Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186-5140
Volume 15, Number 1, pages 1–5, January 2026

A Case Study on Extending Rules of Games

Takahiro Imano
Grad. Sch. of Inf. Sci. and Arts
Toyo University
Kawagoe, Japan
s3b102400031@toyo.jp

Minoru Uehara
Grad. Sch. of Inf. Sci. and Arts
Toyo University
Kawagoe, Japan
Uehara@toyo.jp

***Abstract*—This study develops a mechanism that allows the flexible addition of new rules to board games and realizes a system that enables the easy creation of rule-extensible games. Previous studies have typically incorporated rules directly into existing game code, but this approach is dependent on game-specific implementations, resulting in a lack of versatility and reusability. This paper describes a method for constructing basic game code using the boardgame.io framework and describing additional rules with a minimum number of lines of code. For evaluation, we measure the number of lines in the rule section and the additional rule section, and determine that sections with fewer lines are easier to expand. As a result, we demonstrate that expansion is possible by changing only the rule logic without modifying the display content or user interface, achieveing high versatility and efficiency. This research facilitates the diverse expansion of board games and the prototyping of new rules, thereby expanding their potential applications.**

Keywords—Board game, checkers, game framework, rule extension

I. Introduction

This study develops a new method for expanding the rules of board games. The goal is to build a system that can implement these expanded games concisely, thereby enhancing the entertainment value of board games. Rule-expansion games are derivatives created by adding new elements to existing games without altering their gameplay. Examples of rule extensions include local rules and handicap settings. One app that can implement such games is Nintendo's "Asobi Taizen". Several games have diverse local rules that vary by region and player, meaning that the precise rules must be confirmed and agreed before starting the game [1]. Methods for extending the input/output interfaces and presentation aspects also exist [2]. However, other methods of rule expansion are scarce. Attempts to formally describe the structure and rules of board games have been made using a domain-specific language [3], while the description of rules by non-programmers can be supported using a knowledge base [4]. At present, however, there is no universal mechanism that enables flexible rule expansion by users.

A number of board games have well-known strategies and winning moves. Games such as 6×6 Reversi and checkers, for example, have been extensively analyzed, and knowledge of the game greatly influences the outcome [5]. Therefore, when new rules are added to existing board games, players are required to develop new strategies and ways of thinking, thereby revitalizing the game. Data-driven analyses have shown that the structure and complexity of board game rules are closely related to player satisfaction and the depth of strategy [6]. Based on this, the aim of this study is to create a framework that allows for the flexible design and introduction of additional rules pertaining to the game board, the game pieces, and the criteria for winning. Specifically, this study seeks to enhance the efficiency and reusability of rule design by enabling expansion components to be described concisely and explicitly without significantly altering the original rule structure. This research is significant because it provides an environment in which users can design and define rules themselves and play an expanded version of the game on the spot.

II. Related Studies

This study primarily focuses on extant works that expand the rules of board games and game creation tools.

A. Gyakuten Othellonia

The smartphone app "Gyakuten Othellonia" is based on the rules of Reversi with additional expansions [7]. The game is played on a 6×6 board, with gameplay following the rules of Reversi. In this game, each player selects 16 pieces from a pool of over 7,000 options, subsequently using the resulting deck to compete against opponents. Each element possesses distinctive statistics, including health, attack, skills, and combo skills. Skills are defined as individual abilities that are automatically activated upon the placement of a piece on the board, whereas combo skills are triggered by the execution of a pincer move. These elements serve to diversify the tactical options available in Reversi. In addition to the standard Reversi victory conditions, the game incorporates an additional objective: reducing the opponent's HP to 0. This results in two distinct victory approaches in Gyakuten Othellonia: board control and damaging the opponent's health. This creates a gameplay experience that is distinct from the simple territory-taking mechanics of traditional Reversi.

