Prototypical – A Framework for Board Game Creation

Vincente Campisi, Helmut Hlavacs, Patrick Pazour and Daniel Martinek

University of Vienna, Research Group Education, Didactics and Entertainment Computing, Vienna, Austria, helmut.hlavacs@univie.ac.at

Abstract. Despite an iterative workflow and common characteristics, board game design differs from its video game counterpart in the tasks and frequency with which digital technologies are leveraged. This study presents "Prototypical", a software application explicitly intended to support the board game design process. "Prototypical" provides modelling, simulation and analysis features for prototyping and developing individual board game mechanics. Three examples demonstrate its current capabilities and the performance of the Monte-Carlo Tree Search artificial intelligence agent, while a user study involving five participants and the System Usability Scale survey identifies the need for user interface improvements.

1 Introduction

With the increasing popularity and commercial success of video games [1], the terms game design and game development are often used in the context of software. There is an abundance of choice when it comes to technologies meant to support the creation of digital games, ranging from game engines such as Unity [2] and Unreal Engine [3] to digital asset creation software such as Blender [4]. As inherently digital media, video games require the use of software during development. When it comes to commercial board games, however, the opposite seems true – while similarly iterative and ludological, board game design relies on paper, human playtesters and analog methods [5].

This research explores how a software application might assist the board game designer during the development process. "Prototypical" supports features to model, simulate, inspect and potentially balance mechanics of a design during the early prototyping phase. This is accomplished through the input of information about game components, component state and interactions, success conditions and simulation parameters. Simulations are performed and the resulting data can be inspected and visualized through generated charts. This work aims to assess the potential of these features through several examples and a user study.

2 Related work

When discussing the game development process, fundamental characteristics inherent to games of all types should be acknowledged. [6] provided an abstract

analysis of games and presented a rules typology which applies to all kinds. Reasons for playtesting were identified and a link between game rules and gameplay was described. [7] provided examples of common playtesting approaches for games and ways to recognize and address gameplay imbalances. The Game Experience Questionnaire was seen to be effective for measuring user experience in digital games [8] and similar efforts include Riot Games' use of quantitative game data to optimize player experience [9].

The intersection of artificial intelligence and games led to a significant body of research. Objective Monte-Carlo [10] incorporated move-selection and backpropagation improvements into the existing software Mango [11] and Monte-Carlo Tree Search (MCTS) was originally proposed as a general game-playing artificial intelligence framework [12]. An MCTS agent for the full rule set of "7 Wonders" performed better than a standard rule-based AI agent [13] and performance of MCTS agents was compared for the game "Ticket to Ride" [14]. Several improvements found in "Prototypical", such as discretization and chance events, were based on this work.

As the complexity and performance of game-playing AI agents grew, so did attempts to incorporate them into software and the game design process. Building on previous projects involving AI and the game of Go, [15] presented an open-source software framework to develop full-information two-player board game agents using game-independent MCTS. [16] proposed seven different strategies complementary to playtest data in which AI and visualization techniques might help a game designer to extract knowledge from a game design. It was shown that game traces can be used for tracking game state information, debugging and improving designer understanding a given design.

[17] studied the card game "Dominion" and showed that specific cards contribute to the perceived game balance, regardless of player strategy. [18] designed and implemented a digital card game, using an AI agent to playtest and progressively modify game components. [19] used an AI agent-based approach with A* and MCTS to model four distinct playing styles of the game Ticket to Ride, identifying two game states not covered by the game rules and demonstrating that small modifications to game entities could drastically effect strategies and gameplay. [20] recognized the possibility of a future "robust system that aids modern board game designers".

An overview of several software applications which can be used in the context of board game development will now be provided. Some entries may be used for more than one objective, but it is helpful to group them according to those which support the game development process or assist playing digital game implementations and those which do both.

