice levels. The casting levels required students to coordinate several related ideas at the same time: variable names, stored values, value types, color labels, and conversion notation. Because the practice level increased the complexity of these relations, the earlier level may not have provided

enough support for all groups to complete the next level more efficiently T1, approx. 00:14:40–00:16:40, T3, approx. 00:05:50–00:07:40; 00:18:45–00:20:55.

The conditional sequence showed a different pattern. Level 17 was the most difficult level in the game for most groups, but level 18 showed a strong decrease in time, attempts, and registered mistakes for almost every group. This suggests that the first programmatic conditional level created a major difficulty point, but that students were able to reuse part of that experience in the following level. In this case, the scaffolding seems to have functioned more clearly: the earlier difficult level provided a structure that students could apply again with less effort.

7.2 Scaffolding and student support

The results suggest scaffolding in the game worked best when a later level reused the same conceptual structure without adding too many new relations at once. This was visible in the decreases from level 9 to level 10, from level 11 to level 12, and especially from level 17 to level 18. In these cases, the first level introduced or combined a concept, while the following level allowed students to practice a related task more efficiently.

However, the casting sequence shows that repeated practice alone is not always enough. Level 8 was designed as practice after level 7, but it produced higher average time, attempts, and registered mistakes. This suggests that level 8 did not only practiced the concepts from level 7, but also increased the complexity before all groups had developed a stable understanding of the concept. For scaffolding to be effective, the next step in the sequence must stay within reach of the learners. In the casting levels, the gap between the introductory level and the practice level may have been too large.

The built-in support tools were used unevenly. Hint use was low to none existent, even on difficult levels. This means that hint data cannot be interpreted straightforwardly as a measure of difficulty. Some students appeared reluctant to use hints, so low hint use may reflect student strategy or attitude rather than lack of need. Students also indicated afterwards that the hints were not useful to them when they did unlock them. Information screens were used more consistently and were sometimes read aloud during problem solving. This suggests that information screens may have functioned as a more acceptable form of support than hints. For future versions of the game, hints might be reframed less as help after failure and more as optional checks or prompts during normal problem solving to promote usage. The current hint system also used static hints, serving as reminders to check certain parts of the level and read carefully, while students might be better served by hints that adapt to what is going wrong. Hints given by the teacher were appreciated by the students.

Collaboration also functioned as a form of support. The recordings show that students read instructions aloud, proposed possible solutions, corrected each other, and revised their solutions after failed attempts. A high number of attempts did not necessarily mean that students were guessing randomly. In many cases, repeated attempts were accompanied by discussion and revision. The escape room format therefore supported collaborative reasoning, which is one of the intended strengths of educational escape rooms [15].

7.3 Engagement and game-based learning

The recordings suggest students generally remained engaged, including during difficult levels. Students continued to discuss possible solutions, reacted to failed attempts, and made positive comments about the activity. One of the reasons for using game-based learning is that it can support motivation and sustained participation [11], [13]. The data from this study do not measure motivation formally, but the recordings provide process-level indications that students continued to participate actively even when the tasks became difficult. One of the groups that got stuck on level 17 did show signs of frustration and fatigue near the end of their attempts, but they kept trying none the less. This might indicate that the escape room continues to engage students, but the students were compensated with a small monetary gift for just participating, which could also have affected their motivation.

The game context also appears to have made abstract programming concepts more discussable. Instead of only reasoning about code, students could refer to visible events in the factory. This gave them a shared representation for talking about variables, types, casting, arithmetic, and conditionals. This is especially relevant for novice programmers, because introductory programming difficulties often involve weak mental models of what code does during execution [3], [4]. The factory metaphor did not remove these difficulties, but it gave students a concrete environment in which they could test and revise their understanding.

7.4 Limitations

Several limitations should be considered when interpreting these findings. First, the study used a small sample of 13 students in six groups from a single school. The findings therefore cannot be generalized to all secondary-school students. The study should be understood as a descriptive case study of a designed intervention. Future research into the topic should look into larger and more diverse student populations, making it possible to do a statistical analysis on the data of each level. We could then see with more certainty which levels prove to be too large a jump in the upward shift of the semantic wave and the scaffolding.

Second, the study focused on process markers rather than direct learning gain. There was no formal pre-test and post-test design. Although the final level required students to apply several programming concepts, completion of this level cannot by itself demonstrate knowledge gain, because prior knowledge was not measured systematically. The final level can therefore be interpreted as evidence of successful task performance within the game, not as proof that students learned the concepts during the intervention. With a large enough group, AB testing could be done to determine the difference in learning outcomes of the educational escape room compared to a class that received a regular instruction on programming. Taking this even further, the learning outcomes could be measured by having different versions of the escape room, with each version having a focus on certain aspects of the pedagogical approaches. Things like granting more or fewer hints in levels, have more but smaller levels, or giving more adaptive feedback when making mistakes, which can all be used to better map how the semantic wave and scaffolding contribute to the learning process of a student.

