Railway Quest: A Gamified Teaching Platform for Concurrent and Parallel Programming

Masiar Babazadeh, Diego Frei, Massimo Bortolamei, Mirko Gelsomini, Adriano Cicco, Loris Bruno and Tiziano Leidi
University of Applied Sciences and Arts of Southern Switzerland (SUPSI), Department of Innovative Technologies (DTI), Lugano, Switzerland

masiar.babazadeh@supsi.ch
diego.frei@supsi.ch
massimo.bortolamei@supsi.ch
mirko.gelsomini@supsi.ch
adriano.cicco@supsi.ch
loris.bruno@supsi.ch
tiziano.leidi@supsi.ch

Abstract: Teaching concurrent and parallel programming presents substantial challenges, primarily due to its conceptual complexity and the required paradigm shift from sequential programming. Railway Quest is a gamified digital platform designed to facilitate students' learning of multithreaded programming concepts through an interactive approach. The platform translates abstract programming challenges into train locomotion scenarios, wherein trains represent threads that must move along tracks while avoiding crashes, resource access conflicts, and indefinite delays. By leveraging game-based learning and gamification principles, Railway Quest enhances students' motivation and engagement through a visual and interactive representation of multithreaded executions and concurrency issues. The platform allows students to develop and execute code-based solutions, immediately observing their impact in real-time simulations presented through a low-poly isometric interface. Such approach significantly helps in conceptualizing threading constructs and synchronization mechanisms. Railway Quest supports both integrated and remote learning environments, making it an effective educational tool, especially for distance learning. The platform has been deployed in a university course in Spring 2025, with the aim to bridge the gap between theoretical knowledge and practical problem-solving in concurrent and parallel programming. Preliminary data from 45 students suggests a positive impact on students' understanding of key concurrency problems such as race conditions, deadlocks, and starvation, as well as their proficiency in utilizing synchronization tools like mutexes and conditional variables.

Keywords: Concurrent programming, Parallel programming, Gamification, Game-Based learning, Computer science education, Interactive learning

1. Introduction

Concurrent and parallel programming (CPP) is a fundamental topic in modern computer science curricula, yet it remains notoriously challenging for students to learn (Ben-Ari**,** 2001**;** El-Nashar & Nakamura, 2013**;** Zhu et al, 2020). The underlying dynamics, such as thread and synchronization behaviours, race conditions, and deadlocks, require a significant paradigm shift from sequential programming, involving a high level of abstraction capability. (Ben-Ari & Kolikant, 1999; Hughes et al, 2005). Traditional teaching methods often struggle to make these dynamic, complex concepts tangible and understandable.

Game-Based Learning (GBL) and gamification have shown promise in increasing student engagement and facilitating the understanding of complex topics across various domains, including computer science (Ibanez et al., 2014; Oktaviati & Jaharadak, 2018). However, applying GBL effectively to the abstract and often counter-intuitive paradigm of concurrent and parallel programming, particularly in terms of visualization, presents unique challenges and opportunities.

This paper introduces Railway Quest, an innovative gamified simulation platform developed at the University of Applied Sciences and Arts of Southern Switzerland (SUPSI). Designed for flexibility, the platform supports various learning contexts including integrated classroom settings and distance learning scenarios. Railway Quest aims to facilitate the learning of CPP concepts by translating abstract programming problems into engaging, visual missions involving train locomotion. Students write programs to control trains (representing threads) advancing through tracks and needing to manage shared resources (crossings, train stations) and avoid concurrency issues like collision risks (data races) and indefinite delays (deadlocks).

The remainder of this paper is structured as follows. Section 2 introduces the related work in game-based learning and CPP. Section 3 presents the design rationale and pedagogical approach, as well as the

development process of Railway Quest. Furthermore, Section 4 presents preliminary findings from its initial deployment in a second-year undergraduate Computer Science course in Spring 2025, evaluating its perceived effectiveness and reception by students. Section 5 concludes the paper by presenting a discussion on the findings, and directions for future work.

2. Background

To position Railway Quest, we conducted a targeted review of existing educational games and platforms for programming. The selection focused on tools mentioned in the academic literature on computer science education and popular educational game repositories. Each tool was analyzed based on its approach to teaching parallelism, the inclusion of core concurrency concepts (e.g., non-determinism, synchronization primitives), the use of actual programming languages, and its visualization strategies for abstract concepts. This analysis aimed to identify gaps that Railway Quest could address.

Computer science education has long explored game-based learning approaches to teach (sequential) programming and algorithmic thinking, as reported in the literature (Akimoto & Cheng, 2003; Cliburn, 2006; Barnes et al, 2008; Papastergiou, 2009; Ibrahim et al, 2011; Harteveld, Smith & Carmichael 2014;, Abernethy, 2018; Zhu et al, 2020), and in off-the-shelf learning applications, with games such as *Space Chem*, *Human Resource Machine*, or *The Farmer Was Replaced*. Even though CPP has become increasingly essential in modern software development, driven by the proliferation of multicore processors, the complexities of the subject remain largely unaddressed by educational games, thereby continuing to present substantial learning challenges for students and novice developers. Unlike traditional sequential programming, adding concurrent thread execution in a software application requires a fundamental shift in how problems are approached, and solutions are structured, often involving complex concepts such as shared memory access, critical section execution, synchronization, deadlocks, and starvation.

Some games, like *Space Chem*, introduce parallel execution but only in a deterministic way, failing to capture the non-deterministic nature of a concurrent program execution. Among others, *Parapple* (Buzek & Kruliš, 2018) presents the non-deterministic aspects of such programs, yet it does not cover synchronization constructs (such as mutexes, conditional variable or barriers).

