BoardBench: Translating Board Game Rulebooks into Executable Environments

Bachelor Thesis in Computer Science

Ben Tischberger



Tübingen AI Center
tuebingen.ai

University of Tübingen

21.09.2026

Abstract

This thesis looks at which problems appear when a large language model translates a board game rulebook into an executable Python environment and how these problems change when a new generation receives additional explanations. This translation is difficult because rulebooks are written for people and can leave rules linguistically unclear, but a program needs clear rules for legal actions, state changes and the end of a game.

This thesis develops BoardBench, a workflow for generating and then evaluating these game environments, and applies it to six games from their German publisher rulebooks. Rule scenarios are tests of expected game behavior that are prepared from the rulebooks before any code is generated. Some of these tests follow a rule that the rulebook states clearly, while others follow a decision that was made for this study when the rulebook left a situation open. A coding agent generates each environment without access to these scenarios. Technical checks, random play, interface checks, rule scenarios and model reviews are then reported separately. In selected later runs, the model receives the original inputs together with an extra file that explains a gap in the rulebook or repeats a clear rule that an earlier implementation missed. The model then generates a new environment instead of modifying the earlier code.

All final implementations passed the technical checks, but rule errors were found in all six games, including incorrect setups, missing legal actions and scoring errors. Some later implementations corrected targeted problems but kept other errors or introduced new ones. Because each later run generated a new environment, different implementation results can come from the extra explanations or from the model's stochastic behavior. Some evaluator expectations and tests were also incorrect and had to be corrected. The results show that executable code does not guarantee rule fidelity and that a test result still needs to be checked against the source. A failed test can result from the implementation, from an incorrect expectation or from a failure in an earlier step that prevents the intended check from being reached, while a passed test can still miss a rule error. BoardBench makes it possible to distinguish between source gaps, implementation errors and evaluation problems.

Zusammenfassung

Diese Arbeit untersucht, welche Probleme entstehen, wenn ein Large Language Model Brettspielregeln in eine ausführbare Pythonumgebung übersetzt und wie sich diese Probleme verändern, wenn eine neue Generierung zusätzliche Erklärungen erhält. Diese Übersetzung ist schwierig, weil Regelwerke für Menschen geschrieben sind und Regeln sprachlich uneindeutig formuliert sein können, ein Programm jedoch eindeutige Regeln benötigt, um erlaubte Aktionen, Zustandsänderungen und das Spielende zu bestimmen.

Diese Arbeit entwickelt BoardBench, ein Verfahren zur Erzeugung und anschließenden Bewertung von Spielumgebungen und wendet es auf sechs Spiele anhand ihrer deutschen Verlagsregelwerke an. Regelszenarien sind Tests von erwartetem Spielverhalten, die vor der Codeerzeugung aus den Regelwerken erstellt werden. Einige dieser Tests folgen einer klaren Regel im Regelwerk, während andere einer Entscheidung folgen, die für diese Arbeit getroffen wurde, wenn das Regelwerk eine Situation offenlässt. Ein Coding-Agent erzeugt jede Spielumgebung, ohne Zugriff auf diese Szenarien zu haben. Technische Tests, zufällige Spieldurchläufe, Schnittstellentests, Regelszenarien und Modellbewertungen werden anschließend getrennt evaluiert. In ausgewählten späteren Durchläufen erhält das Modell die ursprünglichen Eingaben zusammen mit einer zusätzlichen Datei, die eine Lücke im Regelwerk erklärt oder eine klare Regel wiederholt, die eine frühere Implementierung nicht umgesetzt hat. Das Modell erstellt dann eine neue Spielumgebung, anstatt den vorherigen Code zu verändern.

Alle finalen Implementierungen haben die technischen Tests bestanden, aber bei allen sechs Spielen wurden Regelfehler gefunden. Dazu gehörten ein falscher Spielaufbau, fehlende erlaubte Aktionen und Fehler bei der Punkteberechnung. Einige darauffolgende Implementierungen korrigierten gezielt ausgewählte Probleme, behielten aber andere Fehler bei oder führten neue ein. Weil jeder spätere Durchlauf eine neue Spielumgebung generiert, können unterschiedliche Ergebnisse der Implementierung von den zusätzlichen Erklärungen oder vom stochastischen Verhalten des Modells kommen. Einige Erwartungen und Tests des Evaluators waren ebenfalls falsch und mussten korrigiert werden. Die Ergebnisse zeigen, dass ausführbarer Code keine korrekte Umsetzung der Regeln garantiert und dass ein Testergebnis weiterhin anhand der Regeln überprüft werden muss. Ein fehlgeschlagener Test kann auf die Implementierung, auf eine falsche Erwartung oder auf einen Fehler in einem früheren Schritt zurückgehen, der verhindert, dass die vorgesehene Prüfung erreicht wird. Andererseits kann ein bestandener Test aber trotzdem einen Regelfehler übersehen. BoardBench ermöglicht diese Analyse, indem Lücken in den Regelwerken, Implementierungsfehler und Evaluationsprobleme getrennt betrachtet werden.

Contents

1 Introduction **1**

2 Fundamentals **2**
2.1 Rulebook and game environments . . . 2
2.2 Large language models and coding agents . . . 2
2.3 PaperBench . . . 3
2.4 OpenSpiel . . . 3
2.5 Evaluation in BoardBench . . . 3

3 Methods **5**
3.1 Game selection . . . 5
3.2 Games . . . 5
3.2.1 6 nimmt! . . . 5
3.2.2 Abalone . . . 6
3.2.3 Bohnanza . . . 6
3.2.4 CATAN . . . 7
3.2.5 Exploding Kittens . . . 7
3.2.6 Wizard . . . 7
3.3 Workflow . . . 8
3.3.1 Rulebook preparation . . . 9
3.3.2 Generating test scenarios . . . 9
3.3.3 Implementation run . . . 9
3.3.4 Implementation evaluation . . . 10
3.3.5 Reviewing and possible additional runs . . . 11
3.3.6 Workflow development and pilot runs . . . 11