The Machinations framework offers a web-based visual programming language to generate game simulation flow charts and view related information. It does not require programming knowledge and is Turing-complete, utilizing nine node types and two connection types for its economy-based flowchart perspective [21]. Specific game parameters can be monitored during random or Monte-Carlo-based simulations using generated charts (e.g. histogram).

Boardgame Lab allows the user to prototype and playtest tabletop games through three main focus areas of prototyping, playtesting and automation [22]. Iterary [23] provides basic prototyping tools and allows the user to model limited game mechanics and simulate results. Tabletopia [24] is a virtual tabletop system without rules enforcement which provides tools to assist in designing, playing and publishing board games using a graphical interface. Tabletop Simulator [25] is similar but rendered in a 3D environment and can be easily used for prototyping [26]. Vassal [27] and ZunTzu [28] are tabletop game engines, popular for playing community-made digital implementations of physical games. Both applications facilitate play of digital versions of existing physical games but can be potentially used for digital prototypes. BoardGameGeek community members discussed applications to assist creation of physical board game components for early prototypes [29]. nanDECK [30] and Squib [31] aim to speed up the design and creation process of physical cards.

3 Prototypical – A Board Game Development Framework

Without requiring programming knowledge, "Prototypical" can help the user model and simulate individual game mechanics by declaring game components, actions, conditions and simulation parameters. The corresponding objects are known in the software as *components*, *actions*, *conditions* and *execution contexts*, respectively. This object-based approach to representing game elements is facilitated by a graphical user interface. Similar to lambda expressions in programming, some objects have variable bindings evaluated only during gameplay simulations.

The user can view intermediate game states and generate graphs to visualize changes in property values or expressions over time for one or multiple simulations. Adjustments to modeled components and behavior can be made "on the fly" to allow the game designer to iteratively test and revise assumptions, as done in conventional development and playtesting. Simulations are referred to as *executions* and either a random or a heuristic UCT-based Monte-Carlo tree search agent can provide decision-making. An *execution* yields a result set of game states called *frames*, where each contains the game state before and after a gameplay event has taken place.

3.1 Examples

Three examples were implemented to demonstrate capabilities of the software. "Movement Points" covers a proposed card-based mechanic for determining player resources. "Polis: Rise of the City State" focuses on the combat mechanic from a board game currently in active development. The last example uses the game of "Tic-tac-toe" to demonstrate AI agent performance and potential support in the software for complete rule sets. It should be noted that words corresponding to entities from "Prototypical" will be italicized and literal values will be in bold font.

Movement Points This proposed game mechanic determined a player movement resource each turn through card values. A deck contained eighteen cards, each with a **move** value according to the initial distribution in Table 1. Each player randomly drew a card, allocating the corresponding number of movement points. Drawn cards were not added back into the deck and gameplay ended once the deck was empty. This example involved modeling the necessary entities, running simulations, making design modifications and iterating until a satisfactory result was achieved. Thirty *executions* were required in total, with modifications made after the first and second sets of ten *executions*. There was one possible action per player each turn when determining movement score and the Random AI agent was used for decision-making.

Table 1: Initial distribution of **move** values (Movement Points)

move Value	Quantity
0	6
1	6
2	6

Polis: Rise of the City State In "Polis: Rise of the City State", players research technologies, construct buildings and leverage military might to conquer territories in the hopes of founding the best empire, measured by victory points. Military power is represented using meeples, while technology and buildings take the form of cards. Regions exist in a shared location at the center of the game board and combat occurs when meeples belonging to both players have been placed in a shared region.

In the example, each player selected a card from the corresponding hand and the **vp** value of the card became a combat modifier. The modifier was added to the respective player meeple **count** property, resulting in a total score where the highest was the winner. Rounds proceeded until no cards remained in the draw deck.