Parallel (Zhu et al, 2019; Zhu et al, 2020) is one of the few games able to introduce some of the basic CPP concepts by retaining its non-deterministic nature. Players are required to design a strategy to coordinate multiple arrows (representing threads) as they collect packages and deliver them to designated points, while correctly managing access to critical sections along their ways. While the game integrates some elements of concurrent and parallel programming, such as threads, shared resources and semaphores, it falls short in establishing a concrete connection to actual programming paradigms and practices (by using real code to program the game). Additionally, it lacks a realistic contextual scenario that helps students visualize multi-threaded executions using a metaphor-based simulation.

This paper presents the approach adopted in Railway Quest to introduce and teach CPP concepts directly with a commodity programming language (such as Java), by means of the visual metaphor of trains, significantly helping in conceptualizing threading constructs and synchronization mechanisms, encouraging situated learning.

3. Game Design and Development

This section presents the rationale behind the game design, the pedagogical considerations underlying the selected approaches, how CPP elements are tied to in-game elements, and the architecture of the platform.

3.1 Conceptual Game Design and Pedagogical Approach

The development of Railway Quest was guided by a Design-Based Research (DBR) methodology. This approach was chosen to systematically address the practical challenges of teaching concurrent and parallel programming while contributing to a deeper understanding of how gamified simulations can support learning in this complex domain. Our DBR process involved iterative cycles of design, implementation, analysis, and refinement, conducted in the authentic context of the second-year undergraduate Computer Science course “Parallel and Concurrent Programming”, which teaches CPP fundamentals in the Java programming language. The interdisciplinary collaboration between the Department of Innovative Technologies (DTI) and the Department of Education and Learning / University of Teacher Education (DFA/ASP) at the University of Applied Sciences and Arts of Southern Switzerland (SUPSI) was central to this methodology, ensuring that pedagogical

principles, gamification strategies, and technical development were tightly integrated. Railway Quest is primarily designed for individual problem-solving, allowing students to progress at their own pace. While direct in-platform collaboration or competition features are not yet implemented, the laboratory setting encourages peer discussion and collaborative problem-solving around the challenges presented by the platform, facilitated by instructors.

The central metaphor translates CPP concepts into a railway scenario: threads become trains, each following a path defined by the track layout. Shared resources critical to concurrency are represented by specific track segments, such as single-track crossings, stations, or loading docks, which only one train can access at a time. Synchronization primitives (locks, conditional variables, synchronizers) are the tools students must use in their Java code to ensure safe traversal through these critical sections and avoid resource contentions. Concurrency issues manifest visually. For example, race conditions might appear as trains attempting to occupy the same segment simultaneously (risks of collision), while deadlocks manifests as trains waiting indefinitely at crossings or when attempting to access shared resources.

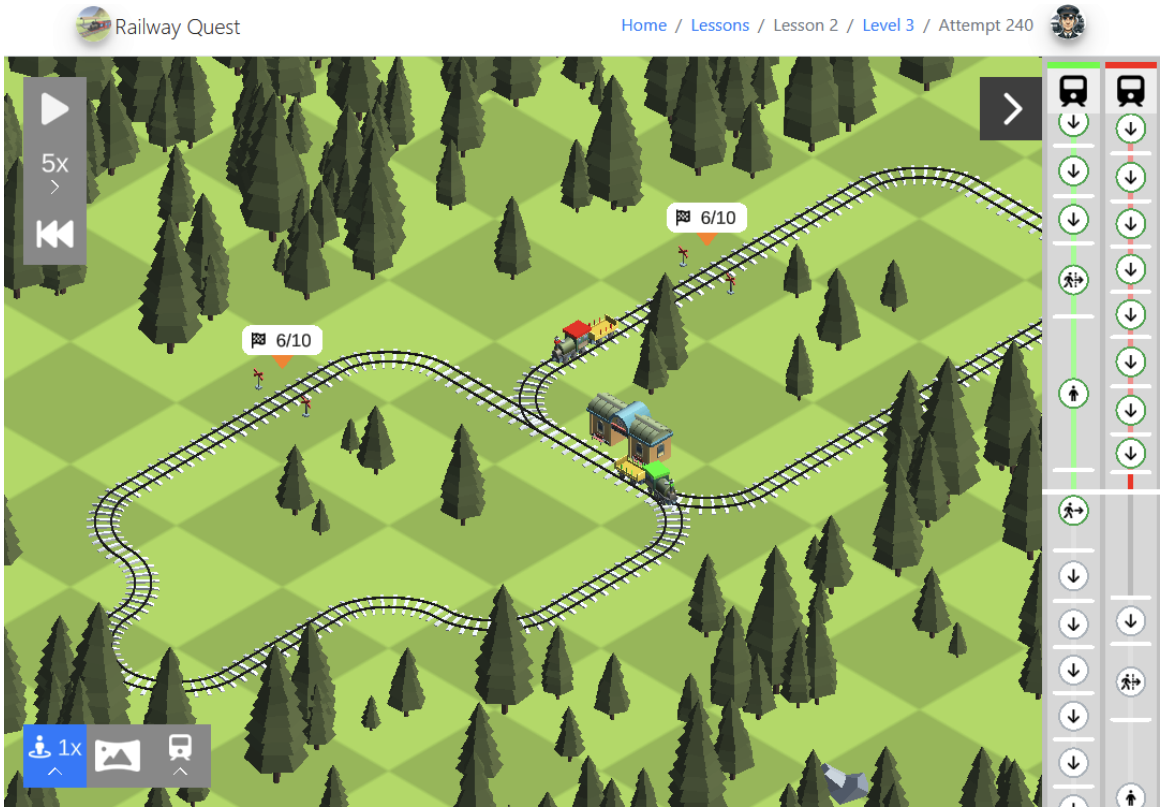

Figure 1: Trains traversing a critical section (the station)