4 Results **13**
4.1 Run configuration . . . 13
4.2 Game results . . . 13
4.2.1 6 nimmt! . . . 13
4.2.2 Abalone . . . 15
4.2.3 Bohnanza . . . 16
4.2.4 CATAN . . . 18
4.2.5 Exploding Kittens . . . 20
4.2.6 Wizard . . . 22

5 Discussion **24**
5.1 Rule translation problems . . . 24
5.2 Corrections and new errors . . . 25
5.3 Scenario configuration and interpretation . . . 25
5.4 Technical checks, rollouts and LLM reviews . . . 26

5.5 Limitations . . . 27
5.6 Conclusion . . . 27
5.7 Future work . . . 28

A Used Language Models **30**

B LLM Prompts **31**
B.1 Rulebook Analysis and Scenario Preparation Prompt . . . 31
B.2 Implementation Task . . . 31
B.3 Environment Contract . . . 32
B.4 Independent Rule Review Prompt . . . 34

C Additional Source Material **35**
C.1 6 nimmt! Component Observation . . . 35

D Experimental Interventions **36**
D.1 Abalone Setup Emphasis . . . 36
D.2 Bohnanza Clear-Rule Emphasis . . . 36
D.3 Bohnanza Structured Clarification . . . 37
D.4 CATAN Clear-Rule Emphasis . . . 38
D.5 CATAN Source-Gap Clarification . . . 39
D.6 Exploding Kittens Source-Gap Clarification . . . 40
D.7 Wizard Source-Gap Clarification . . . 40

Bibliography **42**

Chapter 1

Introduction

Board game rulebooks are written for people in natural language, which is why the wording can leave rules unclear or give no explanation for certain special cases. This is often not a big problem for a human group at the table, because players can agree on a meaning or even invent a house rule for it. For a computer program this does not work in the same way, because it needs clear rules and needs to know which moves are possible, which are legal, when the game ends, and who wins.

This raises the question of how well current large language models can translate board game rulebooks into executable game environments and where their limits are. Difficulties can come from the source text, additional components, the number and complexity of the rules or from the generating model. Even when generated code runs, it is not automatically a correct implementation and has to be checked.

The approach for BoardBench is inspired by PaperBench (Starace et al., 2025), where AI agents read scientific papers and rebuild the described experiments in code. BoardBench uses the same kind of translation from a document into code, but the aim is to see which problems appear in this translation rather than to reproduce a paper. The model receives a publisher rulebook and creates an executable Python game environment, which is then tested and reviewed. An important part of the work is therefore an evaluation setup that keeps technical checks and rule checks apart, instead of combining everything into one success score. A single score can hide where something went wrong, because two implementations with different errors in different parts of the evaluation could otherwise look equally good, even though they failed for different reasons.

This thesis examines six games using German publisher rulebooks and images of their pages. After an original implementation has been evaluated, a separate file can clarify a source gap or repeat a clear rule that the model missed. The model then generates a new environment with the additional file, rather than changing the earlier code, so later results show whether this problem changes and whether new errors appear in other rules.

This thesis asks which problems appear when a coding agent translates a rulebook into an executable Python environment. It also examines whether and how these problems change after additional explanations are provided and a new implementation is generated, and how reliably the evaluation can distinguish technical quality from rule fidelity.

This thesis is structured as follows: The chapter Fundamentals explains game environments, large language models and the connection to PaperBench and OpenSpiel. The chapter Methods describes the thesis design, the game selection and the BoardBench workflow. The Results chapter contains the results of the different game cases and conditions. The Discussion chapter examines the findings and limits of the thesis, summarizes the work in a conclusion and presents possible next steps in future work.

Chapter 2

Fundamentals

This chapter explains how a game environment represents rules, how large language models generate code and how the resulting code can be checked. It then introduces PaperBench as the initial inspiration and OpenSpiel as a framework for game environments that was used for an optional comparison during the early BoardBench pilot.

2.1 Rulebook and game environments

A board game rulebook explains how to set up and play a game through text, illustrations and examples. Some information comes from the physical components, such as the cards in a deck or a printed layout on a board. A digital environment must represent this information and provide actions that change the game state according to the rules.

A game state contains the information needed to continue play, such as card positions, the current player and the game phase. A legal action is a choice permitted by the rules in that state. The complete state may even contain information that individual players cannot see. An observation contains only the information available to a particular player, such as their own hand and the public cards. A terminal state marks the end of a game, while returns express the outcome numerically for each player, for example as a win, loss or a final score. The environment must determine when a game is over and calculate results according to the rules.

2.2 Large language models and coding agents

A large language model (LLM) is a neural network that typically uses the transformer architecture (Vaswani et al., 2023) and is trained on large collections of text. During training, internal numerical values called weights are adjusted so that the model learns patterns from the training data. Text is processed in units called tokens, which can represent words, parts of words or punctuation marks. When text is generated, the model predicts the next token from the available input and the tokens already generated.

The input contains instructions and can include additional context such as files or images, depending on what the model supports. All these inputs together are called a prompt. The output can be text, such as explanations or code, and some models can also generate images. Adding a clarification to the prompt gives the model more information, but does not retrain it or change its weights. The response a model gives depends on the prompt and what the model learned during its training. The context window limits how much information the model can work with in one request and its size is measured in tokens.

Models differ in their training data, size, architecture and training procedure, which can all affect how they interpret instructions and generate their outputs (LeCun et al., 2015). Text

generation can include random choices internally, so that the same input can produce different outputs in different runs.

Without tools, models return a response but do not execute any actions such as running a program file to check for runtime errors. An agent harness is software that gives the model access to tools for reading files, writing code and running commands. With this, the model can request an action, the harness executes it and the result becomes additional input for the model. The model can then use error messages, for example, to find and fix bugs that might otherwise go unnoticed before the task is finished.

Earlier work on LLM code generation evaluated models on relatively small programming tasks. Codex, for example, was evaluated with HumanEval, where models generate Python functions from natural language descriptions (Chen et al., 2021). SWE-agent studies agent-computer interfaces and shows how an agent interface can let a language model inspect files, edit code and run tests during a task (Yang et al., 2024).