According to the designer, the combat system had not presented itself as a dynamic or interesting user experience in previous playtesting sessions. A sense of thrill was to be achieved by iteratively modifying the composition of the card deck until the simulation data pointed towards greater variation of observed combat outcomes. The range of the difference between player scores was permitted to increase between turns but the average difference across all simulations was permitted to change only slightly. Described differently, the outcomes from one combat round to the next were expected become more disparate, but neither player should have gained an inherent asymmetrical advantage due to the modifications. The **vp** property of each card was set according to the initial distribution outlined in Figure 1 (provided by the designer) and fifty simulations were carried out iteratively in sets of ten, where component changes were made at each interval and the generated charts referenced.

Combat			
Value		Copies	Deck %
1	17	38	39.58
2	10	26	27.08
3	9	23	23.96
4	4	9	9.38
		96	

Fig. 1: Original card value distribution in Polis

Tic-tac-toe Tic-tac-toe was implemented to demonstrate that the software can support complete rule-sets and to observe performance of the MCTS agent. The 3x3 grid was modeled such that location numbers increased from left to right and top to bottom, as seen in Table 2. No iterative changes were made to the design itself. This example used a known game design to demonstrate the performance of the MCTS agent so that sensible gameplay moves and poor strategic choices were easily identifiable during state inspection. All necessary elements were modeled in and *executions* were performed with MCTS agents playing against each another. Performance was observed by modifying the MCTS iteration count parameter and recording changes in winning player results, where draw outcomes would indicated consistent decision-making performance. Sixty

Table 2: Grid layout of 3x3 Tic-tac-toe board

field 0	*field* 1	*field* 2
field 3	*field* 4	*field* 5
field 6	*field* 7	*field* 8

executions were run in total, each corresponding to a specific value of the MCTS iterations parameter, also referred to as playouts or playout count. The *execution* count per set varied. The winning player or draw result was summed across data sets and playout parameter count. Values of 50, 500 and 1000 were used, where twenty *executions* were performed for each. The twenty which made use of 50 playouts were broken into two sets of ten sequential *executions*, while those using 500 playouts were divided into one set of ten and two sets of five. For the 1000 playouts set, two sets of ten were again used.

3.2 User Trials

A user study was conducted to measure the user experience of the software interface. The goal was to present and explain features and estimate its utility through participant interviews. Another purpose was to solicit feedback and considerations for future work. The target demographic was individuals who identified as board game designers or players. For the latter, the individual willingly participated in a board game with others, whether regularly or occasionally. The first

implies a frequency of once or more per month, while the latter indicates at least once per year.

Five sessions took place individually over the course of two weeks. Each began with a short explanation about the intended use of Prototypical, followed by a demonstration of the example from Section 3.1. The user was instructed to explore the software features and ask questions. It was then requested that the example was modified towards what the participant considered a more-balanced or interesting design. Some users created additional cards, while others only modified component property values. All participants asked questions and required additional instruction.

To qualitatively measure these experiences, users filled out the System Usability Scale (SUS) survey. The SUS is a Likert scale with ten statements, meant to gauge subjectively the usability of a given system. Answers range from one to five (inclusive), where one indicates that the participant strongly disagrees with the given statement and a score of five indicates that the user strongly agrees. The survey was presented electronically without a time limit and the results were translated into scores out of 100 using [32].

3.3 Hardware and Software

Examples were implemented on an Apple MacBook Pro 2019 laptop with Intel i7 2.8 GHz Quad-Core processor, 16 GB of 2133 MHz DDR3 RAM, Intel Iris Plus Graphics 655 and Mac OS 11.4. The IntelliJ (2020.1.1-2021.1.1) IDE with relevant plugins was used for both development and testing, while the web application itself was run using the Chrome (83.0-93.0) browser.

4 Results and Discussion

The software was successfully used to model game concepts and simulate playtesting of the example mechanics without human participants. It visualized subsets of the simulation data and displayed changes in game state parameters and potential trends. Both individual mechanics and a full simple rule set for two-player games were implemented, showing flexibility in how the software could be applied.