Third, not all groups completed all levels. Groups 1 and 5 did not reach levels 20 to 24, so the averages for the later levels are based only on the groups that progressed the farthest. This creates a selection effect. Lower times or fewer mistakes in later levels may partly reflect improved understanding, but they may also reflect the fact that only stronger or faster groups reached those levels.

Fourth, the registered mistake count was limited by the way the game logged errors. Mistakes were only counted at specific checkpoints. Students could still make incorrect assumptions, discuss wrong solutions, or restart before an error was registered. Completion time, attempts, recordings, and observation notes are therefore necessary to interpret the mistake data.

Fifth, some of the recorded difficulty was related to interface use rather than programming content. Students sometimes struggled with dragging, clicking, entering text, or understanding buttons. These issues may have increased completion time independently of conceptual difficulty, though most of these struggles appeared in the earlier levels. Future versions of the intervention should further separate interface learning from programming learning, especially in the early levels.

Finally, the recordings were automatically transcribed and one group's recordings were corrupted. The transcript references are therefore useful as approximate markers, but they should not be treated as exact quotations without manual correction. The transcriptions were corrected where relevant, but not everywhere.

7.5 Possible misalignment between game design elements

One possible explanation for the difficulty observed in some levels is a partial misalignment between the four design elements described by Van der Linden et al 2019 [14]: game goal, learning goal, game mechanics, and pedagogical approach. The game was designed so that completing each level required

students to apply the intended programming concept. In this sense, the game goal and learning goal were intended to be aligned. However, the results suggest that this alignment was not equally strong in all levels.

In several levels, especially levels 7, 8, and 17, the game mechanics may have introduced additional demands that were not fully part of the learning goal. In the casting levels, students had to coordinate the factory representation of colored boxes with the programming concepts of variable names, values, and types. The recording data shows that students sometimes confused these elements. This suggests that the concrete representation did not always make the abstract concept easier to interpret. Instead, the representation sometimes became an additional object of reasoning.

A similar issue appears in the conditional levels. These levels required students to understand conditional logic, but also to connect that logic to the spatial routing of boxes through the factory. Students therefore had to reason about both the code structure and the direction of movement in the level. For some groups, this may have split attention between the programming concept and the spatial puzzle mechanic. This could explain why level 17 showed the highest average completion time, number of attempts, and number of registered mistakes.

This partial misalignment may help explain why some levels did not show the expected pattern of decreasing time, attempts, and mistakes after earlier scaffolded levels. Level 8, for example, practiced the same skill area as level 7 but had a higher average completion time and more attempts. Rather than indicating that scaffolding was absent, this may indicate that the increase in representational and mechanical complexity outweighed the support provided by the previous level.

7.6 Implications for design

The results suggest several improvements for future versions of the escape room. The casting sequence should be redesigned with smaller intermediate steps. Students may need more explicit separation between variable name, value, type, and conversion syntax before being asked to combine these ideas in a complex task. Additional levels could first ask students to identify a variable's name, then its value, then its type, before using casting notation. Additional intermediate levels should not increase the total duration of the escape room, because most groups already needed close to the full available session time and the design also requires time for introduction and debriefing.

The conditional sequence should also be reviewed, but in a different way. Level 17 was very difficult, but the decrease on level 18 suggests that the sequence did support later application. Future versions could keep the same overall structure while adding clearer visual feedback that links the truth value of a condition to the route taken by the box.

The support system could also be improved. Since hints were used rarely, hints may need to be integrated more naturally into the gameplay. For example, the game could provide optional reflection prompts after repeated failed attempts, or it could make hints feel like diagnostic questions rather than direct assistance. Information screens appeared more acceptable to students, so reframing hints like additional information could therefore be explored in future iterations.

Finally, the debriefing phase should be included more explicitly in future research. The present results focus on gameplay data, but the debriefing is theoretically important because it can help complete the movement from concrete game experience to abstract programming concepts. Future studies could record the debriefing or include a short post-task reflection to examine how students articulate the programming concepts after playing.

8 Conclusion

This study explored how a digital educational escape room based on scaffolding and the semantic wave contributed to the process of learning fundamental programming concepts for secondary-school students. The findings show that the escape room created meaningful process markers for both mechanisms.

