The Eurasia Proceedings of Science, Technology, Engineering & Mathematics (EPSTEM)
ISSN: 2602-3199

The Eurasia Proceedings of Science, Technology, Engineering and Mathematics (EPSTEM), 2025

Volume 34, Pages 172-181

ICBASET 2025: International Conference on Basic Sciences, Engineering and Technology

Performance Comparison of WebAssembly and Phaser in Procedural Maze Generation

Noa Midzic
University of Zagreb

Matija Novak
University of Zagreb

Abstract: Phaser and WebAssembly are both excellent options for web game development, each offering unique advantages. This paper specifically focuses on procedural maze generation and compares the performance of WebAssembly (using C++ and SDL) and Phaser across various levels of complexity, defined by maze dimensions. The study aims to assess execution time efficiency and identify performance trends across different scenarios. Using the Depth-First Search (DFS) algorithm, we conducted multiple test runs at increasing maze sizes to evaluate the scalability and computational efficiency of both technologies. The results indicate that WebAssembly enables near-native execution of C++ code in web browsers, leading to significant performance advantages in larger maze dimensions. Specifically, WebAssembly exhibited a more consistent, linear increase in execution time, making it a strong candidate for handling computationally intensive tasks. In contrast, Phaser performed slightly better in smaller maze dimensions but demonstrated greater variability and steeper increases in execution time as complexity grew. The findings suggest that while Phaser is an accessible and user-friendly framework for game development, its performance may not scale as efficiently as WebAssembly when dealing with complex procedural generation tasks. However, the study also acknowledges limitations, such as fewer test runs for larger mazes and potential differences in implementation efficiency between the two technologies. Overall, WebAssembly appears better suited for tasks requiring high-performance execution, whereas Phaser remains a viable option for less demanding applications where ease of use and rapid development are prioritized.

Keywords: Phaser, WebAssembly, Performance comparison, Experiment, Web game development

Introduction

Game development has emerged as a significant subfield of software engineering, driven by the widespread adoption of computer games (Wu et al., 2012). Additionally, game development has gained prominence as an effective pedagogical tool for teaching programming, as illustrated in Cagiltay (2007) or Wilson et al. (2024). A variety of frameworks exist to support this purpose, with Phaser (Faas, 2017) being a good choice for web-based game development, which is the focus of this paper. Phaser offers advantages due to its versatility, enabling the development of both basic web games and more serious games (Pavkov et al., 2017) as project requirements evolve. The selection of an appropriate framework is largely dependent on the specific requirements and genre of the game under development.

On the other hand, frameworks like Phaser, which is written in JavaScript, tend to have more overhead compared to games developed in C++. With the advent of WebAssembly, it is now possible to develop web games using C++, as demonstrated in Battagline (2019). In a previous work Midžić (2024) one of the objectives was to create a browser game using C++ with WebAssembly, which raised the question of the potential benefits of game develop- ment with WebAssembly compared to traditional frameworks used for browser games. This

- This is an Open Access article distributed under the terms of the Creative Commons Attribution-Noncommercial 4.0 Unported License, permitting all non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

- Selection and peer-review under responsibility of the Organizing Committee of the Conference

© 2025 Published by ISRES Publishing: www.isres.org

question forms the central focus of this study, which compares the performance of WebAssembly with C++ and SDL against Phaser in the procedural generation of a game world, exemplified by a dynamically generated maze.

WebAssembly is a binary instruction format that serves as a compilation target for programming languages like C, C++, and Rust. It allows these languages to be executed in web browsers at near-native speeds, enabling developers to write performance-critical parts of web applications in languages other than JavaScript. It has gained widespread support from major browsers, but compiling source code into WebAssembly requires the use of additional tools such as Emscripten to integrate C/C++ into web applications. On the other hand, Phaser is an open-source JavaScript framework for building HTML5 games, widely adopted due to its simplicity and extensive documentation. Because of its popularity and rapid development capabilities, Phaser was chosen as a suitable framework for browser game development to compare against a WebAssembly-based game.

The procedural generation of game worlds refers to using algorithms to create game environments dynamically, rather than designing them manually. This approach is often used in game development to both reduce storage requirements for game assets and offer non-repeating or limitless variations in level design. Since the complexity of procedural generation algorithms and the computational resources required for real-time generation can impact performance, particularly on less powerful devices, optimisation is necessary to ensure smooth performance in games across different platforms.

A review of the literature shows that several studies have compared the performance of JavaScript and WebAssembly. A 2022 study by Macedo et al. (2022) on the runtime and energy efficiency performance between WebAssembly and JavaScript demonstrates gains in favour of WebAssembly. Similarly, other studies like Herrera et al. (2018) or Reiser et al. (2017) also show the advantage of WebAssembly compared to JavaScript.