The software appeared potentially useful to game designers by providing new ways to view game state information, such as inspection and tracking of individual properties throughout simulated gameplay. This type of comparative data is expected to inform the designer during development, making obvious which design changes caused large observable differences in the profile of the charted property. With the expectation that more information promotes improved decision-making, the utility of Prototypical in this context is evident. There was also a possibility to partially replace the initial need for human playtesters, saving resources in the early stages of development.

Table 3 outlines the number of objects created when modeling the mechanic(s) of each example. “Movement Points” required the fewest and was the

simplest mechanic of the three to implement. The second example required the most *components* but the fewest *conditions*, needing only one which was always met and another for tracking the current round number. It is unsurprising that "Tic-tac-toe" used the most *conditions*, as it contained the entire rule set of the game and *conditions* were necessary for every potential winning board state per player (e.g. *fields* **0**, **1** and **2** containing the same player value). While up

Table 3: Object counts required to implement each example

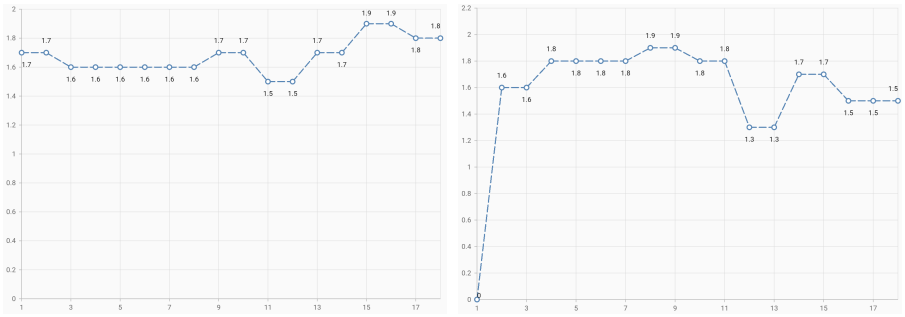
	component	*action*	*condition*	Total
Movement Points	23	10	3	36
Polis: Rise of the City State	104	15	2	121
Tic-tac-toe	12	18	77	107

to fifty simulations were taken into consideration when analyzing results, there is no indication that this adequately sampled the game space of the first two mechanics. It is possible that results and assumptions in these examples did not reflect the normal outcomes of the games, until compared to real playtest data. Additionally, the user study demonstrated that the current user interface is inadequate for practical use.

4.1 Movement Points

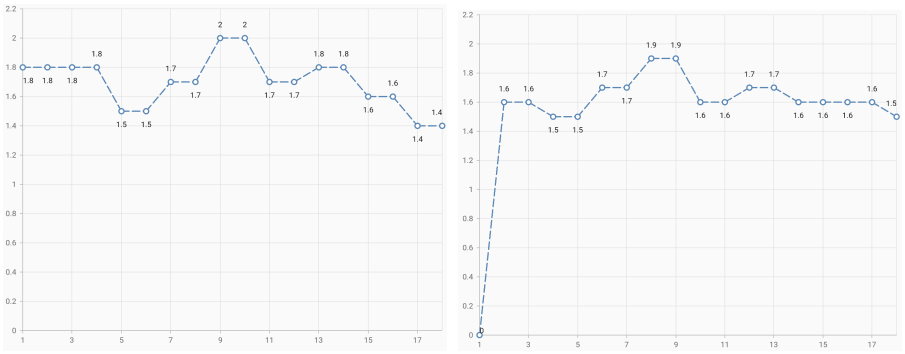
A chart was generated to display the average movement score of the players for the first set of simulations. The values seen in Figures 2 using the initial distribution were deemed too low and modifications to the card values were made. Six cards with a value of **0** were changed to **2**. Ten more *executions* were performed and corresponding charts created. While the charted average score at this interval displayed five directional changes and a larger range of values, it still appeared to remain the same or very close to the preceding or subsequent value across significant segments of the *execution*. These characteristics can be observed through the shape of the line charts from Figures 3. Six card values of **3** were changed to **5**. Updated charts were again generated but this time the displayed trends were sufficient in both range and shape. Both first and second player charted average score values displayed six directional changes, which was two more than the originals. These plotted average move scores had been noticeably altered, as seen in Figures 4, done with the expectation that these changes would be observable in user experience and gameplay of the encompassing game. It is expected that a dynamic line chart would yield correspondingly dynamic gameplay possibilities.