For the semantic wave, students were observed moving between concrete factory representations and abstract programming concepts. They used programming vocabulary while reasoning about boxes, colors, machines, routes, and values. The most difficult levels occurred when the game required an upward shift toward more programmatic representations or when students had to combine several earlier concepts. This suggests that the escape room succeeded in making abstract concepts visible and discussable, but also that these upward shifts require careful pacing.

For scaffolding, several level sequences showed lower completion times, fewer attempts, and fewer registered mistakes after an earlier introductory level. This was especially visible in the conditional sequence from level 17 to level 18. However, the casting sequence showed that scaffolding was not equally effective throughout the game. Level 8 became more difficult than level 7, which suggests that the step from introduction to practice was too large for some groups.

The study therefore indicates that a digital educational escape room can support the learning process of introductory programming by giving students concrete representations to reason with, opportunities to collaborate, and a structured sequence of increasingly abstract tasks. However, the results should be interpreted as process evidence rather than proof of learning gain. The escape room showed how students interacted with programming concepts during gameplay, where they struggled, and where the design supported or failed to support progression.

Future versions should refine the casting levels, reduce interface-related difficulty, integrate hints more naturally, and include the debriefing phase as part of the research data. A follow-up study with a larger sample and a pre-test/post-test design could examine whether the observed process markers also correspond to measurable learning gains.

References

[1] SLO, *Digitale geletterdheid*, `https://www.slo.nl/thema/meer/basisvaardigheden/digitale-geletterdheid/`, Geraadpleegd op 25 mei 2025, Oct. 2024. [Online]. Available: `https://www.slo.nl/thema/meer/basisvaardigheden/digitale-geletterdheid/`.

[2] SLO, *Domein d: Programmeren*, `https://www.slo.nl/handreikingen/havo-vwo/handreiking-se-info-hv/examenprogramma/domein-programmeren/`, Geraadpleegd op 25 mei 2025, 2020. [Online]. Available: `https://www.slo.nl/handreikingen/havo-vwo/handreiking-se-info-hv/examenprogramma/domein-programmeren/`.

[3] Y. Qian and J. Lehman, "Students' misconceptions and other difficulties in introductory programming: A literature review," *ACM Transactions on Computing Education (TOCE)*, vol. 18, pp. 1–24, 1 2017, ISSN: 1946-6226.

[4] A. Swidan, F. Hermans, and M. Smit, "Programming misconceptions for school students," in *Proceedings of the 2018 ACM Conference on International Computing Education Research*, 2018, pp. 151–159.

[5] C. Y. Tsai, "Improving students' understanding of basic programming concepts through visual programming language: The role of self-efficacy," *Computers in Human Behavior*, vol. 95, pp. 224–232, Jun. 2019, ISSN: 0747-5632. DOI: `10.1016/J.CHB.2018.11.038`.

[6] H. Hosseini, M. Hartt, and M. Mostafapour, "Learning is child's play: Game-based learning in computer science education," *ACM Transactions on Computing Education (TOCE)*, vol. 19, pp. 1–18, 3 2019, ISSN: 1946-6226.

[7] M. Videnovik, T. Vold, L. Kiønig, A. M. Bogdanova, and V. Trajkovik, "Game-based learning in computer science education: A scoping literature review," *International Journal of STEM Education*, vol. 10, p. 54, 1 2023, ISSN: 2196-7822.

[8] C. S. Cheah, "Factors contributing to the difficulties in teaching and learning of computer programming: A literature review," *Contemporary Educational Technology*, vol. 12, ep272, 2 2020, ISSN: 1309-517X.

[9] K. Becker, "What's the difference between gamification, serious games, educational games, and game-based learning," *Academia Letters*, vol. 209, no. 2, pp. 1–4, 2021.

[10] S. Deterding, D. Dixon, R. Khaled, and L. Nacke, "From game design elements to gamefulness: Defining" gamification"," in *Proceedings of the 15th international academic MindTrek conference: Envisioning future media environments*, 2011, pp. 9–15.

[11] M. Prensky, "Digital game-based learning," *Computers in entertainment (CIE)*, vol. 1, p. 21, 1 2003, ISSN: 1544-3574.

[12] S. Erhel and E. Jamet, "Digital game-based learning: Impact of instructions and feedback on motivation and learning effectiveness," *Computers & education*, vol. 67, pp. 156–167, 2013.

[13] E. L. Deci and R. M. Ryan, "The" what" and" why" of goal pursuits: Human needs and the self-determination of behavior," *Psychological inquiry*, vol. 11, pp. 227–268, 4 2000, ISSN: 1047-840X.

[14] A. van der Linden, W. R. van Joolingen, and R. F. G. Meulenbroeks, "Designing an intrinsically integrated educational game on newtonian mechanics," in *Games and Learning Alliance: 7th International Conference, GALA 2018, Palermo, Italy, December 5–7, 2018, Proceedings 7*, Springer, 2019, pp. 123–133, ISBN: 303011547X.