Research on the development of artificial intelligence (AI) for Gyakuten Othellonia has explored the optimal player actions using reinforcement learning and the optimization of skill activation timing [8]. These approaches diverge from conventional board game AI in that they necessitate multi-layered state evaluation and decision-making under uncertainty,

thereby establishing them as a novel and promising domain of research.

B. Game Development Tools

Examples of game creation tools include RPG Maker and Unity. RPG Maker is a software program designed for the creation of role-playing games (RPG). This software facilitates the creation of RPGs, even for users lacking programming expertise. Specifically, the software encompasses map data, effect data, event data, and other such components, which users can integrate to create original games. Game creation with RPG Maker resembles no-code development, because creators are not required to directly write event flags or conditional branches using a programming language. Thus, the difficulty of game creation is relatively low [9].

The Unity game engine includes integrated development environment functionality [10]. A salient feature of Unity is its capacity to integrate physics calculations, including gravity and collision dynamics, into game development. Endowing objects with the desired actions involves writing scripts using the C# programming language and attaching these scripts to objects to control their behavior. Furthermore, the Unity engine itself incorporates objects to be rendered, facilitating their placement and control. This methodology can be applied to the creation of games characterized by original rules. Another advantage is that games can be created and executed on any platform. However, a certain level of programming knowledge is required, and the need to write C# scripts presents a barrier to those with limited game development experience [11].

The game creation tools described above are designed to provide an environment for developing games, allowing players to construct original games from scratch. In contrast, the system developed in this study is specialized for extending existing games with different objectives. For this purpose, we wish to incorporate supplementary regulations into the relevant sections of the source code, while preserving the mechanics and overarching ruleset of the original game.

C. Gomoku/Renju

Gomoku is a two-player board game with simple rules: players take turns placing black and white stones on a board, and the first player to line up five stones in a row wins. Renju is an official competitive game derived from Gomoku, yet it incorporates a series of supplementary regulations designed to augment its competitiveness and fairness. In Renju, the actions of the first player (black) are restricted to prevent certain moves, such as san-san, tatsu-yon, and chōren. Failure to comply with these restrictions results in a loss for black [12]. Conversely, white players are permitted to move freely. This introduction of asymmetry in the rules of Renju is a deliberate strategy aimed at recalibrating the equilibrium of the game in relation to Gomoku.

Although Gomoku and Renju appear, at first glance, to be the same game, Renju is more sophisticated, with clearly defined victory conditions, restriction rules, and mechanisms in place to ensure fairness. In this study, we posit that such rule extensions are a related study and examine how the insertion of additional rules into a simple game contributes to new game design and strategic elements. In the specific instance of implementing the Renju code, 54 of the 145 lines of code used to implement Renju are allocated to the rule engine. Furthermore, three lines are incorporated as additional rules for Renju. The present study proposes a methodology for the facile creation of games with supplementary regulations by means of a simplified implementation of the additional rule sections, analogous to Renju.

D. Boardgame.io

Boardgame.io is an open-source framework for turn-based board games that supports Node.js environments and React-based user interface (UI) rendering [12]. The primary characteristic of this framework is its division of the game state from the context, thereby enabling game rules to be defined as pure functions. This approach enables logic design that is independent of the UI layer, thereby significantly enhancing reproducibility, maintainability, and testability. Furthermore, boardgame.io natively supports complex processes such as phase management, turns, AI opponents, serialization, and online synchronization, thus allowing the creation of sophisticated games without the need to implement these features from the beginning. In addition, boardgame.io employs a style that explicitly describes game state transitions, facilitating the local reflection of rule changes and extensions. This encourages flexible rule replacement and the addition of partial features, rendering it suitable for experimental approaches and academic verification in the domain of game design. Specifically, features such as moves, phases, end-if structures, and events facilitate the establishment of dynamic rules and turn control. In consideration of these structural advantages, boardgame.io is an effective platform for research on rule-extensible games. In this study, we employ boardgame.io to verify a simple method for adding rules to existing games.

III. Method

The objective of this study is to develop a system that facilitates the incorporation of additional rules into existing board games, thereby enabling the creation of rule-expansion-type games. The methods for achieving this objective are described below.