2.3 PaperBench

PaperBench is a benchmark from OpenAI that evaluates whether an AI agent can reproduce research results from a given paper (Starace et al., 2025). The agent receives the paper and additional explanations, then writes code and runs the experiments without using the original implementation. This inspired BoardBench's initial idea of giving an agent a board game rulebook and letting it create an executable environment.

The PaperBench evaluation divides the reproduction task into smaller requirements that distinguish whether code was written, whether the required computation was executed and whether the reported result was reproduced. An implementation can therefore meet some requirements even if it does not reproduce the complete result.

BoardBench follows the general idea of translating a document into code and evaluating the result, but does not use PaperBench code or its evaluation system.

2.4 OpenSpiel

OpenSpiel provides implementations of board games and algorithms for research in reinforcement learning (Lanctot et al., 2020) and its interface defines how another program can access these games, for example by requesting legal actions or applying a move. The interface separates the general properties of a game from its state at a particular point. For a random event, the environment can provide the possible outcomes and their probabilities.

BoardBench uses the same kind of methods, such as creating an initial state, requesting legal actions and applying a move, but it does not use the OpenSpiel interface. All generated games instead share one BoardBench interface, so the same technical checks can run on different games. This BoardBench interface is needed for the evaluator and is independent of the earlier OpenSpiel comparison. BoardBench generates its own environments rather than reusing OpenSpiel implementations or algorithms, because the thesis only evaluates whether those environments follow the rules and does not train agents to play them.

An optional direct game state and action comparison with OpenSpiel was implemented during the pilot runs, but none of the games selected for the final study had a matching OpenSpiel environment. For this reason, the final evaluation does not include an OpenSpiel comparison.

2.5 Evaluation in BoardBench

BoardBench's evaluation checks whether an implementation can be used through the required interface and whether selected actions follow the assigned rules. Expectations from clear rules

remain separate from decisions added where the source is insufficient.

Technical checks can confirm that an implementation starts and provides the required methods, but they do not show if it uses the correct setup or follows the rulebook. A rule scenario therefore defines a game state, an action and an expected result. The evaluator applies the action and compares the observed result with the expected one.

If the expected result does not match the assigned source, the scenario can fail even when the code follows the rules. Passing every configured scenario therefore only shows that the implementation matches the tested expectations, not that it is correct for every possible state and action sequence.

Chapter 3

Methods

This chapter explains how the six games were selected in Section 3.1 and describes each game in Section 3.2. It then presents the final workflow for preparing the rulebooks, generating the game environments and evaluating the implementations.

3.1 Game selection

The games were selected to cover different game types and to balance rule complexity, because games with very few rules and little interaction reveal little about problems that appear when rules depend on each other or contain special cases. Games with very high complexity, many edge cases or many exceptions, on the other hand, quickly exceed what can be evaluated transparently within a bachelor's thesis. The selection therefore aims to provide enough structure to expose translation problems without making systematic evaluations infeasible. The selection also avoids widely known traditional games such as chess to reduce the risk of the model relying on familiar rules instead of the supplied rulebook. However, the selected games may also have appeared in the model's training data, so prior knowledge cannot be ruled out.

A second design constraint concerns the model input. Game publisher rulebooks often mention components such as cards, tokens or dice only briefly, because players can identify them from the physical game. If the rulebook does not show or describe all required components, the model has to make assumptions about the missing information. The text alone, even with a few graphical explanations, may be insufficient to reconstruct a complete digital environment. Identifying the missing component information may require a physical copy of the game or additional component lists. For most games the rulebook is the only source of game rules. Two exceptions are 6 nimmt! and CATAN. For 6 nimmt!, a separate component list describes the cards in the game and CATAN uses an additional rulebook called the Almanac, which is required by the base rulebook. In addition to the page images, the model receives the generation instructions together with the required interface structure.

3.2 Games

3.2.1 6 nimmt!

6 nimmt! is a strategic card placement game (Kramer, 2011). The deck uses cards numbered from 1 to 104 and each card also shows a number of bullhead penalty points. Most cards are worth one bullhead, but multiples of five are worth two, multiples of ten are worth three and repeated digit numbers (11, 22, ..., 99) are worth five. Card 55 is an exception worth seven.

Each player receives ten hand cards. Four cards from the deck are placed face up to start the four rows and the leftover cards are set aside. Each turn every player secretly chooses one hand card and after all choices are made, the cards are revealed together and are placed from the

lowest to the highest card number. A card is normally placed at the end of the row whose last card has the highest value below the played card's value. If a card is lower than every row end, its owner must choose one of the four rows and take all cards from that row as penalty points. The player's card then starts a new row that replaces the old one. If a card would become the sixth card in a row, its owner takes the five cards already there and the played card starts the new row. After all ten cards have been played, the game ends if any player has more than 66 penalty points and the player with the fewest points wins. Otherwise players keep their points and receive new cards for the next round.

3.2.2 Abalone

Abalone is a two player marble pushing game on a hexagonal board, where each player controls one color and tries to push opposing marbles off the board (Lévi and Lalet, n.d.). The first player to push six opposing marbles off the board wins.

The game starts from a fixed setup shown in the rulebook, where each side begins with exactly 14 marbles. On a player's turn they make one move with one, two or three of their own marbles. If two or three marbles are selected, they must form one straight continuous line. All selected marbles must move one pit in the same direction, either along their row or sideways. Opponent marbles move only when they are pushed as part of such a move.

Pushing is allowed only in a straight inline movement and the pushing group must be strictly larger than the opposing group. There must be no gap between the groups and there must be an empty pit behind the defending marbles unless they are pushed off the board. A marble pushed over the edge is removed from the game.

3.2.3 Bohnanza

Bohnanza is a planting and trading card game (Rosenberg, 2016). The deck contains eight bean types that differ in rarity and in harvest value. Each bean card has a beanometer that shows how many cards must be harvested from one field to earn one, two, three or four coins. Rare bean types need fewer cards in a field to earn the same number of coins as more common types.

Each player has three fields in a game with three players and two fields in a game with four or five players. A field may hold only one bean type, but the same type can also be planted in more than one field. Players can harvest their fields at any time in the game, including outside of their own turn. However, a field with only one card cannot be harvested while another of that player's fields still has more than one bean on it.