Figure 1 shows the main interface of Railway Quest. Two train tracks share a critical section: the train station. The red train on top is correctly waiting until the station is freed by the green train, at the bottom. Students can modify the execution speed of the simulation by clicking the buttons at the top left corner of the screen, or modify the track viewpoint with the bottom left buttons. The timeline view on the right shows the commands' sequence for easier comprehension and code debugging. This view shifts top to bottom at real time, continuously showing the execution of the commands of each train. The colored top half shows the commands to be executed, the bottom greyed-out part shows the commands that still need to be performed.

The core gameplay loop was designed to involve the students visually analyzing a specific railway challenge presented on the platform (e.g., ensure two trains cross safely as shown in Figure 1, manage resource loading/unloading between trains as shown in Figure 2), writing Java code locally to control the trains' behavior using CPP constructs, and running their code to see the outcome. We designed the platform with a constructivist approach in mind, allowing the students observe the resulting train movements and interactions in the real-time visual simulation. Students can debug and refine their code based on the visual feedback until the challenge objective is met without errors, supporting formative assessment and allowing students to learn interactively. Challenges are hand-made and designed with increasing complexity, starting from basic thread execution to complex multi-thread coordination patterns. This approach was implemented to ensure that

students first acquire foundational knowledge before progressing, thereby supporting the maintenance of a continuous state of flow (Nakamura & Csikszentmihalyi, 2002) during the process.

Each challenge is explicitly designed to target specific CPP learning objectives, such as understanding mutual exclusion, implementing atomic operations, preventing deadlocks, or applying specific thread coordination patterns. The goal of each level is not to merely enable train movement, but to ensure that their *behavior adheres* to the principles and disciplines of concurrent execution. This fosters metacognitive skills, helping students understand *how* and *why* concurrency bugs happen.

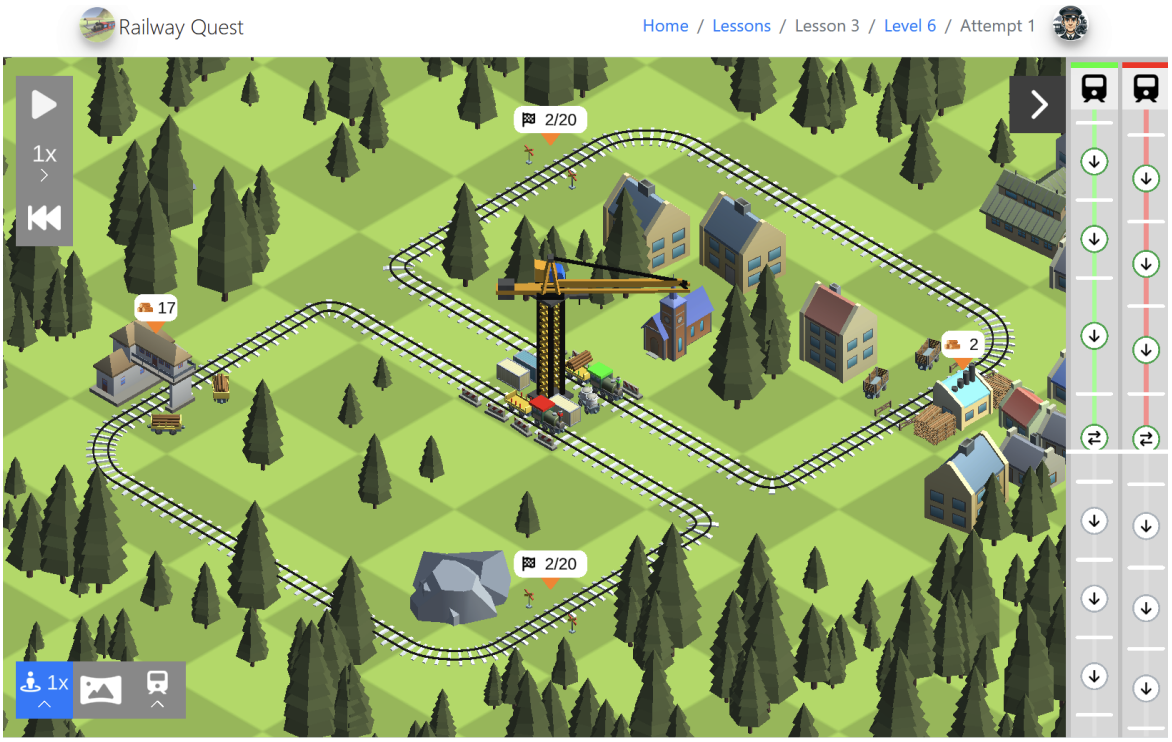

Figure 2: Trains loading and unloading resources, introducing the concept of thread coordination

Learning is guided through several mechanisms, such as scaffolding, by which challenges increase progressively in complexity, or problem-based learning (Hung, Jonassen, & Liu, 2008), by which concepts are learned by solving concrete (simulated) problems. The simulation instantly shows the results (success, collision, deadlock) of the student's code, facilitating immediate understanding and debugging. In this regard, the timeline view aids the understanding of the code execution and simplifies error correction. Following the concepts of game-based learning, students can freely test solutions without real-world consequences, allowed to re-attempt when the code execution does not yield the intended outcome. Elements like clear goals per challenge, progression through levels, and the intrinsic satisfaction of gamification in solving complex problems were introduced to enhance intrinsic motivation and lower fatigue perception. Moreover, the platform was designed to be used with teacher guidance, including contextualization, support during exercises, and debriefing sessions to consolidate learning and promote transfer of knowledge—that is, the students' ability to apply concurrency and synchronization principles learned within the platform to solve new, unseen programming problems beyond the Railway Quest environment.