This mechanic was suitable for Prototypical because of low rules complexity and component count. Gameplay-altering changes were easily made and modifying few values led to noticeable effects on game state. Changes involved updating a single *property*, which made it simple to view both the values of the **move** *property* for each *frame* of an *execution result* data set, but also across multiples. The



(a) Player 1 (b) Player 2

Fig. 2: Movement Points average score frames 1-10



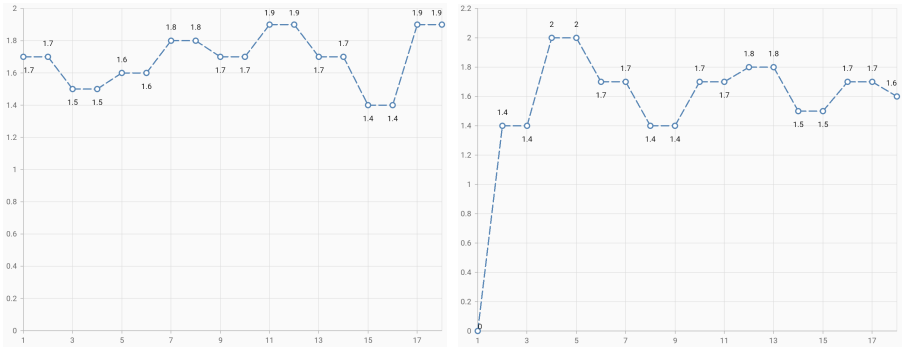
(a) Player 1 (b) Player 2

Fig. 3: Movement Points average score frames 11-20

average was displayed on a line graph and could be superimposed on other data. The visualized *properties* demonstrated that design changes had taken effect, but did not clarify to what extent. The example showed that Prototypical could assist the designer make observations and hypotheses about the current state of a game design but it could not guide the development process (e.g. suggest potential changes). A clearer characterization of the direct relationship between objective simulation data and the subjective user experience is desired.

4.2 Polis

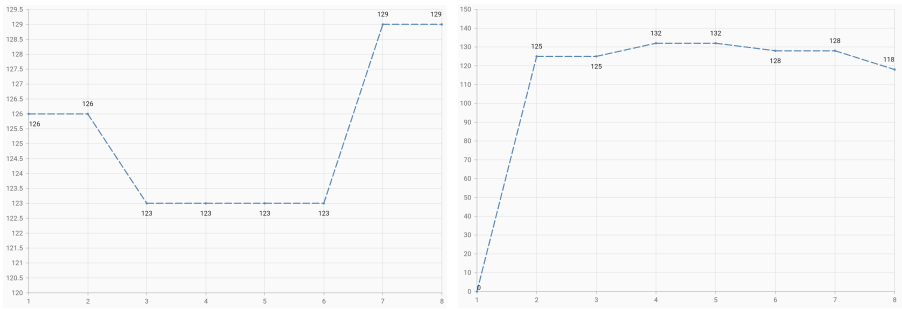
Ten *executions* were performed at the start, as seen in Figures 5. Fifteen **vp** values were changed from **1** to **0** to lower the minimum possible score. After ten additional simulations, this was estimated to be too excessive and was partially reverted by changing ten card values from **0** back to **1**. This was done because the range of observed values was reduced such that the gameplay impact of the