The hand order is fixed for the whole game and newly drawn cards are added behind the existing ones. The game is played in turns and has four phases:

1. The active player must plant the first card in their hand. Planting a second hand card is optional, but more are not allowed.

2. The player reveals the top two cards of the draw pile and may trade with other players. The other players are only allowed to trade with the active player and not with each other. Unequal trades are allowed if both sides agree.

3. All received cards must be planted and are not added to the hand. If the revealed cards were not traded, then the active player must plant them.

4. The active player ends their turn by drawing three cards from the draw pile.

The first two times the draw pile becomes empty, the discard pile is shuffled to form a new draw pile. The game ends when the draw pile becomes empty for the third time. If this happens in phase two, trading and planting are completed before the final scoring, but phase four is skipped. All fields are then harvested, while cards left in the player's hand do not score. The player with the most coins wins.

3.2.4 CATAN

CATAN is a game in which players collect resources, trade and build roads, settlements and cities (Teuber, 2022a,b). The thesis uses the illustrated beginner setup for three and four players. The board consists of terrain hexes that produce wood, brick, wool, grain or ore, while the desert produces nothing. Settlements and cities stand at the corners of these hexes and roads connect them along the edges.

Each player starts with two settlements and two roads. On a turn the active player must roll two dice, may then trade and may then build. When a number other than seven is rolled, the matching terrain hexes produce resources for every adjacent settlement and city. A settlement receives one resource card and a city receives two. One development card can be played during the active turn, including before the roll, but not on the turn when it was bought. Victory point cards are an exception and can be revealed to win immediately, even when bought in that turn.

Only the active player may trade with other players or with the bank. Resources are spent on new roads and settlements, upgrading settlements to cities or buying development cards. A new settlement must connect to the player's road and cannot be placed next to another settlement or city. Development cards can move the robber or place free roads.

When a seven is rolled, no terrain produces resources, and players with more than seven resource cards discard half of them, rounded down. The active player then moves the robber to another hex. That hex produces nothing while the robber is there and the player can steal a resource card from an opponent with an adjacent settlement or city.

A settlement is worth one victory point and a city is worth two. The Longest Road and the Largest Army each provide two points, while development cards with victory points provide one each. The Longest Road needs a continuous road of at least five segments. Branches are not added together and an opposing settlement or city can break a route. The game ends as soon as the active player has at least ten victory points during their turn.

3.2.5 Exploding Kittens

Exploding Kittens is a last player standing card game (Exploding Kittens, 2018). All players start with seven random cards, plus one Defuse card. On their turn players may play as many of their action cards as they like. Unless a card ends their turn without drawing, they have to finish by drawing the top card from the draw pile. If they draw an Exploding Kitten, they must reveal it. If they still have a Defuse, they play it. Otherwise, they are eliminated and their remaining cards, including the Exploding Kitten, go to the discard pile. The last remaining player wins.

After a successful Defuse the player has to take the Exploding Kitten card and can place it anywhere in the draw pile. Skip ends the current turn without drawing. Attack ends the attacker's turn without drawing and forces the next player to take two turns in a row. If that player also plays Attack during those owed turns, the following player then owes exactly two turns as well. The Favor card forces a selected opponent to give the active player one card chosen by that opponent. Two cards of the same title can be played as a pair to steal one random card from a chosen opponent. For Favor and Pair, the thesis excludes opponents with no cards because the rulebook leaves this situation unclear.

3.2.6 Wizard

Wizard is a trick taking card game for three to six players (AMIGO Spiel + Freizeit GmbH, 1996). The deck contains 60 cards, including four colored suits with numbers from one to thirteen, four Wizard cards and four Jester cards. The game is played in rounds, where the deck is shuffled every round and the dealer also moves one seat clockwise after each round.

The number of cards dealt to each player increases with the round number and starts with one card, then two and so on. The top undealt card is then turned face up and its color determines

the trump color for that round. Among ordinary cards, even a one of the trump color beats a thirteen of another color, while the four colors otherwise have no ranking among themselves. If a Jester is the top card then there is no trump. If a Wizard is on top then the dealer chooses the trump color, after looking at their hand. In the final round, the complete deck is dealt and no trump suit is determined.

After the cards have been dealt, every player bids how many tricks they expect to win this round. The bidding starts with the player on the left of the dealer and continues clockwise. This player also leads the first trick. Later tricks are led by the person who won the last trick. Players must follow the led color if they can, but may play a different color or trump card otherwise. Wizards and Jesters can be played at any time, even if the player has a card of the led color. The first Wizard in a trick always wins. The first Jester also wins the trick if only Jesters are being played and otherwise always loses the trick. If a Wizard leads, every later card is legal and the first Wizard wins anyway, but if a Jester leads, a later ordinary card sets the led color. For the unclear sequence in which a Wizard appears before that ordinary card, an approved decision keeps the trick colorless and allows all later cards.

The players get twenty points if they win exactly the number of tricks they bid, plus ten bonus points for every trick they win. If they miss their bid, they lose ten points for each trick above or below their bid. The player with the highest score after the last round wins.

3.3 Workflow

The final workflow consists of the following five steps:

1. The workflow starts by rendering the original PDF rulebooks as PNG images. The later LLM calls receive both formats so that the model can use the original document and clearly see visual rules such as diagrams and setup examples.

2. The LLM is first called with a rule analysis prompt that derives small rule statements and possible edge cases from the PDF and page images. The output is checked against the assigned sources when test scenarios are prepared. Testable clear rules provide expectations for clear-rule scenarios. Ambiguous, missing or conflicting rules can become human-decision scenarios after an approved decision defines the expected behavior. Both types of scenarios are kept hidden from the later implementation call.

3. A new independent call to the same LLM receives the PDF, page images and a prompt with programming instructions. It creates the original Python game environment in an isolated workspace, separated from the complete test setup. The implementation must pass BoardBench's technical gate before it continues to the full evaluation.

4. The game implementation is then evaluated immediately after passing the technical gate. The technical checks test whether the game can be started and used through the required interface. Other tests run random games, cover the supported player counts and execute the hidden rule scenarios. Three independent LLM calls also review the produced code against the rulebook. Technical checks, runtime robustness, interface checks, rule scenarios and LLM reviews are reported separately.