A. Rule Expansion Policy

The rule-extended game generation system proposed in this study preserves the rule structure of the original game while allowing flexible extensions to elements such as squares, pieces, actions, and victory conditions. The objective is to reduce the number of additional rules that developers must write, thereby creating a highly reusable and portable system. Initially, we conducted rule expansion trials based on checkers. In this section, we discuss the implementation of expansions, including the introduction of HP for pieces, warp hole effects, scoring based on cells, and victory determination based on the number of turns. After confirming the normal functionality of the expansions, each expansion logic is converted into a function and extracted as a string to an external file. This process establishes a structure in which rule management is independent of the game's main code.

B. Dynamic Loading of Rule Extensions using JSON

The initial approach used the JavaScript fetch API to load the rule extension logic from an external JSON file into the

existing checkers source code [14]. In this approach, additional rules are defined as strings, converted into JavaScript functions using fetch, and then inserted into the existing game logic. While this method has the advantage of allowing additional rules to be managed separately from the game itself, it also presents significant challenges. The original checkers code [15] is heavily dependent on the structure of the original game code. Therefore, it is important to understand the insertion points and context of the rules. Furthermore, the use of the fetch API necessitates an HTTP server environment, a factor that constrains its application in local environments. Furthermore, given the heterogeneity of game code structures, the implementation of extension rules in other games necessitates individualized adjustments, thereby compromising the extension's versatility.

C. Design using Boardgame.io

The method of directly editing the aforementioned free software relies on the structure of the original game code, which poses challenges in terms of the portability and reusability of rule extensions. To address this issue, we employ the boardgame.io framework and redesign the rule extension mechanism. This feature enables the explicit and localized understanding and editing of the game structure when adding or modifying rules. Moreover, given the framework's consistent provision of internal processing components, such as state management, there is no need to reconstruct the processing for each game. We use the structure of boardgame.io to organize the rule extension portion into functions within game.js, and design it to be activated only under specific conditions.

To illustrate the proposed extension method, we implemented a health system, configured special effects on specific squares, and incorporated victory conditions based on the number of turns for the game of checkers. The UI side (App.js) was unaltered, ensuring that the game's appearance and control system were unaffected, while only the logic was modified. This approach enables the use of a uniform screen design, irrespective of the presence or absence of extension rules. Consequently, it establishes an implementation environment in which extension rules can be appended or eliminated with autonomy. In the implementation using boardgame.io, an expansion-type game based on checkers was designed. The various expansions were described as independent modules within the game.js file on boardgame.io, with their application or deactivation selected via comment-out switching or function calls.

D. Evaluation Method

The efficacy of rule extensions is evaluated by focusing on the number of lines of code present in the rule definition and rule extension sections. Specifically, we consider the number of lines of code added to the base game rules relative to the number of lines of code in the original base game rules. This metric provides an objective indicator of the conciseness of the extension. The evaluation assesses the ease with which rule extensions can be implemented with minimal code. The evaluation of the methodology's practicality and versatility is achieved by assessing several aspects, including the clear separation of rule definitions and additional rules, the conciseness of the extension code, and the applicability of multiple rules simultaneously.

IV. Implementation

In this study, we implemented two methods for generating new rule-expansion games by incorporating additional rules into existing board games. One approach involved leveraging the fetch API to import rules from an external source into conventional JavaScript-based code, while the other involved implementation using the boardgame.io framework.