(a) Player 1 (b) Player 2

Fig. 4: Movement Points average score frames 21-30

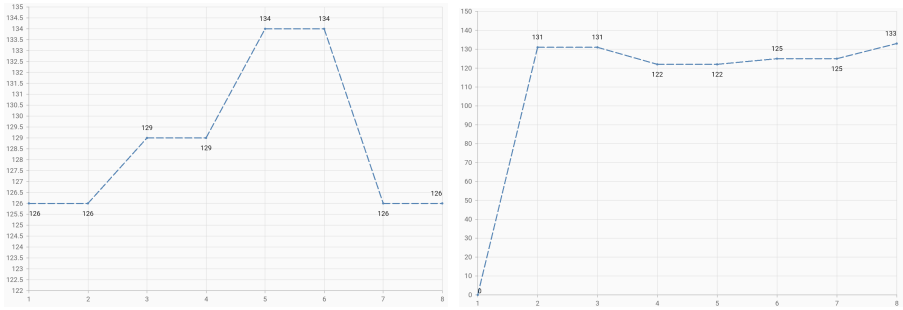
combat mechanic became questionable. Thirteen Cards were also changed from **2** to **3** to increase the probability of obtaining a higher sum. Ten further *executions* led to observed results centered too closely on mid-range values, so eight card values were changed from **3** to **4**. A higher upper bound was once again desired, so four card values were changed from **4** to **5**, followed by ten *executions* and the final changes of two card values from **4** to **7**. The final values can be seen in Figures 6. The range of observed values for the first player summed score increased by two from six in the first *execution* set to eight in the last. The change was even more pronounced for the second player, increasing by four from an initial range of seven to a final observed range of eleven.



(a) Player 1 (b) Player 2

Fig. 5: Polis score sum frames 1-10

The component modifications in this example did not significantly change the difference between observed scores summed across multiple *executions*. This



(a) Player 1 (b) Player 2

Fig. 6: Polis score sum frames 41-50

value in first and second player scores summed at each *frame* for *executions* 1 through 10 was 125.25 and 128.33, respectively. The final ten (41-50) yielded a difference of 2.75. This fulfills the initial condition that no modifications may significantly increase the discrepancy in opposing scores. The generated charts show that it is possible to unify multiple game state properties and associate them with a single visualization. The charts were used to display the total score of each player at each *frame* and for multiple *executions*. However, the combat score itself was comprised of a combination of the meeple count and card victory point value, demonstrating that it is possible to simultaneously track multiple elements of game state. This functionality was desired by the designer and was the motivation for suggesting this example.

4.3 Tic-tac-toe

MCTS AI agent performance benchmark data, including the respective mean time per playout set, is seen in Table 4. All ended with a draw result and end game states from ten *executions* were randomly observed without anomalies or rules errors. The playout values of 50, 500 and 1000 were chosen to explore the relationship between higher playout values and superior performance. The latter comes at the expense of time and computing resources, where values less than 50 produced unsatisfactory results. The chosen values were expected to estimate performance for normal use cases, balancing performance and resource costs, and no specific boundaries for these parameters were discovered or otherwise tested.

While optimal moves were not used as a benchmark, the random sampling of performed moves did not reveal obviously strategically sub-par choices which would indicate poor performance. Additionally, the results from Table 4 indicated consistent decision-making behavior. Inconsistent win rates would have indicated unpredictability from one or both opposing agents, implying an incorrect implementation [33]. Performance differences at these levels in the benchmarks were not observed but the *execution* run time required increased linearly with the configured number of MCTS iterations. This was expected, as higher

decision-making performance requires more iterations per selected move, resulting in longer computing time.

This example showed that Prototypical can potentially support a full rule set and simulate an entire game using AI agents. The simplicity of the game and the small board size made it random inspection of moves possible. It was not, however, a useful demonstration of Prototypical during the development process because the aim was to demonstrate modelling capabilities of the software and its AI agent performance.

Table 4: Tic-tac-toe win rates and execution time benchmarks.