5. After the full evaluation the results are reviewed to decide whether another run is useful. This can be a repetition with unchanged input, a clarification of an unclear, missing or conflicting rule or an emphasis of a clear rule missed by the original implementation. For a clarification or emphasis, the original PDF and page images remain unchanged and only a separate file with extra explanations is added. The model then creates a new implementation, which is evaluated in the same way as the original implementation.

3.3.1 Rulebook preparation

For each game the workflow starts with the assigned publisher rulebook in PDF format. An official companion document is added only when the main rulebook requires it for additional rules. Before the companion document is used, the agent checks that it matches the edition of the main rulebook. This helps prevent mixing rules from different editions. A separate custom component list may also be included when the rulebook does not provide enough information about the physical game components. This list only describes the components, but does not change any rules.

The original PDF remains unchanged throughout the workflow and each page is additionally rendered as an image at 150 DPI so that diagrams, setup illustrations and other visual information are available for the model. The source files and rendered pages are also hashed, allowing each run to be linked to the exact files used as inputs. A hash is calculated from a file's contents and helps check whether the file has changed.

3.3.2 Generating test scenarios

For each game the test scenarios are prepared from the assigned rule sources and edge cases identified during the rule analysis. The rule analysis divides the relevant rules into small claims that each describe one rule requirement or gap. Each claim is classified as clear, ambiguous, missing, conflicting or untestable. Testable clear claims provide direct expectations for clear-rule scenarios, while exceptions to scenario coverage remain documented. Ambiguous, missing or conflicting claims require an approved decision before they can support a human-decision scenario. Untestable claims remain documented but do not receive a scenario.

After human review and approval, the rules and decisions are saved as separate files for the game. A Markdown file provides a readable overview of the approved rules, while JSON files store the rules, decisions and planned scenarios for the evaluation. The planned scenarios are then made executable by defining the required game state, action and expected result. Each executable scenario also contains a direct quote and a related page or file location. A human-decision scenario links the source evidence for the identified gap with the separately approved decision that defines its expected behavior.

Some scenarios need specific states that random play does not reliably reach. For such scenarios an evaluator adapter prepares the required game states. The adapter prepares states, identifies the requested actions and compares the observed data with the expected values supplied by the scenario. Both the completed scenarios and their adapters are kept hidden from the implementation model. Generation of the game environment starts only after these files are complete.

3.3.3 Implementation run

After the test scenarios have been completed, the implementation model generates the game environment. It receives both the assigned PDF and image source files. The model also receives the programming prompt, environment contract and the game data profile. The environment contract specifies the common methods and JSON structure, while the data profile specifies the state fields, action names and supported player counts for the selected game. These files standardize the representation used by the evaluator, while the assigned sources remain the authority for game behavior. For example, the profile can require a particular state field even when the game situation could be represented differently. A failure to provide that field must therefore be distinguished from an incorrectly implemented game rule.

The generation prompt, the judge prompt and the environment contract are included in Appendix B. Other generation artifacts are available in the BoardBench repository.

Before the run starts, BoardBench checks the supplied files and confirms through an isolation test that the model cannot access the scenarios or evaluator files. The approved rule claims, test scenarios and evaluator adapters therefore remain hidden for the implementation run.

Inside this workspace, the model operates through an agent harness, which allows it to inspect the supplied documents, write the Python module and run commands. The model also records its material assumptions and reports which parts of the rulebook it implemented.

Before the implementation can enter the evaluation, it must pass a technical gate run by BoardBench. The gate checks the required files, compiles the module and tests its public methods, data structures and supported player counts. It does not use the hidden rule scenarios and therefore does not determine whether the game rules have been implemented correctly. If a check fails, the model receives only the technical output and may repair the implementation in the same workspace. A run allows at most two repair attempts. Only an implementation that passes the gate continues to the evaluation while a run that does not is recorded as failed.

3.3.4 Implementation evaluation

BoardBench uses five separate evaluations: the technical gate, runtime robustness, the interface checks, rule scenarios and independent LLM reviews. Their results are reported separately instead of being combined into one correctness score.

The technical gate first confirms that the generated module can be imported and used to create an initial game state and that it provides the required public methods and data conversions. Random rollouts play the game by repeatedly choosing a random action from those offered by the environment in each game state. They check that the game does not crash, that an unfinished game has legal actions, that a finished game has none and that rendering and action names stay consistent. A rollout also passes when it reaches the step limit without detecting an error. The reported pass rate therefore measures runtime robustness within the sampled steps, not how often the game reaches a terminal state.

The interface result counts the offered actions checked in the states visited during separate random rollouts. For each action, the evaluator checks that its generated name is unique within that state and can be converted back to the same action. These counts can include repeated actions and do not represent independent rule tests. A separate check starts the environment for every supported number of players and confirms that the unsupported player counts listed in the game profile are rejected. Setup rules that depend on the player count are checked by the rule scenarios because starting a game alone does not show that its setup is correct.

Rule fidelity describes how closely the environment's behavior follows the assigned game rules and is evaluated with the scenarios prepared before the generation. A scenario passes when the observed behavior matches its cited expectation, fails when the expectation is contradicted and crashes when an expected legal execution raises an error. If the required state cannot be reached or the result cannot be observed, the scenario remains unevaluated. If a required earlier action in the sequence fails, the scenario still counts as failed, even when a later intended check was not reached. BoardBench groups the results by whether the expectation comes from a clear rule or an approved human decision. It calculates a separate pass rate for each group using the scenarios that pass, fail or crash. Scenario evaluated coverage is the share of configured scenarios that passed, failed or crashed rather than remaining unevaluated. In both groups a failure only records that the expected behavior was not produced, but it does not identify the cause. Claim mapping coverage is the share of required claims linked to at least one scenario, while evaluated claim coverage is the share of required claims linked to at least one evaluated scenario. Neither measure shows that every part of each claim was tested.