[15] A. Veldkamp, L. van de Grint, M. C. P. Knippels, and W. R. van Joolingen, "Escape education: A systematic review on escape rooms in education," *Educational Research Review*, vol. 31, p. 100 364, Nov. 2020, ISSN: 1747-938X. DOI: 10.1016/J.EDUREV.2020.100364.

[16] D. Wood, J. S. Bruner, and G. Ross, "The role of tutoring in problem solving," *Journal of child psychology and psychiatry*, vol. 17, pp. 89–100, 2 1976, ISSN: 0021-9630.

[17] W. Min, B. Mott, and J. Lester, "Adaptive scaffolding in an intelligent game-based learning environment for computer science," in *Proceedings of the workshop on AI-supported education for computer science (AIEDCS) at the 12th international conference on intelligent tutoring systems*, 2014, pp. 41–50.

[18] X. Piotrowska and T. Alekseeva, "Scaffolding for clil in computer science courses: Data-driven learning approach," in *Ceur Workshop Proceedings. Proceedings of the XV International Conference (NESinMIS-2020)*, 2020, pp. 87–99.

[19] K. Maton, "Making semantic waves: A key to cumulative knowledge-building," *Linguistics and Education*, vol. 24, pp. 8–22, 1 Apr. 2013, ISSN: 0898-5898. DOI: 10.1016/J.LINGED.2012.11.005.

[20] P. Curzon, J. Waite, K. Maton, and J. Donohue, "Using semantic waves to analyse the effectiveness of unplugged computing activities," in *Proceedings of the 15th Workshop on Primary and Secondary Computing Education*, 2020, pp. 1–10.

9 Appendix: Additional Tables & Figures

A Transcript Data

The video and audio recordings of the student groups were automatically transcribed using a digital transcription tool. The transcripts were then manually checked and corrected in segments that were relevant for the analysis. The full transcripts are not included in this thesis because they contain research data from student participants. The transcripts have instead been uploaded to the university data management system, where they are stored securely and can be accessed according to the university's data management and privacy procedures, and can be found at: https://science.yoda.uu.nl/research/browse?dir=%2Fresearch-veltk101-msc-gmt-nieuwkoop.

In the results chapter, references to transcript fragments are given using the transcript number and an approximate timestamp. For example, a reference such as [T1, approx. 00:14:40–00:16:40] refers

to the fragment in transcript T1 around that point in the recording. These references are used as markers for the qualitative analysis. They should not be treated as exact quotations unless a fragment is quoted explicitly in the text.

Table 6: Overview of transcript files

Reference	Group	File Name
T1	Group 1	Student1Student2.txt
T2	Group 2	Student3Student4.txt
T3	Group 3	Student5Student6.txt
T5	Group 5	Student9Student10.txt
T6	Group 6	Student11Student12Student13.txt

B Log Data Graphs

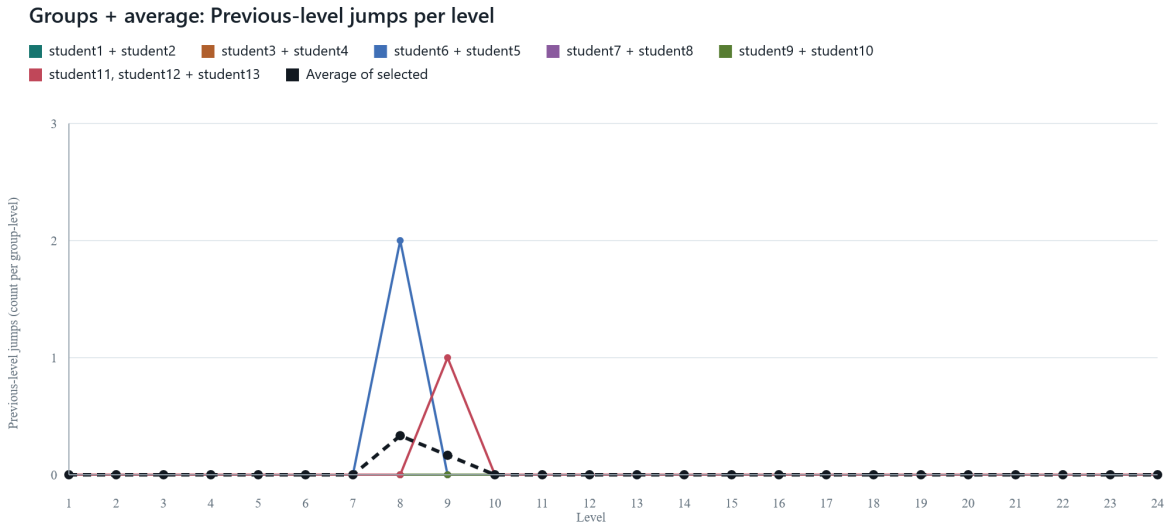

Figure 4: Previous-level jumps per level for each group and the average of the selected groups.