A. Dynamic Rule Insertion Method using Fetch API

Initially, we examined the existing source code of checkers, incorporating additional elements such as piece effects, square effects, and other pertinent rules. Corresponding code was then integrated to facilitate the implementation of these additional rules. These additional rules were incorporated directly into the game logic, and their functionality was validated. Subsequently, these processes were converted into functions and their contents were saved as strings in JSON files, which were formatted for reading from JavaScript files by means of the fetch API. Specifically, the code shown in Fig. 1 was defined in game.js and inserted into the rule extension section to generate a rule-extended game, function.(function name)(). The folder hierarchy is illustrated in Fig. 2.

```
function loadJson() {
    return fetch('extrarules.json')
        .then(response => response.json())
        .then(data => {
            // JSONから関数を復元する
            const s2f = (k, v) => (typeof v === 'string' && v.startsWith('function')) ? eval('(' + v + ')') : v;
            return JSON.parse(JSON.stringify(data), s2f);
        })
        .catch(error => console.error('Error loading JSON:', error));
}
```

Fig. 1. Fetch definition part.

```
Checkermaster/
├ index.html     … Screen layout (No change)
├ app.js         … Display/UI (No change)
├ game.js        … Game logics and rules (Insert additional rules)
├ style.css      … Design/layout (No change)
└ extrarules.json … Store additional rules as functions
```

Fig. 2. Folder hierarchy of fetch API method.

This method enables the management of rule extensions as independent JSON files, thereby facilitating the separation of game logic and extension rules. Furthermore, the extension of game.js facilitates the seamless execution of rule expansions. This substantiates the hypothesis that rule extension-type games can be generated dynamically.

B. Implementation Method using Boardgame.io

The conventional method using the fetch API exhibits limitations in terms of versatility. These limitations are exemplified by the need to comprehend the original code when inserting rules. The method using boardgame.io was executed on Google Colab, with the board game environment constructed

using HTML, JavaScript, and React components. The folder hierarchy is delineated in Fig. 3.

```
checker/
├── index.html        ← The layout of the display
└── src/
    ├ App.js          ← Game engine startup and display control
    └ Game.js         ← Game logics and rules (Insert additional rules)
```

Fig. 3. Folder hierarchy of boardgame.io method.

Because all rule definitions are consolidated within game.js, it is possible to create an environment in which new rules can be easily introduced by editing game.js alone. Examples of such extensions include the implementation of a health system, warp zones, turn management, and victory conditions, all of which can be achieved with just a few lines of additional code. This indicates that the design of rule-extensible games can be both simple and flexible. The implementation method using boardgame.io confers several advantages, primarily the clear separation of game logic from the UI. This separation facilitates both visual and functional comprehension of the code structure, thereby enhancing its overall intelligibility. Moreover, it enables localized management of extension components, a feature that will be particularly advantageous in complex, multifaceted projects. In this study, two methods were implemented, and it was confirmed that rule expansion is possible with minimal changes to the core game components. Specifically, the implementation using boardgame.io achieved a highly versatile and maintainable structure, establishing an environment in which expansion rules can be efficiently inserted and verified.

V. Evaluation

The evaluation of the proposed method explores the extent to which lines of code are added to the rule extension section in comparison to the rule definition section. In this section, the rule definition and rule extension sections are defined. The rule definition section delineates the fundamental rules that are prerequisites for the game to function properly prior to the extension, including player actions, turn-based gameplay, and victory conditions. The UI and other sections are not included in the evaluation. Conversely, the rule extension component pertains to the code incorporated into the logic associated with the extension elements, as introduced in this study, including piece effects, space effects, and alterations to the victory conditions. In the case of Renju, the rule definition section comprises 54 lines, while the prohibited move section consists of three lines, approximately 2% of the total. Table I summarizes the number of lines of code required to modify the checkers game using the fetch API and boardgame.io.

TABLE I. Number of Code Lines Required for Each Implementation.

	Checkermaster	***Boardgame.io***
Base code (rule part)	315 lines	262 lines
HP concept	8 lines	5 lines
Warp zone	6 lines	7 lines
Turn count + End condition	12 lines	9 lines

The results in Table I demonstrate that the effort required for extension is minimal in both methods, and that it is straightforward to separate existing rules and add extensions. In addition, the extensions can be completed without modifying the UI or screen configuration files. This observation indicates that the benefits of structural separation are successfully implemented in the design.