The last evaluation step uses multiple LLM judges to compare the assigned sources and approved rule facts with the implementation. Each review is a separate model run that receives the assigned sources, approved rule facts and the generated implementation. LLM based evaluation can provide an additional review signal, but its scores are not calibrated measurements of

rule correctness. Previous work on LLM as a judge has also identified biases and disagreements in model based evaluation (Zheng et al., 2023). The judge scores are therefore used as rough summaries of reviews and small differences between scores are not interpreted as meaningful differences in rule fidelity.

Each judge provides a score from zero to one and a confidence level. The judge scores are summarized by calculating the mean, while the sample standard deviation shows how much the scores differ. These values remain separate from the mechanical and scenario results because an LLM review is not a deterministic test. The judge also cannot access the scenarios, evaluation results, earlier reviews or implementations from other runs. Critical and major findings require source evidence, the relevant code location and a description of the expected and implemented behavior.

3.3.5 Reviewing and possible additional runs

After the evaluation, the results are reviewed to identify problems that should be investigated in another run and the code remains unchanged once evaluation starts. Any later implementation is recorded as a new run and the earlier code and results are kept. Depending on the observed problem, this can be a repetition with unchanged input, a source gap clarification or an emphasis of a clear rule. A repetition uses the same source files and programming instructions as before to check whether an observed result appears again, while the model does not receive the previous implementation. The repeated input can be the original condition or one that already included a clarification. If instead a rule is unclear, missing or conflicting, an approved explanation is added in a separate file. This is called a source gap clarification. The new run then checks whether making this source gap explicit changes the observed problem. If the implementation missed a clearly stated rule, a separate emphasis file repeats that requirement in the model input instead of adding a new approved decision. This tests whether highlighting an existing rule helps the model implement it correctly, rather than whether additional information resolves a source gap. An improvement after this emphasis does not mean that the original rulebook lacked the required information.

For a clarification or emphasis, the original rulebooks and page images remain unchanged. Only the separate clarification or emphasis file is added to the model input. The model then creates a new implementation from the beginning with the same programming instructions, contract and game profile. The new implementation is evaluated in the same way as the original. The comparison examines both the targeted problem and possible regressions in other rule areas. Because the changed input is selected after reviewing the earlier results, the run is not treated as an independent repetition.

3.3.6 Workflow development and pilot runs

The final workflow was developed through earlier pilot runs with several games and different model providers. These pilots compared generation in one response with agentic runs that let the model look at the files, run checks and change its own code. Some of these one-shot implementations needed extra fixes before the checks could run, while the agentic runs could try those checks themselves. The final study uses this agentic workflow for that reason, but the sources and the evaluation also changed during development.

The early pilots used one combined score for each game that mixed technical checks, random play, action names and LLM reviews, while no rule scenarios were included. The technical checks often passed fully, so the total score stayed high even when the reviews found visible rule errors. A CATAN implementation made this problem clear because the combined score looked strong despite these rule problems. The final workflow therefore reports technical checks, runtime robustness, interface checks, rule scenarios and LLM reviews separately, so a successful technical run cannot hide incorrect rules with one high score.

An early Exploding Kittens comparison added extra decisions in a later run, but the way the rules were given to the model also changed as an experiment. The original run received the publisher PDF, while the later run received only one text file that contained the rules and the extra decisions together. A different result could then come from the decisions or from using text instead of the PDF. In the final workflow the publisher PDF and page images remain unchanged and extra decisions are added in a separate file.

The pilots also included an optional comparison with OpenSpiel, but it required matching the actions, states and player turns of the two implementations. None of the games selected for the final study had a matching OpenSpiel environment, so this comparison was only used for early BoardBench calibration and not used in the final evaluation. A common interface for all game implementations was still needed so that the same technical checks can run on different generated games.

Other pilot checks paired a rulebook with an implementation of a different game to test whether the judges would notice this mismatch. The judges recognized the mismatch and gave mostly zero scores, but recognizing only such a mismatch does not show that they can detect small rule errors and the final reviews therefore need source evidence, a code location, and a description of the expected and implemented behavior.

Chapter 4

Results

4.1 Run configuration

Aspect	Configuration
Evaluated games	6 nimmt!, Abalone, Bohnanza, CATAN, Exploding Kittens and Wizard
General source material	Assigned German publisher rulebook PDF
Additional sources	Custom component list for 6 nimmt!, Matching official Almanac for CATAN
Implementation model (thinking level)	`openai-codex/gpt-5.6-sol` (low)
Judge model (thinking level)	`openai-codex/gpt-5.6-sol` (medium)
Runtime robustness rollout count	100
Maximum steps per robustness rollout	1 000
Independent LLM reviews per implementation	3

Table 4.1: Configuration of the evaluated runs.

The thinking level is a model setting for the requested reasoning effort, not a measured level of correctness. For each game, separate resource tables summarize model usage during implementation, repairs and reviews. Cached tokens are input tokens recorded as reused from earlier processing, while reasoning tokens are tokens reported for the model's internal reasoning. Cached tokens are included in the input total and reasoning tokens are included in the output total. For 6 nimmt!, these totals also include three additional reviews with specialized instructions, which do not contribute to the neutral judge mean. The USD values estimate equivalent API costs rather than actual subscription charges.

4.2 Game results

4.2.1 6 nimmt!

The assigned sources for 6 nimmt! were the German publisher rulebook (Kramer, 2011) and a separate custom component list. This list defines the complete deck and the bullhead value of each card. The original condition covered the base game for two to ten players. The professional variant and optional target and game count variants were excluded.

	Original
Technical checks	4/4
Robustness rollouts	100/100
Interface checks	77 788/77 788
Clear-rule scenarios	24/24
Human-decision scenarios	9/9
Neutral judge mean (sample SD)	0.847 (0.031)

Table 4.2: 6 nimmt! evaluation results.

	Original
Input tokens (cached)	2 561 820 (2 115 840)
Output tokens (reasoning)	56 099 (21 455)
Estimated equivalent API cost	USD 4.97
Generated Python lines	377

Table 4.3: 6 nimmt! resource usage.

The implementation passed all four technical checks and passed all 100 sampled robustness rollouts. Every sampled action name was unique within its state and could be converted back to the corresponding action. All 33/33 configured scenarios passed. The implementation run required two repairs and produced 377 Python lines.