In the context of gaming performance, there are some experiments that highlighted WebAssembly's benefits over JavaScript. In (Ylenius, 2020) three versions of the same game were implemented: one in pure JavaScript, one using a combination of JavaScript and WebAssembly, and one entirely in WebAssembly. The WebAssembly version not only demonstrated the best execution performance but also the most stable memory usage. However, no studies were found that directly compare WebAssembly's performance to Phaser or other game frameworks. Only mention of Phaser and Webssembly was in a master thesis (Mustonen, 2023) that compared web based engine design.

The motivation behind this article is to research whether WebAssembly can be used to optimise the performance in procedural generation of game worlds, compared to a JavaScript framework such as Phaser. In the context of a procedurally generated game world, a maze generation algorithm implemented with Depth-First Search (DFS) was chosen due to its sufficient time complexity for performance testing. Section 2 describes the structure of the projects created for performance comparison, the methods used for performance evaluation and measurement, as well as the specifications of the device used to conduct testing. In Section 3 the results of the measurements are presented, analysed and discussed while Section 4 presents the conclusion.

Method

The code used for testing implements a maze generator utilizing a Depth-First Search (DFS) algorithm with backtracking, written in C++ for the WebAssembly version and in JavaScript for the Phaser version. It creates a grid of cells, each initially surrounded by walls. The algorithm starts at a random cell, marks it as visited, selects an unvisited neighbouring cell, removes the wall between them, and moves to the next cell. This process repeats until all cells are visited, resulting in a complete maze with a single path from start to finish. If the algorithm encounters a cell with no unvisited neighbours, it backtracks to the previous cell and continues. The final output is a randomly generated maze with a path from upper left corner to lower right corner.

In terms of code, Phaser not only manages the maze generation but also the rendering, making it a higher-level solution for building browser games. In contrast, the WebAssembly version is implemented in C++ using SDL2 for rendering, which involves directly manipulating the pixel buffer and provides a lower-level control over the rendering process. Both implementations employ the same DFS with backtracking algorithm, ensuring equivalence in logic used for maze generation. However, their performance may vary due to differences in the environment and tools. As a high-level JavaScript framework, Phaser carries the overhead of JavaScript's garbage collection and additional processing related to its game engine features. In contrast, WebAssembly

benefits from direct memory access, allowing it to bypass garbage collection. The code, along with the measurements and the R script used for result analysis, is available at *https://github.com/nmidzic20/phaser-wasm-maze-generator*.

The measurements were conducted by running tests on mazes of increasing size: 10x10, 30x30, 60x60, 90x90 (with 100 test runs each), 110x110 (50 runs), and 130x130 and 150x150 (20 runs each). The goal was to compare performance both for small mazes, which require only a few minutes to generate, and large mazes, which can take several hours. Tests were carried out in order from smaller to larger dimensions, with each dimension repeated either 100, 50, or 20 times to obtain an average for that size. To avoid bias, test were conducted alternately – one run for Phaser, followed by one for WebAssembly, and so on. Since larger mazes take significantly longer to generate, the number of runs for these mazes was reduced due to time constraints.

During the analysis of the results, a descriptive statistical analysis was performed, which included visualizing the data with boxplots and calculating measures such as quantiles, mean, median, standard deviation, and variance. The Shapiro-Wilk test was applied to assess the normality of the data distributions. Based on the Shapiro-Wilk test, if the p-value was greater than 0.05, indicating that the data followed a normal distribution, parametric tests were conducted. Conversely, if the p-value was less than 0.05, indicating that the data did not meet the normality assumption, non-parametric tests were performed.

These tests were employed to determine whether there was a statistically significant difference in performance times between the Phaser and WebAssembly technologies across the generation of different maze dimensions.
The technologies employed in this study were WebAssembly and Phaser, with C++ and JavaScript as the respective programming languages for the test code. Additionally, an R script and R Studio were used for analysing the measurements. The testing device was equipped with an Intel Core i7-10750H CPU running at 2.60 GHz and 16 GB of RAM, with Google Chrome serving as the browser for testing. The measurements were collected over the course of one week, with no significant breaks between the test runs for a particular maze dimension category.

Results and Discussion

The analysis involves performance measurements of two technologies, Phaser and WebAssembly, across various maze dimensions used in the testing. Average execution times for each technology were collected, and the results are presented in Table 1.