3.2 Architecture

Railway Quest is a web-based platform. The main architecture of the platform was designed around the concept of having students write code on their local machines using their preferred Integrated Development Environment (IDE) and viewing the results of their code execution via the web-based visualization interface, as shown in Figure 3. The execution of the program initiates REST-based communications with the system's back-end running remotely, which conducts the simulation and supplies real-time data to the web-based visualization interface, implemented through an embedded Unity application.

When logging-in on the platform and starting a new attempt for a challenge, students are assigned a token that must be included in their code. This allows the back-end to tie the client code execution with the attempt and show the Unity train interface of the challenge on the student's web browser.

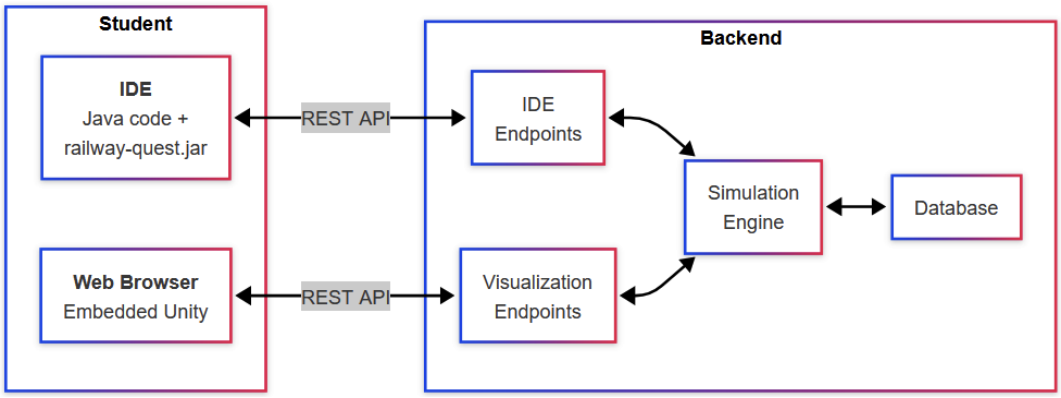

Figure 3: Architecture of the platform

The back-end simulation engine is structured around a discrete-event simulation model, which allows for control over the sequencing and coordination of concurrent events. This engine is responsible for replicating the execution behavior of the student's code and evaluating the actions of the simultaneously running threads. The simulation engine ensures the correct execution order, therefore maintains consistency with the logical flow dictated by the student's program. In addition to orchestrating execution, the engine incorporates mechanisms for detecting key concurrency problems such as collisions or resource access conflicts and updates the visualization accordingly in real time.

Figure 4 shows some example code snippets of a solution for a challenge which asks students to transfer wood chunks from the deposit to the factory by coordinating trains at the crane, as shown in Figure 2.

Students are tasked to define different Java classes to solve the challenge: one refers to the level itself ("WoodWorker") which contains the class definition of GreenTrain (right-hand side code) and RedTrain (left-hand side code, on the bottom). The WoodWorker class also maintains important state, including a ReentrantLock, a Condition, and boolean flags (trainIsWaiting, transferCompleted) to coordinate the synchronization logic during wood transfer. The main() function, defined in the bottom-right corner, orchestrates the simulation by creating an instance of the WoodWorker class and setting the unique token generated by the platform to address this specific attempt. The execute() function initiates the attempt by retrieving the crane, creating and registering both trains, and starting their threads. The transferWood() method leverages locking and condition signaling to coordinate safe and ordered interaction between trains at the crane. By completing the challenge, students are allowed to move to the next challenge in the series.

4. Preliminary Test

Railway Quest was initially introduced at our university as part of the "Concurrent and Parallel Programming" course, during the first half of the Spring 2025 semester, where a total number of approximately 60 students were actively enrolled. The course was conducted weekly over a four-hour period, comprising one hour of traditional lecture, followed by two hours of laboratory sessions, and one hour of asynchronous activities (where students work independently on assignments). During the laboratory sessions, in which students practiced core CPP elements under the guidance of the instructors, they were tasked with solving challenges presented through Railway Quest, in particular for the course topics of "Introduction to thread execution", "Race conditions and synchronization of threads", and "Coordination of threads with conditional variables". Teachers introduced the challenges (a total of seven fully developed levels) and students were prompted to formulate solutions by applying the CPP principles and techniques recently covered in the lecture. Lecturers were available during the laboratory sessions to help students figure how to solve the task, and to observe their interactions with the platform. At the end of the laboratory sessions, a small debriefing session took place, in which lecturers gathered insights about how the solution was found by the students.