Two earlier evaluator runs were invalid and remained unscored. The first did not create a game with the four players required by the scenario setup and caused 19 setup crashes. The second checked for visible cards in only one part of the public state. The corrected evaluator created the required game and accepted a played card wherever it appeared in the public state. The rule expectations remained unchanged, but the executable setup and visibility checks were corrected.

However, independent review found an information error outside the configured scenario suite. Under the rules all four selected cards should become visible before placement starts. The replay began with a row containing cards 10, 11, 12 and 13, and the players selected cards 20, 30, 50 and 70. Card 20 was placed as the fifth card in the row before card 30 was placed and took that row. The implementation then deleted the shared record of revealed cards. The cards in the taken row were visible only to the player who received them. As a result, card 20 was missing from the shared observation, although all selected cards should have been visible together.

The finding was not added as a failed scenario because the replay was created and confirmed after the scenario evaluation had finished.

Figure 4.1 shows the separate scenario and judge results for the original condition.

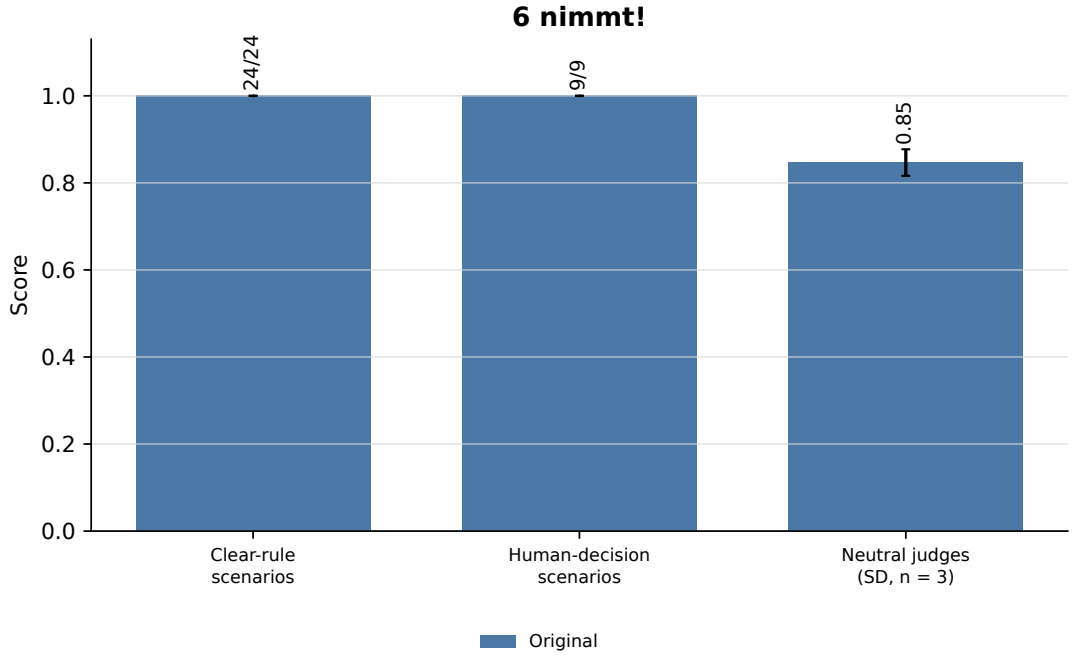


Figure 4.1: 6 nimmt! evidence profile for the original condition.

The neutral judge mean was 0.847, but two of the three neutral judges independently reported the missing shared visibility. The implementation passed every configured technical check and rule scenario, but independent review found one reproducible rule error outside the scenario suite.

4.2.2 Abalone

The assigned source for Abalone was only the German publisher rulebook (Lévi and Lalet, n.d.). BoardBench covered the untimed base game for two players and the optional clock was therefore excluded. The rule emphasis repeated the requirement to start with 14 marbles of each color.

	Original	Rule emphasis
Technical checks	4/4	4/4
Robustness rollouts	100/100	100/100
Interface checks	8 888 062/8 888 062	8 528 518/8 528 518
Clear-rule scenarios	32/33	33/33
Human-decision scenarios	4/4	4/4
Neutral judge mean (sample SD)	0.867 (0.031)	0.893 (0.081)

Table 4.4: Abalone evaluation results.

	Original	Rule emphasis
Input tokens (cached)	1 101 308 (910 336)	944 830 (747 264)
Output tokens (reasoning)	21 546 (9 444)	21 111 (8 197)
Estimated equivalent API cost	USD 2.06	USD 1.99
Generated Python lines	245	344

Table 4.5: Abalone resource usage.

The original and first rule emphasis implementations passed all four technical checks, passed all 100 sampled robustness rollouts and passed the sampled action conversions. The original passed 32/33 clear-rule scenarios, while the rule emphasis passed 33/33, and both passed 4/4 human-decision scenarios. The neutral judge mean increased from 0.867 to 0.893.

The only configured failure in the original concerned the initial setup. It started with 13 instead of the required 14 marbles of each color. The rule emphasis corrected this setup and therefore passed all 33 clear-rule scenarios. The scenario results differed only in this setup check, while the new implementation contained 344 Python lines rather than 245.

An earlier evaluator required a forced pass when the active player had no legal moves, although the rulebook does not define this situation. The constructed state was also not shown to be reachable from the normal setup, so the current evaluator leaves this exception unscored. A second implementation with the same rule emphasis had already been generated after the first failed this earlier expectation. It also passed 33/33 clear-rule scenarios and 4/4 human-decision scenarios under the current evaluator, but does not appear as a separate table column. The judge reviews were kept from the earlier evaluation and still include criticism of the missing forced pass. Their scores were not recalculated after this expectation was removed from the scenario evaluation.

Figure 4.2 compares the original implementation with the first rule emphasis implementation.