Pl.	Draw	P1 Win	P2 Win	Qty. E	T(s)	T/E (s)
50	20	0	0	20	150.07	7.50
500	20	0	0	20	266.73	13.34
1000	20	0	0	20	515.13	25.76

4.4 User Trials

The results of the survey can be seen in Table 5. The median score was 62.5, where the overall range was between 50 and 75, inclusive. The mean score was 61.5, which indicates that the usability of the current Prototypical interface is "poor" (see Figure 7).

The user study and SUS surveys revealed both successes and failures of the user interface. The time required to explain the premise and functionality of the software was negligible, implying that its novel approach to modelling is intuitive. While participants asked questions throughout the trial, no assessments of comprehension were made. The results of the survey itself indicated deficits in the UI and the low average score implies that improvements must be made before the software can be used in a practical context. Other areas of concern are the ineffective handling of games with imperfect information and stochastic events. While chance nodes handle select situations well, The inability to comprehensively handle imperfect information limits the potentially supported games.

Score	Rating
> 80.3	A / Excellent
68 – 80.3	B / Good
68	C / Okay
51 – 68	D / Poor
< 51	F / Awful

Fig. 7: System Usability Scale Rubric

Table 5: System Usability Scale scores from five participants.

Score A	Score B	Score C	Score D	Score E
65	55	62.5	50	75

5 Conclusion

This work investigates how the "Prototypical" software application might assist the board game designer in the context of an iterative development process. It is a board game development framework with a graphical user interface, supporting modelling, simulation and analysis of game mechanics. The examples from Section 4 demonstrated several features and revealed inadequacies of the user interface.

The "Movement Points" and "Polis: Rise of the City State" examples showed how a simple card-based resource mechanic could be modified across several iterations and it was clear from generated charts how the movement or score properties evolved throughout the performed simulations with respect to the implemented changes. Differences in the observed values or ranges was evident through the shape of the generated line charts, providing the developer with the ability to track and visualize evolving elements of game state.

The final example further demonstrated flexibility of the software through its ability to support complete rule sets. It measured the performance of the UCT-based MCTS AI agent, which was found to be consistent for varied MCTS loop parameter values. Random inspection of moves did not reveal strategically unsound choices and the chosen moves resembled the skill of a human player.

Although using the software appeared intuitive for the study participants, the complexity was higher than desired. Future efforts have the potential to fix these issues and implement additional features, making it a valuable tool for developing board game mechanics. Increased player counts, additional game types, improved UI workflows and stability improvements are possible focus areas. "Prototypical" controls a small set of features but demonstrates potential to iteratively assist the designer to inspect simulated playtest data and search for trends in evolving game state. At the very least, it serves as inspiration for what a system to aid modern board game designers might provide.

References

1. Videogames are a bigger industry than movies and north american sports combined, thanks to the pandemic, https://www.marketwatch.com/story/videogames-are-a-bigger- industry-than-sports-and-movies-combined-thanks-to-the- pandemic-11608654990, accessed: 2022-06-03 (2020).
2. Unity real-time development platform — 3d, 2d vr & ar engine, https://unity.com/, accessed: 2022-02-28 (2022).
3. The most powerful real-time 3d creation tool - unreal engine, https://www.unrealengine.com/en-US/, accessed: 2022-02-28 (2022).