Table 1. Average time for maze generation comparison between WebAssembly and Phaser

Dimension	Number of test runs	WebAssembly (miliseconds)	Phaser (miliseconds)
10x10	100	3.359,14	3.278,96
30x30	100	30.017,28	29.942,66
60x60	100	119.993,50	119.925,40
90x90	100	269.948,00	270.331,50
110x110	50	403.215,90	405.354,80
130x130	50	563.200,80	723.647,30
150x150	20	749.787,20	1.127.674,00

Figure 1 illustrates the average performance times across all maze dimensions, recorded in milliseconds. Both Phaser and WebAssembly show a gradual increase in mean execution times as the maze dimensions expand, but the increase is more pronounced for Phaser, especially from dimension 130x130 onward, where Phaser's mean execution time becomes significantly higher than that of WebAssembly.

At smaller dimensions, Phaser has a slight speed advantage over WebAssembly on average, though this advantage diminishes as the maze increases. Specifically, Phaser is approximately 2,39% faster for 10x10 mazes, 0,25% faster for 30x30, and 0,06% faster for 60x60 mazes. However, starting at 90x90 dimension, Phaser's execution time increases more rapidly than that of WebAssembly, leading to WebAssembly consistently outperforming Phaser at larger dimensions. WebAssembly is 0,14% faster at 90x90 and 0,53% faster at 110x110. The performance gap becomes significantly more pronounced for larger mazes, with WebAssembly being 22,17% faster at 130x130 and 33,51% faster at 150x150 dimensions.

This trend is more clearly illustrated in Figure 2, which shows the difference in median execution time between the two technologies. After a minimal increase at smaller maze sizes, a sharp jump occurs at the 110x110 dimension, followed by rapid linear growth.

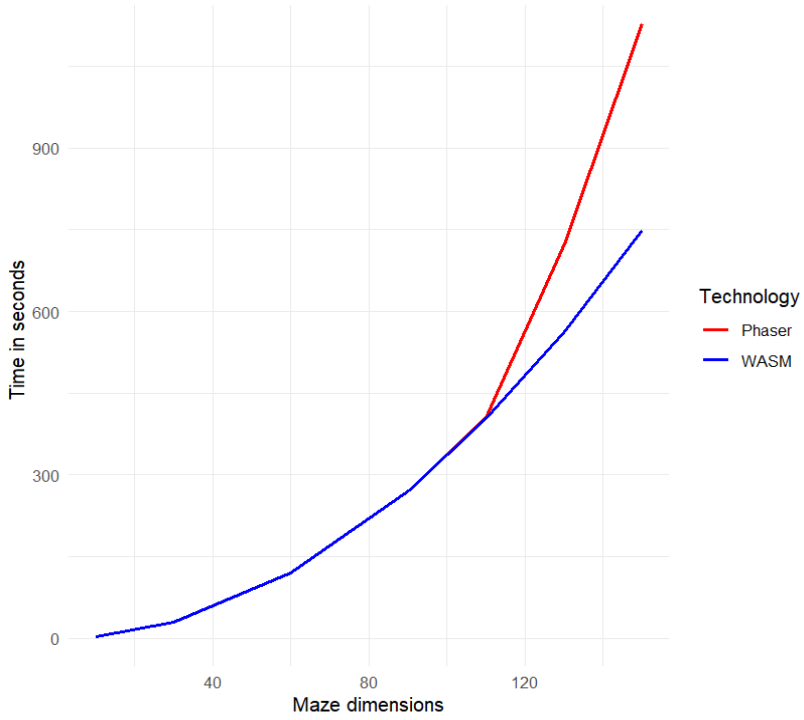

Figure 1. Comparison graph for Phaser and WebAssembly by means in seconds for each dimension

The distribution of execution times in milliseconds for each maze dimension can be evaluated by comparing the mean and median values. When the mean and median are the same or nearly identical, it suggests a normal distribution, while a significant difference between them indicates skewness, either to the left (negative skew) or to the right (positive skew). Table 2 outlines the distributions for each dimension and technology category.