Figure 5 displays the interface for "Level 3 - Station" on the Railway Quest web platform. The top section presents the student with the description of the scenario (e.g., two trains accessing a shared station needing synchronization), specific goals (e.g. ensuring 10 station entries and preventing collisions), and important level elements. These elements inform about specific parameters such as the number of trains and stations, suggest required programming tools like ReentrantLocks, and outline necessary train actions like calling enterStation(),

exchangePassengers(), and leaveStation(). A supplementary visual panel provides a representation of the level's layout. Beneath this section, an 'Attempts' panel presents the student's activity history for the current level in tabular format. The table includes columns for attempt number, status, resolution (SUCCEEDED/FAILED), duration, number of code executions, and date. Each previous attempt can be reviewed through an associated action button. Additionally, students may initiate a new attempt by selecting the 'New Attempt' button.

```java
public class WoodWorker extends RailwayQuest {
    private Crane crane = null;
    private final Lock lock = new ReentrantLock();
    private final Condition condition = lock.newCondition();
    private boolean trainIsWaiting = false;
    private boolean transferCompleted = false;

    private void transferWood() throws InterruptedException {
        lock.lock();
        try {
            if (trainIsWaiting) {
                crane.transfer();
                transferCompleted = true;
                condition.signal();
            } else {
                trainIsWaiting = true;
                while (!transferCompleted)
                    condition.await();
                trainIsWaiting = false;
                transferCompleted = false;
            }
        } finally {
            lock.unlock();
        }
    }

    private class RedTrain extends Train implements Runnable {
        @Override
        public void run() {
            for (int lap = 1; lap <= 20; lap++) {
                move( nSteps: 6); // Go to deposit
                load();          // Load wood
                move( nSteps: 4); // Go to crane
                try {
                    transferWood();
                } catch (InterruptedException e) {
                    return;
                }
                move( nSteps: 4); // Go to spawn point
            }
        }
    }

    private class GreenTrain extends Train implements Runnable {
        @Override
        public void run() {
            for (int lap = 1; lap <= 20; lap++) {
                move( nSteps: 4); // Go to crane
                try {
                    transferWood();
                } catch (InterruptedException e) {
                    return;
                }
                move( nSteps: 4); // Go to the factory
                unload();         // Unload wood
                move( nSteps: 6); // Go to spawn point
            }
        }
    }

    public void execute() {
        startAttempt();
        crane = getCrane(Crane.Label.A);
        RedTrain redTrain = new RedTrain();
        GreenTrain greenTrain = new GreenTrain();
        register(redTrain, Train.Color.Red);
        register(greenTrain, Train.Color.Green);
        List<Thread> threadList = Stream.of(redTrain, greenTrain)
                .map(Thread::new).peek(Thread::start).toList();
        for (Thread thread : threadList) {
            try {
                thread.join();
            } catch (InterruptedException e) {
                System.err.println("Interruption occurred.");
            }
        }
        terminateAttempt();
    }

    public static void main(String[] args) {
        final WoodWorker quest = new WoodWorker();
        quest.setToken("LEVL-098J-A73Y-7SJN-RBQW");
        quest.execute();
    }
}
```

Figure 4: A possible Java-based solution for the challenge presented in Figure 2

Figure 6 shows a failed attempt. Both trains tried to access a train station, no synchronization was used in the program code, which resulted in a collision at the entrance of the station. In this scenario, the code reflects two threads concurrently attempting to access a critical section without using synchronization mechanisms such as locks. By hovering on the timeline view, students can understand what happened: a collision between two trains.

After the first half of the semester, students were asked to complete an anonymized interim survey aimed to measure their satisfaction with the course. The survey was answered by 45 students, and included Likert-scale questions rating the usefulness of various course components and open-ended questions for qualitative feedback. The 45 survey respondents represent 75% of the approximately 60 students actively enrolled in the course. This response rate provides substantial preliminary insight into the student perceptions within this cohort. The demographic characteristics of the respondents are assumed to generally align with the overall class composition of second-year computer science undergraduates at our institution.

Within the survey, two specific questions addressed the perceived utility of Railway Quest for understanding concepts and its overall usefulness compared to other resources mentioned in class. Finally, an open-ended comment was left for students to express their feelings and ideas towards the platform.