4. blender.org - home of the blender project - free and open 3d creation software, https://blender.org/, accessed: 2022-03-27 (2022).
5. J. Schell, Chapter seven - the game improves through iteration, in: J. Schell (Ed.), The Art of Game Design, Morgan Kaufmann, Boston, 2008, pp. 75–95. doi:https://doi.org/10.1016/B978-0-12-369496-6.00007-7.
URL https://www.sciencedirect.com/science/article/pii/ B9780123694966000077
6. A. Järvinen, Making and breaking games: a typology of rules., in: DiGRA Conference, 2003, pp. 68–79.
7. A. B. Jaffe, Understanding game balance with quantitative methods, Ph.D. thesis, University of Washington (2013).
8. J. Barbara, Measuring user experience in board games, International Journal of Gaming and Computer-Mediated Simulations (IJGCMS) 6 (1) (2014) 64–79.
9. Riot games launches player dynamics to help improve multiplayer experiences, https://venturebeat.com/2020/03/11/riot-games-launches- player-dynamics-to-help-improve-multiplayer-experiences/, accessed: 2021-10-02 (2022).
10. G. Chaslot, J.-T. Saito, B. Bouzy, J. Uiterwijk, H. J. Van Den Herik, Monte-carlo strategies for computer go, in: Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium, 2006, pp. 83–91.
11. Mango, https://project.dke.maastrichtuniversity.nl/games/go4go/mango/index.htm, accessed: 2021-10-03 (2021).
12. G. Chaslot, S. Bakkes, I. Szita, P. Spronck, Monte-carlo tree search: A new framework for game ai., AIIDE 8 (2008) 216–217.
13. D. Robilliard, C. Fonlupt, F. Teytaud, Monte-carlo tree search for the game of "7 wonders", in: Workshop on Computer Games, Springer, 2014, pp. 64–77.
14. C. Huchler, An mcts agent for ticket to ride, Master's th., Maastricht Univ. (2015).
15. M. Enzenberger, et al., Fuego—an open-source framework for board games and go engine based on monte carlo tree search, IEEE Transactions on Computational Intelligence and AI in Games 2 (4) (2010) 259–270.
16. M. J. Nelson, Game metrics without players: Strategies for understanding game artifacts, in: Workshops at the Seventh Artificial Intelligence and Interactive Digital Entertainment Conference, 2011, pp. 14–18.
17. T. Mahlmann, J. Togelius, G. N. Yannakakis, Evolving card sets towards balancing dominion, in: 2012 IEEE Congress on Evolutionary Comp., IEEE, 2012, pp. 1–8.
18. J. Krucher, Algorithmically balancing a collectible card game, Bachelor's thesis. ETH Zurich (2015).
19. F. de Mesentier Silva, S. Lee, J. Togelius, A. Nealen, Ai-based playtesting of contemporary board games, in: Proceedings of the 12th International Conference on the Foundations of Digital Games, 2017, pp. 1–10.
20. F. de Mesentier Silva, S. Lee, J. Togelius, A. Nealen, Ai as evaluator: Search driven playtesting of modern board games., in: AAAI Workshops, 2017, pp. 959–966.
21. Machinations.io, https://machinations.io/, accessed: 2021-10-02 (2021).
22. Boardgame lab, https://boardgamelab.app/, accessed: 2021-10-02 (2021).
23. Iterary, https://www.iterary.com/, accessed: 2021-10-02 (2021).
24. Tabletopia, https://tabletopia.com/, accessed: 2021-10-02 (2021).
25. Tabletop simulator, https://www.tabletopsimulator.com/ (2021).
26. How to make a tabletop simulator demo of your board game, https://brandonthegamedev.com/how-to-make-a-tabletop-simulator- demo-of-your-board-game/, accessed: 2021-10-02 (2020).
27. Vassal, https://vassalengine.org/, accessed: 2021-10-02 (2021).
28. Zuntzu, https://www.zuntzu.com/, accessed: 2021-10-02 (2021).

29. Early prototype software, https://boardgamegeek.com/thread/ 1982913/early-prototype-software, accessed: 2021-10-02 (2021).
30. nandeck, http://www.nandeck.com/, accessed: 2021-10-02 (2021).
31. Squib, https://squib.rocks/, accessed: 2021-10-02 (2021).
32. Sus calculator, https://uiuxtrend.com/sus-calculator/ (2022).
33. Monte carlo tree search for tic-tac-toe game in java, https://www.baeldung.com/java-monte-carlo-tree-search (2020).