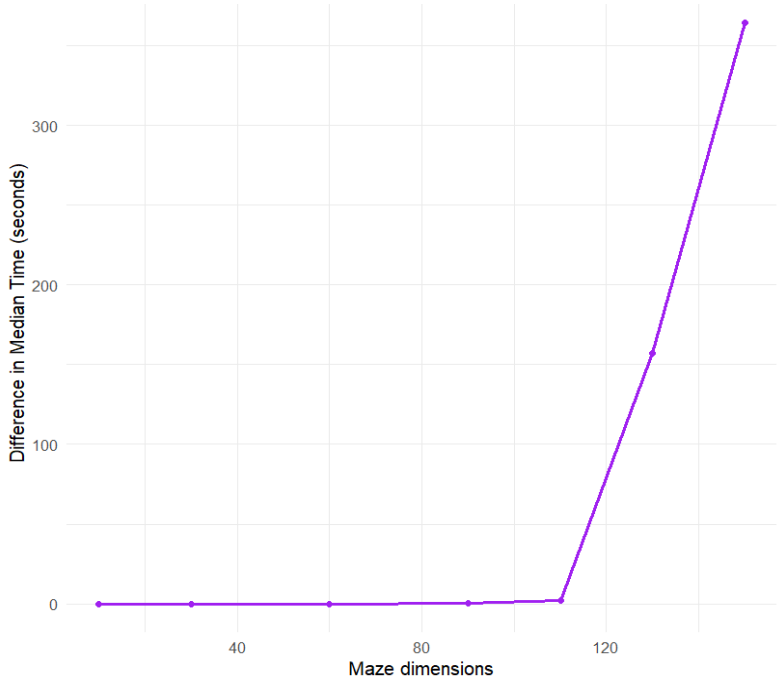

Figure 2. Graph of the median time difference in seconds for Phaser and WebAssembly for each dimension

Phaser tends to show slight to significant positive skew for larger maze dimensions (90, 110, 130, 150), with the skew becoming more pronounced as the dimensions increase. In contrast, WebAssembly generally displays a normal distribution for most dimensions. Boxplot analysis indicates that Phaser has more outliers than WebAssembly, particularly as the maze dimension increases. For example, dimension 110x110 as shown in Figure 3 or dimension 150x150 as shown in Figure 4.

Table 2. Data distribution by dimension and technology

Dimension	Phaser Distribution	WASM Distribution
10x10	Normal	Slight positive skew
30x30	Normal	Normal
60x60	Normal	Normal
90x90	Slight positive skew	Normal
110x110	Slight positive skew	Normal
130x130	Positive skew	Normal
150x150	Strong positive skew	Normal

We can analyse whether the spread (variability) of performance times increases more rapidly for Phaser or WebAssembly by using the standard deviation or interquartile range (IQR) as measures of spread, and plotting how these values change across the dimensions. Figure 5 shows the comparison using standard deviation, which measures the overall variability of the data. Figure 6 displays the comparison using interquartile range, which indicates the range within which the middle 50% of the data falls (between the 1st and 3rd quartiles), thereby reducing the impact of outliers on the comparison.

As maze dimensions increase, the spread in performance times grows more rapidly for Phaser than for WebAssembly, particularly from the 60x60 dimension onwards, where outliers begin to emerge, with the greatest spike occurring at the 110x110 dimension. This indicates that larger mazes lead to more variability in performance, particularly for Phaser, where execution times are more inconsistent. In many cases, Phaser struggles to maintain stable performance, resulting in a greater number of outliers.