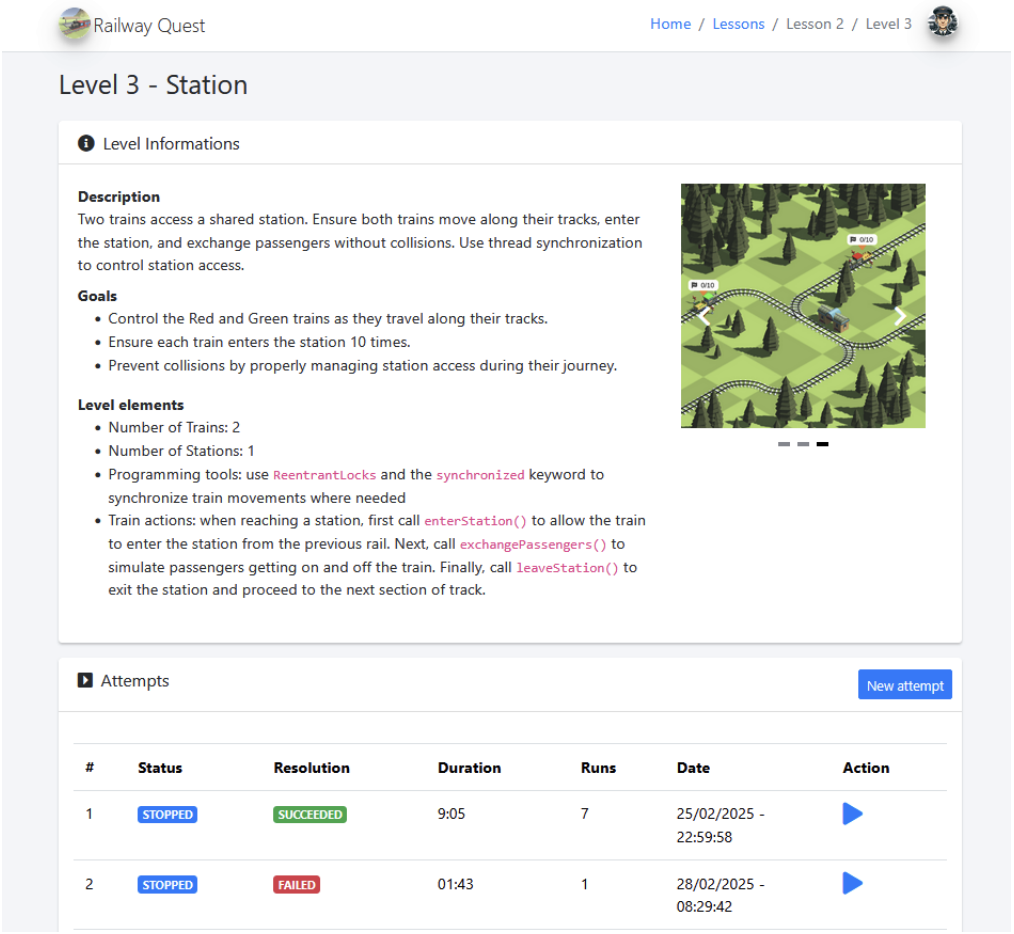

Figure 5: Web interface of one level on the Railway Quest web-based platform

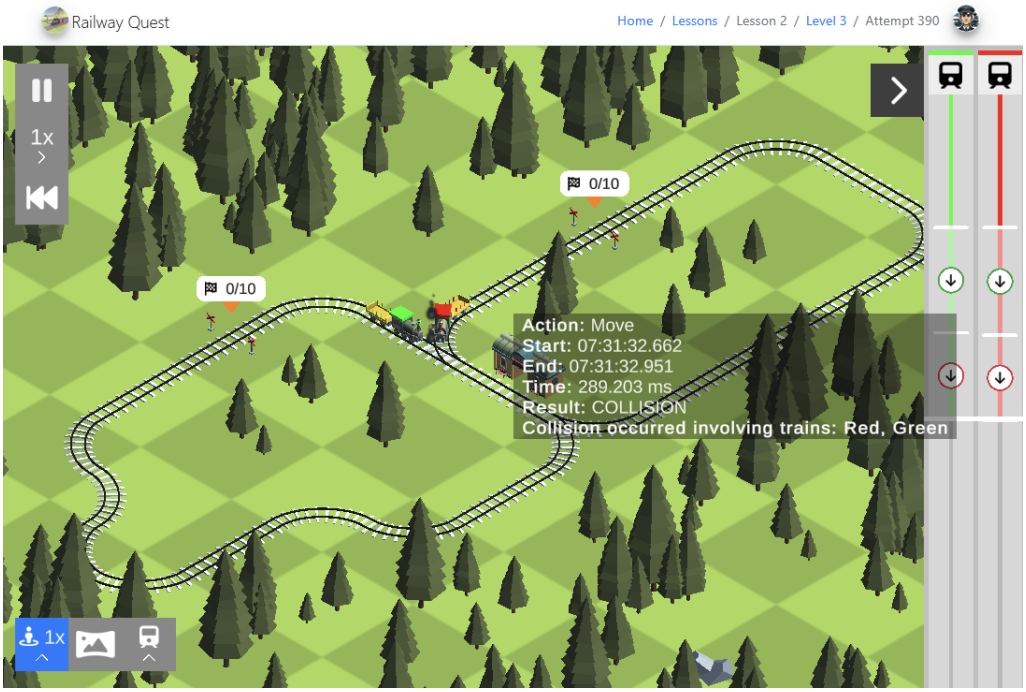

Figure 6: Trains collided, the attempt failed

5. Results

The first question regarding Railway Quest was "Railway Quest made it easier and quicker for me to grasp the concepts during both group and solo exercises". The answers are shown in Figure 7.

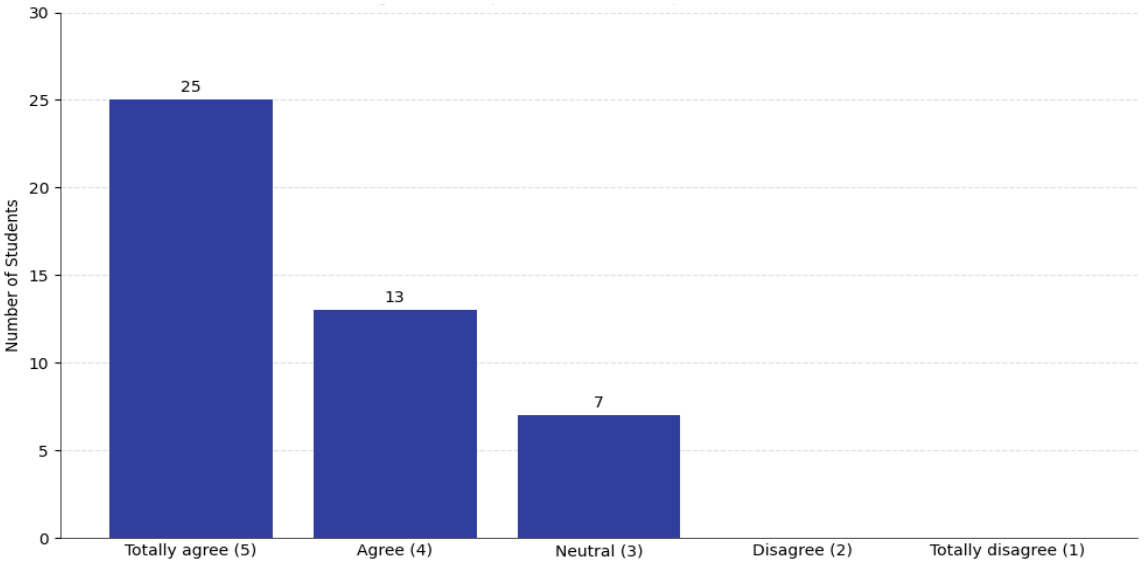

Figure 7: Answers for "Railway Quest made it easier and quicker for me to grasp the concepts during both group and solo exercises"