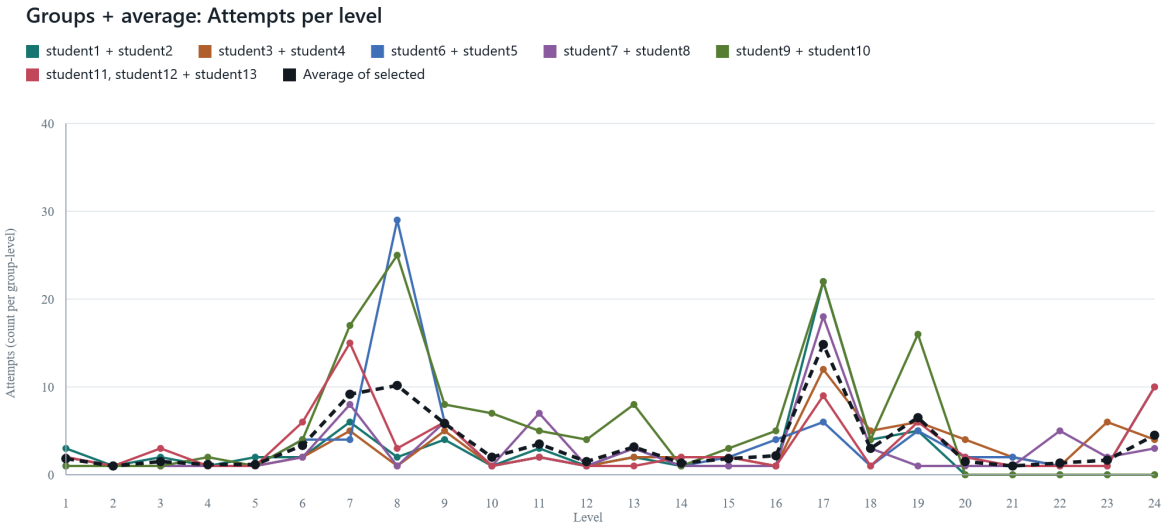

Figure 5: Attempts per level for each group and the average of the selected groups.

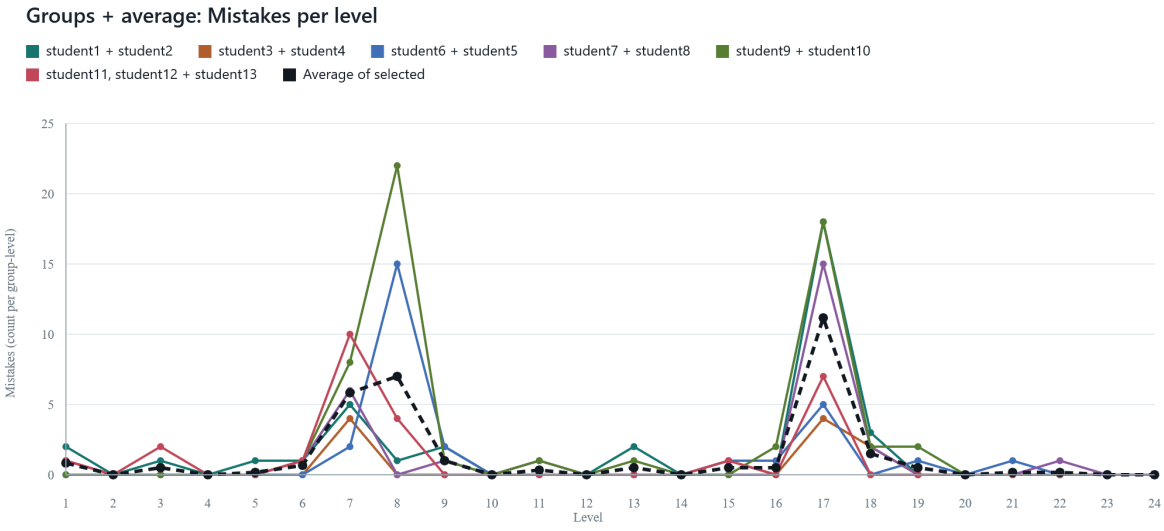

Figure 6: Registered mistakes per level for each group and the average of the selected groups.