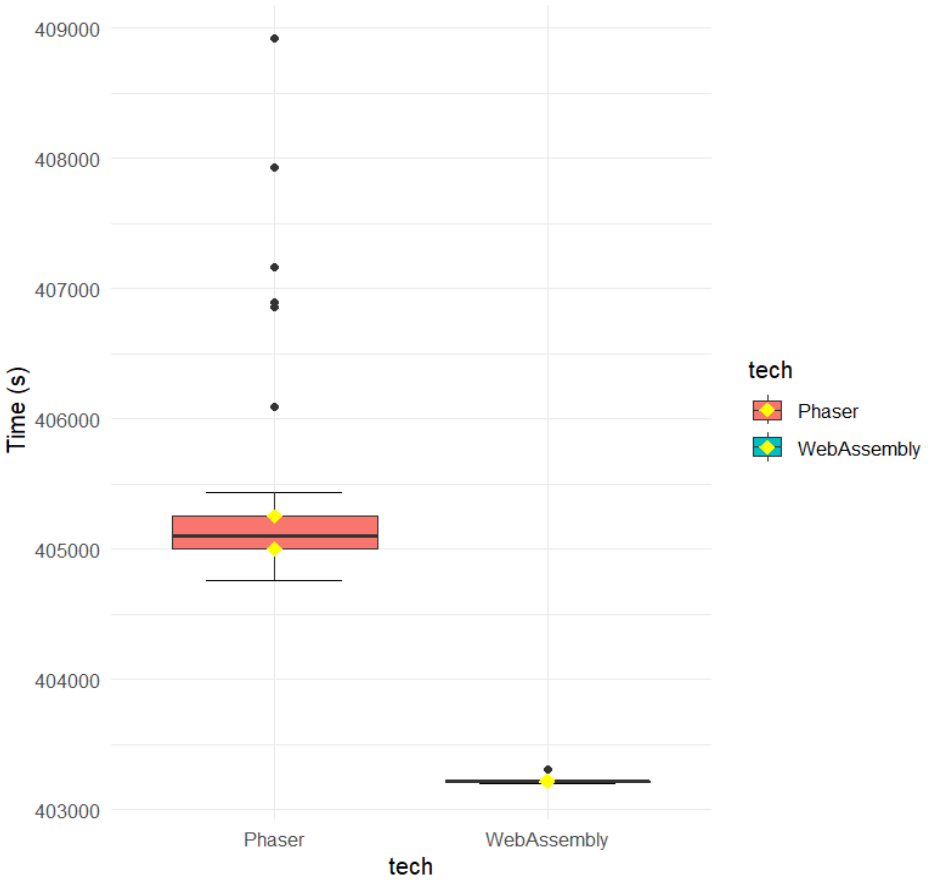

Figure 3. Boxplot of execution time in milliseconds for 110x110 dimension

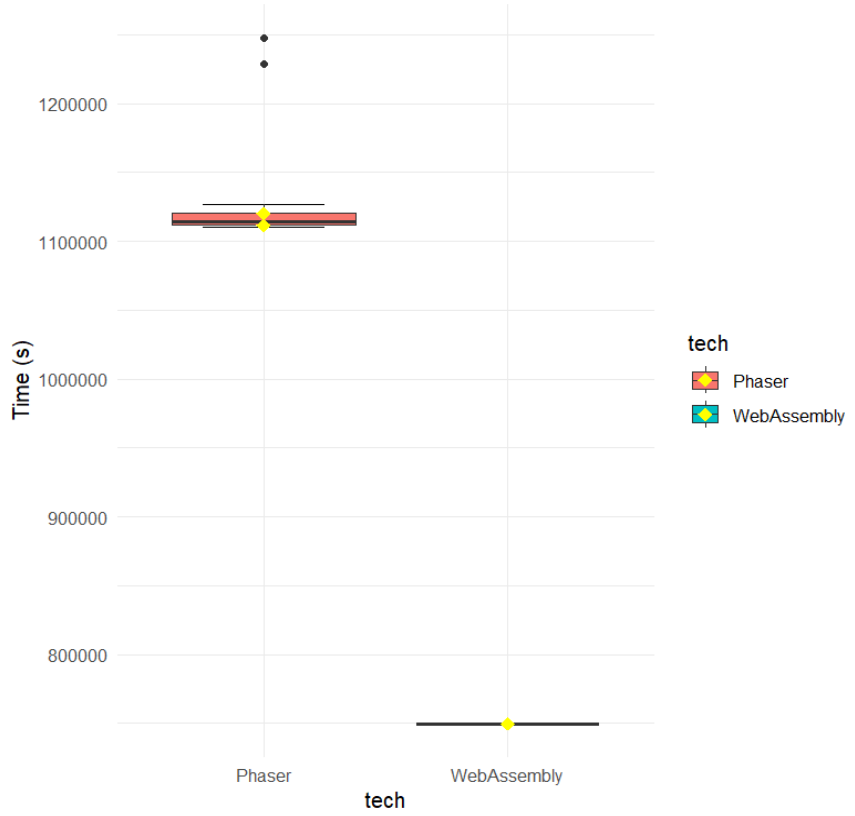

Figure 4. Boxplot of execution time in milliseconds for 150x150 dimensions

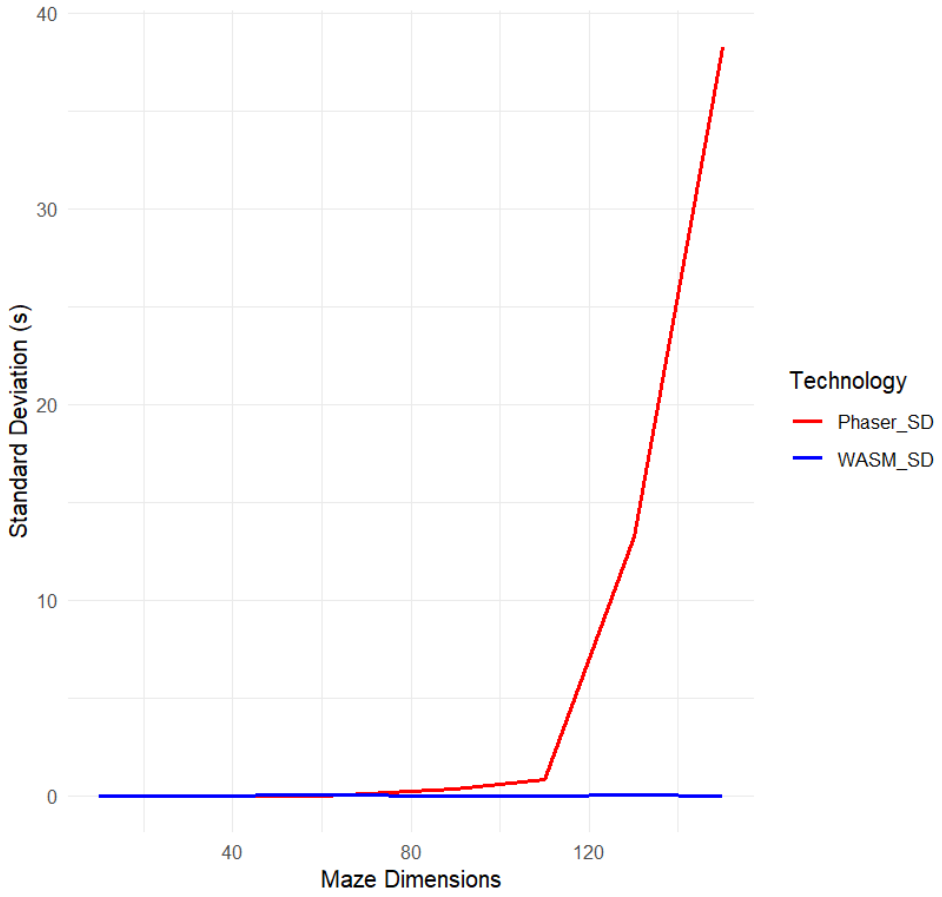

Figure 5. Comparison of standard deviation in performance times between Phaser and WebAssembly across maze dimensions

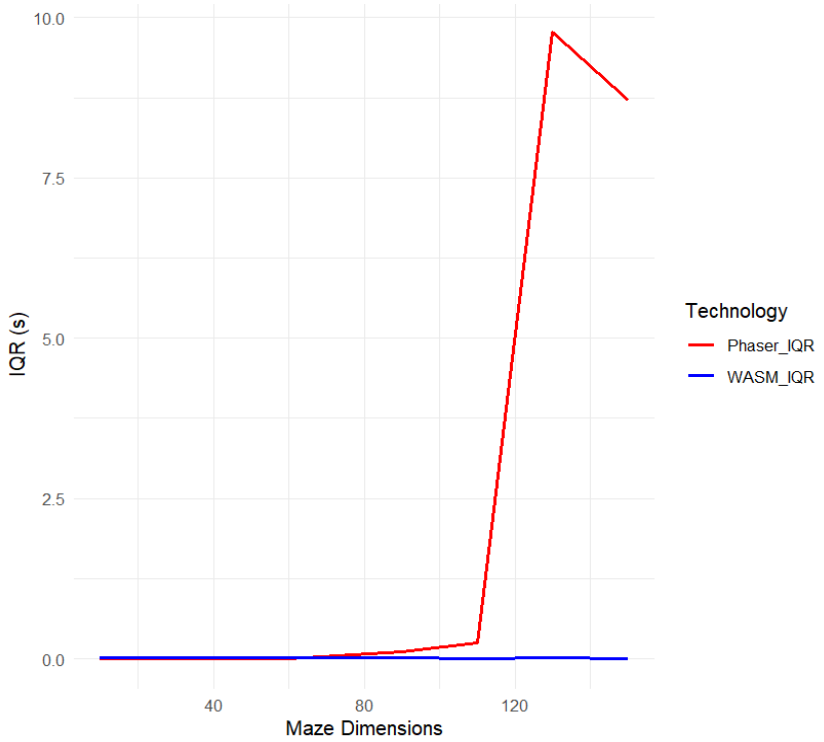

Figure 6. Comparison of interquartile range (IQR) in performance times between Phaser and WebAssembly across maze dimensions

Discussion

To determine whether a parametric or non-parametric test should be used to assess whether the difference in average execution time between Phaser and WebAssembly is statistically significant, Shapiro-Wilks tests were conducted on the data for each maze dimension. All dimensions for both technologies failed the normality test (p-values <0.05, as shown in Table 3, indicating that the execution time data does not follow a normal distribution. As a result, the non-parametric Mann-Whitney-Wilcoxon test was chosen over the parametric t-test to evaluate statistical significance.