The average score given by students was 4.4 out of 5 (N=45, with 84% agreeing or totally agreeing). Among the various items in the questionnaire addressing the educational tools employed during the course, this question yielded the highest mean score, indicating a particularly positive student perception of the platform. Such result strongly suggests that students perceived Railway Quest as being very effective at its core goal: breaking down complex, abstract topics into something more understandable and visually intuitive. The fact that 84% of students rated it a 4 or 5 further emphasizes this widespread positive perception of its impact on their learning and comprehension.

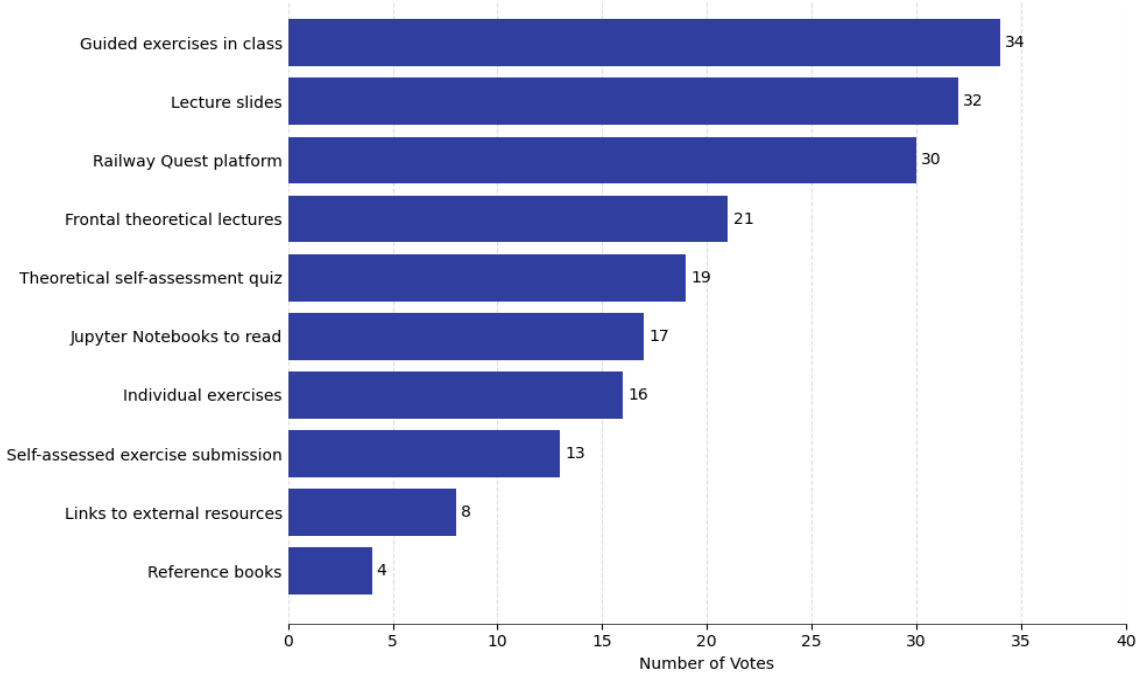

Figure 8: Answers for "What activities or resources do you find particularly useful for your learning?"

The second question was a multi-choice question asking, "What activities or resources do you find particularly useful for your learning?". Answers are shown in Figure 8. Railway Quest ranked 3rd (out of 10 options, receiving 30 votes), behind guided exercises and lecture slides. This ranking could imply that students view Railway Quest not just as a novelty, but as a valuable and central part of their learning experience during the course. Its perceived utility is comparable to fundamental teaching methods like practical exercises and core theoretical materials (slides). This high ranking validates its importance within the course structure from the students' perspective and suggests it effectively complements other teaching approaches.

The final open-ended comments reinforced the quantitative data results. Students praised Railway Quest for providing "*an excellent possibility to visualize the concept of Threads and various concurrency problems*". Many highlighted the engaging and motivating aspects ("*I love the trains! They are pure dopamine!*", "*beautiful... very fun and stimulating*", "*the best addition to encourage doing the exercise series because you get immediate visual feedback and it's very pleasant*"). The platform was perceived as more effective for immediate understanding than traditional terminal output. Constructive criticism mainly focused on a desire for "*more levels"* potentially optional ones, and suggestions to increase its usage within the course.

These student testimonials corroborate the platform's aim to enhance motivation and engagement through its visual and interactive nature, as claimed in Section 1. The high ratings for enjoyment and perceived utility in understanding concepts further support this.

Moreover, these results encourage further development of the platform, with the addition of more levels and gamification elements (i.e., badges, visual progression, rankings, ...), as well as support for lecturers (i.e., lecturer dashboard with insight on usage data, level editor, ...). There were no negative comments regarding Railway Quest, or its use within the laboratory sessions.