VI. Conclusion

This study explored a method for enhancing the diversity and entertainment value of existing competitive board games by developing a mechanism that facilitates the concise addition of novel rules. The dynamic loading of rules from external JSON files was examined using JavaScript's fetch API and eval function, with a focus on components such as squares, pieces, actions, and victory conditions. However, this approach was found to be dependent on the original code structure, exhibiting a lack of flexibility. To address this issue, the boardgame.io framework was introduced, and the system was redesigned with a structure that separates the game logic from the user interface. The rule definitions were centralized in game.js, leaving the UI untouched. This approach resulted in a flexible environment that could be expanded with just a few lines of additional code. We implemented extensions to the checkers game, incorporating features such as health points, designated squares, and victory conditions contingent on the number of turns. The proportion of code required for each extension rule was found to be less than 5% of the total. Furthermore, the clear separation between rule definitions and extensions confirmed the ease of rule expansion. However, challenges such as rule interference, the programming knowledge required for implementation, and scalability to other games were identified. In summary, the present study contributes to the realization of rule-extensible game generation.

References

[1] J. Sugiura, "Applying card games to understand social problems," Stud. Simul. Gaming, vol. 24, no. 1, pp. 11–21, 2016. DOI: 10.32165/jasag.24.1_11.

[2] J. S. Malliaros et al., "Electronic augmentation of traditional board games," Int. Conf. Adv. Comput. Entertain. Technol. (ACE), 2009.

[3] T. Schroten, "Introducing BGD: A DSL to express board games and gameplay," Bachelor's thesis, University of Twente, Netherlands, 2015. [Online]. Available: https://essay.utwente.nl/79157/1/Schroten_BA_ewi.pdf

[4] M. Exman and I. Alfia, "Knowledge-driven game design by non-programmers," 2014, *arXiv:1404.4713*. [Online]. Available: https://arxiv.org/abs/1404.4713

[5] J. Schaeffer et al., "Checkers Is Solved," Science, vol. 317, pp. 1518–1522, 2007. DOI: 10.1126/science.1144079

[6] S. Samarasinghe et al., "A data-driven review of board game design and interactions of their mechanics," IEEE Access, vol. 9, pp. 114051–114069, 2021.

[7] DeNA Co., Ltd., "How to play | Gyakuten Othellonia official site | Dramatic Reversal Battle," [Online]. Available: https://othellonia.com/about/howto. [Accessed: Aug. 19, 2025].

[8] K. Ōwatari et al., "Top-level battle AI for the modern mobile game 'Gyakuten Othellonia' using reinforcement learning," Proc. 37th Annu. Conf. Japanese Soc. Artif. Intell. (JSAI), p. 2Q4OS27b05, 2023 (in Japanese).

[9] Enterbrain, "RPG Tsukūru (RPG Maker)," 2004. [Online]. Available: http://tkool.jp/. [Accessed: Aug. 23, 2025].

[10] Unity Technologies, "Unity 3," 2006. [Online]. Available: http://unity3d.com/. [Accessed: Aug. 23, 2025].

[11] ECMA, "Standard ECMA-334: C# Language Specification, 4th edition," 2006. [Online]. Available: https://www.ecma-international.org/publications-and-standards/standards/ecma-334/. [Accessed: Aug. 23, 2025].

[12] Kyoto Renju Association, "What is Renju?" (in Japanese). [Online]. Available: https://www.kyogo.org/contents/kouza/river01/river01_01.html. [Accessed: Aug. 23, 2025].

[13] Boardgame.io, "Turn-based game framework for the web." [Online]. Available: https://boardgame.io/. [Accessed: Aug. 23, 2025].

[14] BreezeGroup, "How to use the fetch API in JavaScript," (in Japanese), Apr. 2020. [Online]. Available: https://breezegroup.co.jp/202004/javascript-fetch/. [Accessed: Aug. 23, 2025].

[15] Codethejason, "Checkers," GitHub repository. [Online]. Available: https://github.com/codethejason/checkers. [Accessed: Aug. 23, 2025].