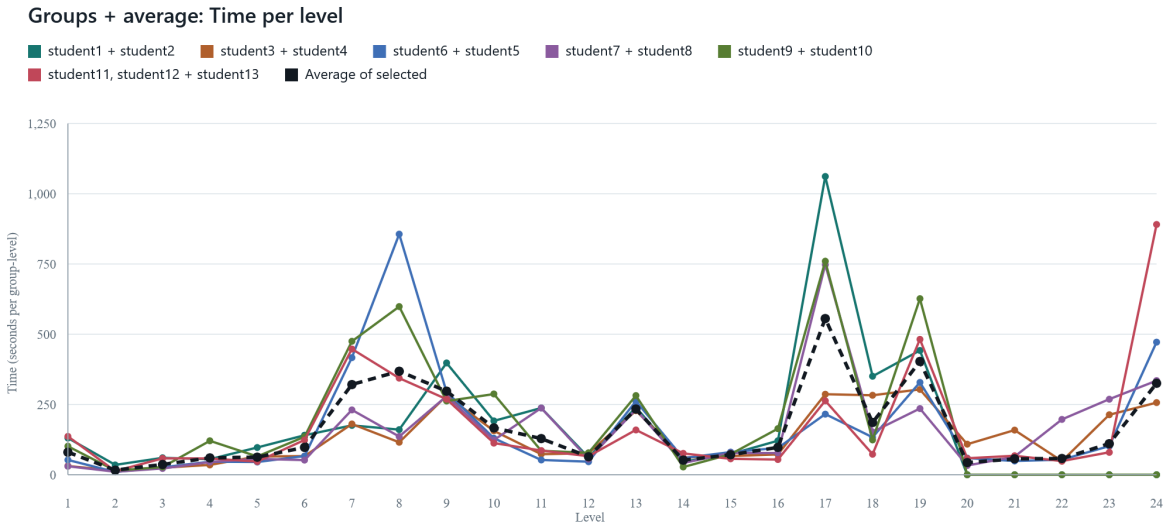

Figure 7: Completion time per level for each group and the average of the selected groups.

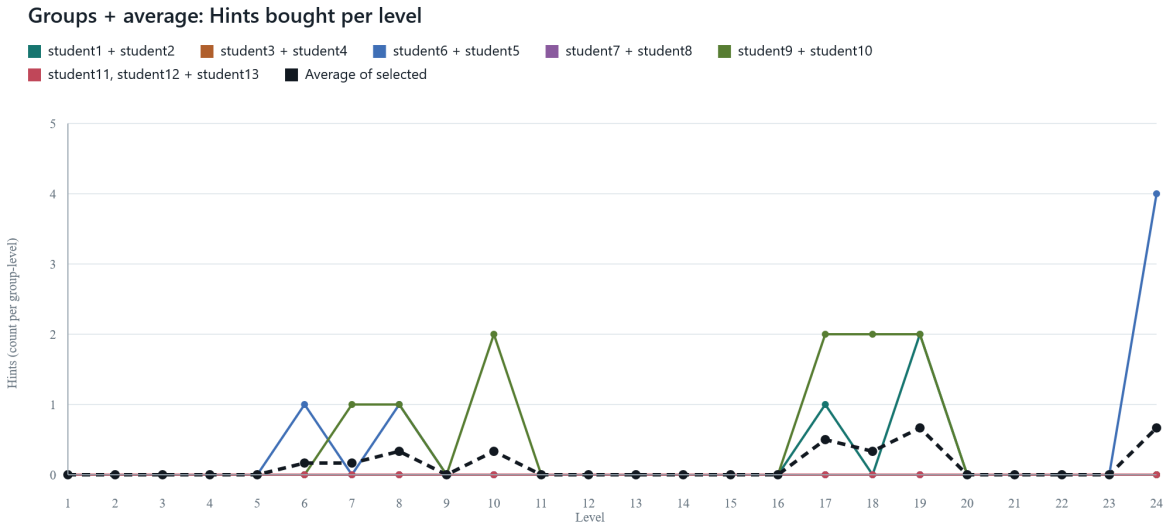

Figure 8: Hints bought per level for each group and the average of the selected groups.