6. Conclusion and Future Work

This paper presented Railway Quest, a platform for innovative application of GBL principles to the challenging domain of concurrent and parallel programming. By translating abstract concepts into a visual, interactive, and gamified railway simulation, it aims to lower the cognitive barriers associated with learning CPP.

Preliminary results from its first deployment are very positive, indicating that students find the platform highly useful, engaging, and effective in making complex concepts more understandable. The combination of immediate visual feedback and gameful challenges appears to be a successful strategy for this specific learning context.

Future work will focus on expanding the content by adding more diverse and advanced challenges, potentially including optional ones, based on students' feedback. We also plan to develop a teacher dashboard to better support monitoring and formative assessment and conduct a more rigorous evaluation of learning outcomes beyond student perception. Exploring support for other programming languages and integration with Learning Management Systems as a MOOC are also potential future directions, as well as collaborative development scenarios or even adapting the curriculum for different educational contexts, including advanced high school courses. The preliminary findings presented in this paper are based on student self-reported perceptions and satisfaction. The current study did not include a comparative analysis with a control group or direct measures of learning outcomes (e.g., comparative exam scores or standardized concept inventories), which could be part of a future work.

We believe Railway Quest demonstrates the significant potential of tailored, gamified simulations as effective pedagogical tools for complex topics in computer science education, fostering both deeper understanding and increased student engagement.

Ethics Declaration: The interim feedback survey described in this paper was conducted in accordance with the institutional guidelines for course evaluation at our institution. Participation was voluntary, and student responses were collected anonymously to ensure confidentiality and protect privacy. Standard procedures for obtaining informed consent for participation in course feedback activities were followed.

AI declaration: We confirm AI tools were not used during the creation of this paper.

References

Abernethy, M., Sinnen, O., Adams, J., De Ruvo, G., & Giacaman, N. (2018). ParallelAR: An augmented reality app and instructional approach for learning parallel programming scheduling concepts. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (pp. 324-331). IEEE.

Akimoto, N., & Cheng, J. (2003). An educational game for teaching and learning concurrency. In *Proceedings of the 1st International Conference on Knowledge Economy and Development of Science and Technology (KEST'03), Honjo, Japan* (pp. 34-39).

Barnes, T., Powell, E., Chaffin, A., & Lipford, H. (2008). Game2Learn: improving the motivation of CS1 students. *Proceedings of the 3rd international conference on Game development in computer science education (GDCSE '08).* Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/1463673.1463674

Ben-Ari, M., & Kolikant, Y. B. D. (1999). Thinking parallel: The process of learning concurrency. *ACM SIGCSE Bulletin*, *31*(3), 13-16.

Ben-Ari, M. (2001) Constructivism in Computer Science Education, *Journal of Computers in Mathematics and Science Teaching*, 20(1), pp. 45–73. Available at: https://www.learntechlib.org/p/8505

Cliburn, D.C. (2006) 'The Effectiveness of Games as Assignments in an Introductory Programming Course', in *Proceedings. Frontiers in Education. 36th Annual Conference*. pp. 6–10. doi: 10.1109/FIE.2006.322314.

Buzek, E., & Kruliš, M. (2018). An entertaining approach to parallel programming education. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (pp. 340-346). IEEE.

El-Nashar, A.I. and Nakamura, M. (2013) *To parallelize or not to parallelize, bugs issue*. arXiv preprint arXiv:1311.0728. Available at: https://arxiv.org/abs/1311.0728

Harteveld, C., Smith, G., & Carmichael, G. (2014). A Design-Focused Analysis of Games Teaching Computer Science.

Hughes, C., Buckley, J., Exton, C., O'Carroll, D., & SVCR Group. (2005). Towards a framework for characterising concurrent comprehension. *Computer Science Education*, *15*(1), 7-24.

Hung, W., Jonassen, D. H., & Liu, R. (2008). Problem-based learning. In *Handbook of research on educational communications and technology* (pp. 485-506). Routledge.

Ibanez, M. B., Di-Serio, A., & Delgado-Kloos, C. (2014). Gamification for engaging computer science students in learning activities: A case study. *IEEE Transactions on Learning Technologies, 7*(3), 291-301.

Ibrahim, R., Yusoff, R. C. M., Mohamed-Omar, H., & Jaafar, A. (2011). Students perceptions of using educational games to learn introductory programming. *Computer and Information Science*, *4*(1), 205.

Nakamura, J., & Csikszentmihalyi, M. (2002). The concept of flow. *Handbook of positive psychology*, *89*, 105.

Oktaviati, R., & Jaharadak, A. A. (2018). The impact of using gamification in learning computer science for students in university. *International Journal of Engineering & Technology, 7*(4.11), 121-125.

Papastergiou, M. (2009). Digital game-based learning in high school computer science education: Impact on educational effectiveness and student motivation. *Computers & education*, *52*(1), 1-12.

Zhu, J., Alderfer, K., Furqan, A., Nebolsky, J., Char, B., Smith, B., Villareale, J., & Ontañón, S. (2019). Programming in game space: how to represent parallel programming concepts in an educational game. In *Proceedings of the 14th International Conference on the Foundations of Digital Games* (pp. 1-10).

Zhu, J., Alderfer, K., Smith, B., Char, B. and Ontañón, S. (2020) *Understanding Learners' Problem-Solving Strategies in Concurrent and Parallel Programming: A Game-Based Approach*. arXiv preprint arXiv:2005.04789. Available at: https://arxiv.org/abs/2005.04789