Table 3. Comparison of average time in milliseconds to complete maze generation for different maze dimensions between WebAssembly and Phaser

Dimension	p-value	Significance	Faster
10x10	< 2,2e-16	Significant	Phaser
30x30	< 2,2e-16	Significant	Phaser
60x60	< 2,2e-16	Significant	Phaser
90x90	< 2,2e-16	Significant	WebAssembly
110x110	< 2,2e-16	Significant	WebAssembly
130x130	5,181e-08	Significant	WebAssembly
150x150	4,695e-08	Significant	WebAssembly

All evaluated dimensions demonstrate significant differences between the two subsets, with very low p-values (< 0.001) providing strong evidence against the null hypothesis that the location shift is equal to zero, indicating that there is no significant difference in time performance between the two technologies. Regarding the number of test runs performed (as listed in Table 1), the time required for maze generation increases dramatically with maze dimensions. Due to time constraints, it was not feasible to run 100 tests for each dimension. Consequently, 50 test runs were performed for 110x110 maze, while 20 test runs were conducted for 130x130 and 150x150 mazes.

The p-values for 130x130 and 150x150 dimensions (p = 5,181e-08 and p = 4,695e08, respectively), though still very small, are larger than those for other dimensions, indicating a less pronounced difference in performance times between Phaser and WebAssembly in these cases. One possible explanation is the smaller number of test runs for these higher dimensions (20 runs, compared to 50 or 100 for others), which reduces the power of the

statistical tests, making them less sensitive to detecting differences. Another factor could be the wider spread in data distribution at higher dimensions. Since the Mann-Whitney test compares ranks rather than actual values, this increased variability may cause overlapping ranks between Phaser and WebAssembly. Although the mean values show the greatest differences between the two technologies at these dimensions, the test may detect overlapping distributions, resulting in a larger p-value.

Based on the analysis of the results, we can conclude that there is a statistically significant difference in execution times across all dimensions, with WebAssembly generally performing faster than Phaser across higher dimensions. The primary reason for WebAssembly's superior performance at higher dimensions appears to be the growth curve of average execution times. WebAssembly shows a more linear increase, whereas Phaser's performance time escalates more rapidly, particularly at larger maze sizes (Figure 1). Given this trend, it could be expected that the gap between Phaser and WebAssembly will continue to widen in favour of WebAssembly as dimensions increase.

In terms of performance variability, while Phaser is consistent with smaller mazes, its performance becomes increasingly variable for larger maze dimensions, as indicated by the greater spread and outliers in the boxplots from dimension 60x60 onwards. The strong positive skew in Phaser's execution times for higher dimensions suggests that a small number of outliers with longer execution times pulls the mean upwards. This indicates that Phaser occasionally takes significantly longer to perform in certain cases, contributing to the higher average. As maze complexity grows, Phaser may experience more frequent performance bottlenecks compared to WebAssembly. In contrast, WebAssembly's lower skew and more stable performance indicate that it is more predictable and less variable across test runs.

One limitation of the testing process was the reduced number of test runs for the 110x110, 130x130 and 150x150 dimensions, which was necessary due to time constraints. Another potential limitation are the possible differences in the implementations of the algorithms, as Phaser and WebAssembly use different programming languages, which may require a slightly different coding approaches, even though the underlying logic remains the same. Additionally, a potential constraint for WebAssembly is the use of a JavaScript function to reload timers for measuring execution time. However, any slowdown resulting from this appears to be negligible compared to measuring the time directly in C++ code.

The performance differences in large maze dimensions may also be attributed to how memory is managed in WebAssembly compared to JavaScript (Phaser). WebAssembly uses a linear memory model similar to lower-level languages, where memory is allocated in contiguous blocks, enabling faster memory access. In contrast, JavaScript relies on automatic garbage collection, which simplifies development but can lead to unpredictable performance drops during long-running tasks as indicated in Haas et al. (2017). As maze sizes increase, the added complexity may strain Phaser's memory management, resulting in slowdowns and greater variability in execution times for larger dimensions.

As a result, depending on the application's complexity and performance requirements, WebAssembly may be the preferred option for generating more complex structures, particularly when execution time is a critical factor. For simpler use cases, however, WebAssembly may not provide a significant speed advantage, making Phaser a more optimal choice due to its simplicity and lack of need for prior compilation.

Conclusion

In this paper, we conducted a comparison of procedural maze generation using WebAssembly and Phaser. The results demonstrate a statistically significant difference in execution times across various maze dimensions, with very low p-values providing strong evidence against the null hypothesis. While both technologies showed notable performance variations, Phaser outperformed WebAssembly up to dimension 60x60. On the other hand, WebAssembly outperformed Phaser from dimension 90x90 onwards, especially at larger maze dimensions. This trend can be attributed to WebAssembly's more linear growth in execution times, in contrast to the steeper increase observed in Phaser. Moreover, Phaser's performance became increasingly variable as maze complexity grew, indicating a greater likelihood of performance bottlenecks. Although limitations, such as a reduced number of test runs for larger dimensions and potential differences in algorithm implementations, may have influenced the results, the findings underscore the advantages of WebAssembly for generating complex structures, while Phaser might be preferable for simpler use cases.