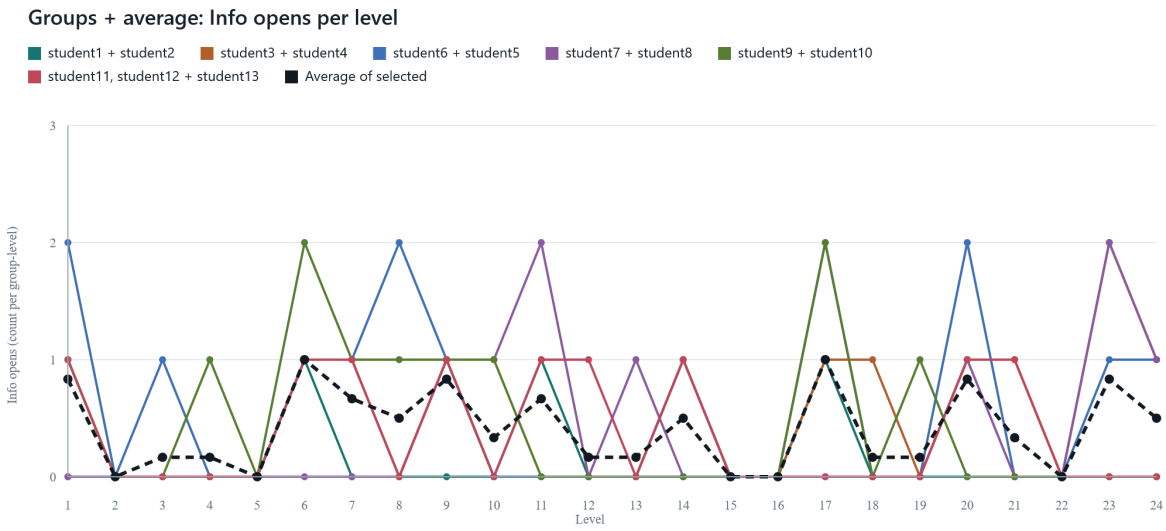

Figure 9: Information-screen opens per level for each group and the average of the selected groups.

C Log Data Tables

Table 7: Average statistics of all groups

level	pairs	avg_time_sec	avg_attempts	avg_mistakes	avg_hints	avg_info	avg_jumps
1	6	80.4	1.83	0.83	0	0.83	0
2	6	16.41	1	0	0	0	0
3	6	36.59	1.5	0.5	0	0.17	0
4	6	60.12	1.17	0	0	0.17	0
5	6	62.64	1.17	0.17	0	0	0
6	6	97.37	3.33	0.67	0.17	1	0
7	6	321.02	9.17	5.83	0.17	0.67	0
8	6	368.41	10.17	7	0.33	0.5	0.33
9	6	297.44	5.83	1	0	0.83	0.17
10	6	166.83	2	0	0.33	0.33	0
11	6	128.69	3.5	0.33	0	0.67	0
12	6	63.97	1.5	0	0	0.17	0
13	6	233.75	3.17	0.5	0	0.17	0
14	6	51.89	1.33	0	0	0.5	0
15	6	71.26	1.83	0.5	0	0	0
16	6	97.34	2.17	0.5	0	0	0
17	6	555.73	14.83	11.17	0.5	1	0
18	6	186.01	3	1.5	0.33	0.17	0
19	6	403.13	6.5	0.5	0.67	0.17	0
20	4	64.75	2.25	0	0	1.25	0
21	4	84.92	1.5	0.25	0	0.5	0
22	4	86.84	2	0.25	0	0	0
23	4	165.96	2.5	0	0	1.25	0
24	4	488.46	6.75	0	1	0.75	0

Table 8: Statistics of group 1

level	time_seconds	attempts	mistakes	hints	info	jumps
1	130.69	3	2	0	1	0
2	35.1	1	0	0	0	0
3	60.19	2	1	0	0	0
4	55.37	1	0	0	0	0
5	96.98	2	1	0	0	0
6	140.54	2	1	0	1	0
7	175.62	6	5	0	0	0
8	160.57	2	1	0	0	0
9	397.82	4	2	0	0	0
10	191.6	1	0	0	0	0
11	237.49	3	0	0	1	0
12	61.27	1	0	0	0	0
13	239.76	2	2	0	0	0
14	46.46	1	0	0	0	0
15	73.31	1	0	0	0	0
16	121.07	1	0	0	0	0
17	1061.45	22	18	1	1	0
18	350.37	4	3	0	0	0
19	442.32	5	0	2	0	0

Table 9: Statistics of group 2

level	time_seconds	attempts	mistakes	hints	info	jumps
1	30.99	2	1	0	0	0
2	15.04	1	0	0	0	0
3	25.07	1	0	0	0	0
4	35.1	1	0	0	0	0
5	65.19	1	0	0	0	0
6	66.33	2	0	0	1	0
7	180.59	5	4	0	1	0
8	115.59	1	0	0	0	0
9	281.29	5	0	0	1	0
10	156.17	1	0	0	0	0
11	73.27	2	0	0	0	0
12	76.44	1	0	0	0	0
13	227.93	2	0	0	0	0
14	60.99	2	0	0	1	0
15	66.46	2	1	0	0	0
16	72.45	1	0	0	0	0
17	286.57	12	4	0	1	0
18	282.87	5	2	0	1	0
19	303.68	6	0	0	0	0
20	108.62	4	0	0	1	0
21	158.83	2	0	0	1	0
22	48.63	1	0	0	0	0
23	213.41	6	0	0	2	0
24	256.71	4	0	0	1	0

Table 10: Statistics of group 3

level	time_seconds	attempts	mistakes	hints	info	jumps
1	52.55	1	0	0	2	0
2	9.76	1	0	0	0	0
3	26.34	1	0	0	1	0
4	46.4	1	0	0	0	0
5	45.19	1	0	0	0	0
6	66.35	4	0	1	1	0
7	416.76	4	2	0	1	0
8	856.11	29	15	1	2	2
9	294.64	6	2	0	1	0
10	129.81	1	0	0	0	0
11	52.48	2	0	0	0	0
12	46.37	1	0	0	0	0
13	261.69	3	0	0	0	0
14	58.85	1	0	0	1	0
15	80.54	2	1	0	0	0
16	95.18	4	1	0	0	0
17	215.74	6	5	0	0	0
18	133.22	1	0	0	0	0
19	328.58	5	1	0	0	0
20	59.12	2	0	0	2	0
21	48.88	2	1	0	0	0
22	53.51	1	0	0	0	0
23	101.8	1	0	0	1	0
24	471.96	10	0	4	1	0

Table 11: Statistics of group 4

level	time_seconds	attempts	mistakes	hints	info	jumps
1	30.09	2	1	0	0	0
2	10.73	1	0	0	0	0
3	22.21	1	0	0	0	0
4	45.15	1	0	0	0	0
5	56.57	1	0	0	0	0
6	52.07	2	1	0	0	0
7	230.8	8	6	0	0	0
8	135.9	1	0	0	0	0
9	279.64	6	1	0	1	0
10	123.41	1	0	0	1	0
11	237.43	7	1	0	2	0
12	55.61	1	0	0	0	0
13	232.14	3	0	0	1	0
14	41.62	1	0	0	0	0
15	77.38	1	0	0	0	0
16	77.16	1	0	0	0	0
17	747.32	18	15	0	2	0
18	152.4	3	2	0	0	0
19	235.56	1	0	0	0	0
20	32.7	1	0	0	1	0
21	64.44	1	0	0	0	0
22	196.76	5	1	0	0	0
23	268.75	2	0	0	2	0
24	334.65	3	0	0	1	0

Table 12: Statistics of group 5

level	time_seconds	attempts	mistakes	hints	info	jumps
1	101.53	1	0	0	1	0
2	15.04	1	0	0	0	0
3	27.6	1	0	0	0	0
4	120.84	2	0	0	1	0
5	65.27	1	0	0	0	0
6	135.38	4	1	0	2	0
7	474.88	17	8	1	1	0
8	598.67	25	22	1	1	0
9	262.36	8	1	0	1	0
10	287.28	7	0	2	1	0
11	85.29	5	1	0	0	0
12	78.32	4	0	0	0	0
13	281.54	8	1	0	0	0
14	27.54	1	0	0	0	0
15	73.27	3	0	0	0	0
16	164.29	5	2	0	0	0
17	760.09	22	18	2	2	0
18	124.2	4	2	2	0	0
19	626.6	16	2	2	1	0

Table 13: Statistics of group 6

level	time_seconds	attempts	mistakes	hints	info	jumps
1	136.53	2	1	0	1	0
2	12.75	1	0	0	0	0
3	58.15	3	2	0	0	0
4	57.89	1	0	0	0	0
5	46.65	1	0	0	0	0
6	123.58	6	1	0	1	0
7	447.44	15	10	0	1	0
8	343.63	3	4	0	0	0
9	268.87	6	0	0	1	1
10	112.7	1	0	0	0	0
11	86.15	2	0	0	1	0
12	65.81	1	0	0	1	0
13	159.47	1	0	0	0	0
14	75.87	2	0	0	1	0
15	56.59	2	1	0	0	0
16	53.91	1	0	0	0	0
17	263.2	9	7	0	0	0
18	73	1	0	0	0	0
19	482.03	6	0	0	0	0
20	58.56	2	0	0	1	0
21	67.51	1	0	0	1	0
22	48.47	1	0	0	0	0
23	79.89	1	0	0	0	0
24	890.53	10	0	0	0	0

D Game Screenshots

Figure 10: Screenshots from level 17 of The Factory

Figure 11: Screenshots showing the conditional interface from level 17 of The Factory

Figure 12: Screenshots showing the programming interface from level 24 of The Factory