Further research could explore the underlying causes of the observed performance differences, such as optimisation techniques or algorithmic complexity in Phaser versus WebAssembly. Additionally, performance comparisons across different browsers or analyses of memory complexity could provide deeper insights.

Recommendations

This paper is relevant for researchers, developers, and practitioners involved in the optimization of web technologies, game development, and procedural generation. It will be of interest to those exploring performance trade-offs between WebAssembly and JavaScript frameworks like Phaser, particularly in scenarios involving the generation of complex structures such as mazes. Additionally, the paper is suitable for those investigating performance optimization, as well as those interested in understanding the impact of different technologies on execution time and scalability in web-based applications. Developers working on applications requiring efficient real-time performance, especially those using WebAssembly or Phaser, will find the findings useful for selecting the most appropriate technology based on their specific use case and system requirements.

Scientific Ethics Declaration

* The authors declare that the scientific ethical and legal responsibility of this article published in EPSTEM Journal belongs to the authors.

Conflict of Interest

* The authors declare that they have no conflicts of interest

Funding

* This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Acknowledgements or Notes

* This article was presented as an oral presentation at the International Conference on Basic Sciences, Engineering and Technology (www.icbaset.net) held in Trabzon/Türkiye on May 01-04, 2025.

References

Battagline, R. (2019). *Hands-on game development with WebAssembly: Learn WebAssembly C++ programming by building a retro space game*. Packt Publishing Ltd.

Cagiltay, N. E. (2007). Teaching software engineering by means of computer-game development: Challenges and opportunities. *British Journal of Educational Technology, 38*(3), 405–415.

Faas, T. (2017). *An introduction to HTML5 game development with Phaser.js*. AK Peters/CRC Press.

Haas, A., Rossberg, A., Schuff, D. L., Titzer, B. L., Holman, M., Gohman, D., Wagner, L., Zakai, A., & Bastien, J. (2017). Bringing the web up to speed with WebAssembly. *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation* (pp. 185–200).

Herrera, D., Chen, H., Lavoie, E., & Hendren, L. (2018, March 14). *WebAssembly and JavaScript challenge: Numerical program performance using modern browser technologies and devices*. McGill University, Montreal: QC, Technical Report SABLE-TR-2018-2. Retrieved from http://www.sable.mcgill.ca/publications/techreports/2018-2/techrep.pdf

Macedo, J. D., Abreu, R., Pereira, R., & Saraiva, J. (2022). WebAssembly versus JavaScript: Energy and runtime performance. *Proceedings of the 2022 International Conference on ICT for Sustainability (ICT4S 2022)* (pp. 24–34).

Midžić, N. (2024). *Using C++ programming language in web application development.* (Bachelor thesis, Faculty of Organization and Informatics at University of Zagreb).

Mustonen, M. (2023). *Web based game engine design.* (Master thesis, School of Engineering Science, Information Technology at Lappeenranta–Lahti University of Technology).

Pavkov, S., Franković, I., & Hoić Božić, N. (2017). Comparison of game engines for serious games. *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (pp. 728–733). IEEE.

Reiser, M., & Bläser, L. (2017). Accelerate JavaScript applications by cross-compiling to WebAssembly. *Proceedings of the 9th ACM SIGPLAN International Workshop on Virtual Machines and Intermediate Languages* (pp. 10–17).

Wilson, K. N., Ghansah, B., Ananga, P., Oppong, S. O., Essibu, W. K., & Essibu, E. K. (2024). Exploring the efficacy of computer games as a pedagogical tool for teaching and learning programming: A systematic review. *Education and Information Technologies, 30*(4),4157-4184.

Wu, B., & Wang, A. I. (2012). A guideline for game development-based learning: A literature review. *International Journal of Computer Games Technology, 2012*(1), 103710.

Ylenius, S. (2020). *Mitigating JavaScript's overhead with WebAssembly*. (Master's thesis, Faculty of Information Technology and Communication Sciences at Tampere University).

Author(s) Information

Noa Midzic
University of Zagreb, Faculty of Organization and Informatics, Pavlinska 2, 42000 Varaždin, Croatia

Matija Novak
University of Zagreb, Faculty of Organization and Informatics, Pavlinska 2, 42000 Varaždin, Croatia
Contact e-mail: *matija.novak@foi.unizg.hr*

To cite this article

Midzic, N., & Novak, M. (2025). Performance comparison of WebAssembly and Phaser in procedural maze generation. *The Eurasia Proceedings of Science, Technology, Engineering and Mathematics (EPSTEM), 34*, 172-181.