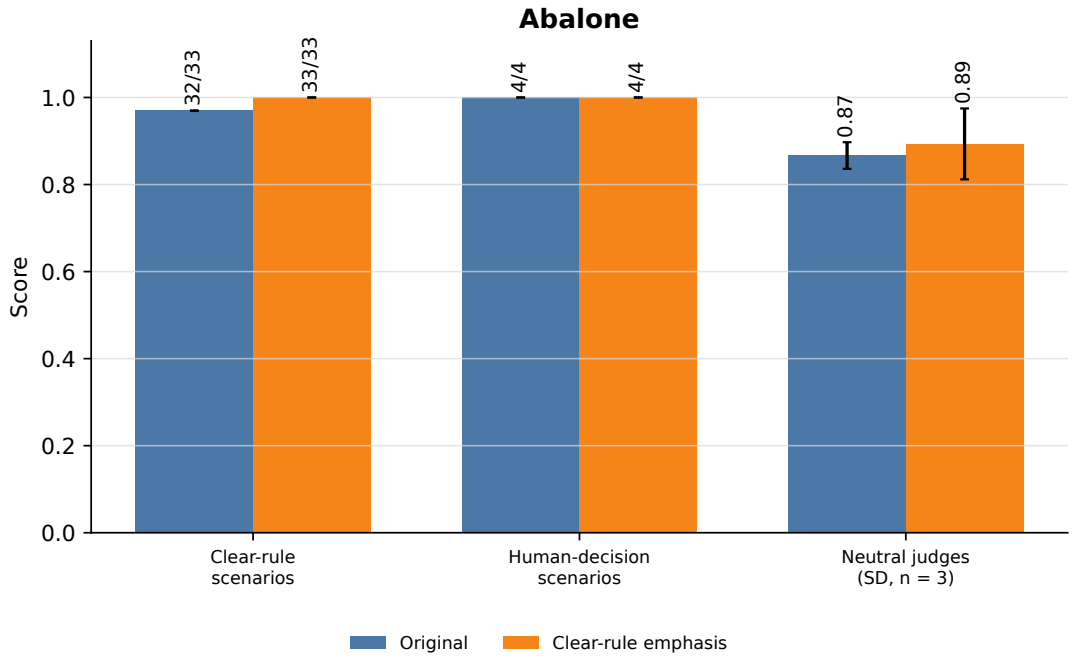


Figure 4.2: Abalone evidence profile.

The neutral judge means were 0.867 for the original, 0.893 for the first rule emphasis run and 0.870 for the second. All three original judges reported the missing starting marbles. All three judges of the second rule emphasis run also reported that the parser accepted different action names when the same marble group was listed in a different order. A replay confirmed these alternative names after evaluation and the finding remains unscored. Both rule emphasis runs therefore corrected the scored setup error, while the second also had this separate confirmed finding.

4.2.3 Bohnanza

Bohnanza used the German publisher rulebook (Rosenberg, 2016) and covered the base game for three to five players. The original implementation received only this rulebook and did not

offer unequal trades, used incorrect Garden and Soy bean values and delayed the game end, while the rule emphasis repeated these requirements. The structured clarification added four approved decisions for the starting player, planting order, harvesting and private information and a checklist covering the complete game.

	Original	Rule emphasis	Structured clarification
Technical checks	4/4	4/4	4/4
Robustness rollouts	100/100	100/100	100/100
Interface checks	800 371/800 371	1 523 314/1 523 314	847 456/847 456
Clear-rule scenarios	33/38	30/38	33/38
Human-decision scenarios	4/4	3/4	3/4
Neutral judge mean (sample SD)	0.643 (0.081)	0.560 (0.060)	0.713 (0.031)

Table 4.6: Bohnanza evaluation results.

	Original	Rule emphasis	Structured clarification
Input tokens (cached)	1 418 536 (1 172 992)	1 254 566 (1 044 224)	1 529 790 (1 296 640)
Output tokens (reasoning)	40 460 (20 220)	33 972 (16 477)	39 841 (21 154)
Estimated equivalent API cost	USD 3.03	USD 2.59	USD 3.01
Generated Python lines	408	421	385

Table 4.7: Bohnanza resource usage.

The three implementations shown in the tables passed all technical checks without repairs, passed all 100 sampled robustness rollouts and passed every sampled action conversion. The original and structured clarification passed 33/38 clear-rule scenarios, while the rule emphasis passed 30/38. The human-decision scenario result was 4/4 for the original and 3/4 in both later evaluations. A separate repetition of the rule emphasis was also evaluated and remains documented in the detailed results.

Two additional implementations with the same structured clarification passed 35/38 and 33/38 in the clear-rule scenarios and both passed 3/4 human-decision scenarios. Their judge means were 0.523 and 0.617, so both the scenario results and the judge reviews varied across generations.

The rules allow unequal trades with several cards, but the original implementation did not offer these exchanges as legal actions. It also used incorrect coin values for Garden and Soy beans, which changed how many coins players received when harvesting these fields.

When the draw pile becomes empty for the third time during the reveal phase, the active player finishes the reveal, trade and planting steps before the game ends. The original implementation instead waited for a later draw action that should no longer have been available. This action drew no cards and ended the game without starting the next player's turn. The error delayed the terminal state and final harvest, but the checked replays did not start another player's turn or change the final coin values after the extra action.

The rule emphasis implementation corrected unequal trades, the Garden and Soy values and the delayed game end, but planting the optional second card no longer advanced the phase. Players also lost the choice of planting order for received and revealed cards, while an incorrect Red bean value changed the final scores.

One of the eight failed clear-rule scenarios expected separate harvest and plant actions. The implementation also allowed the player to select a field and harvest it before planting within one action. A later review showed that this preserved both the field choice and the required order of steps, so the failure reflected the expected action structure rather than an observed rule violation. The tables keep the originally recorded result.

The structured clarification implementation passed the scenarios for setup, bean values, harvesting outside the active turn and final scoring. However, planting the optional second card from the hand did not move the game to the reveal phase and unequal trades with several cards

were still missing. During the planting phase, players could only choose the first card from each list of received or remaining revealed cards to plant next. For example, when Soy and Red beans were waiting to be planted, only Soy could be chosen first, although the rules allow either order. This restriction caused two clear-rule scenarios and one human-decision scenario to fail, so these three failures reflected the same problem with choosing the next card to plant.

Figure 4.3 compares the original, rule emphasis and structured clarification implementations.

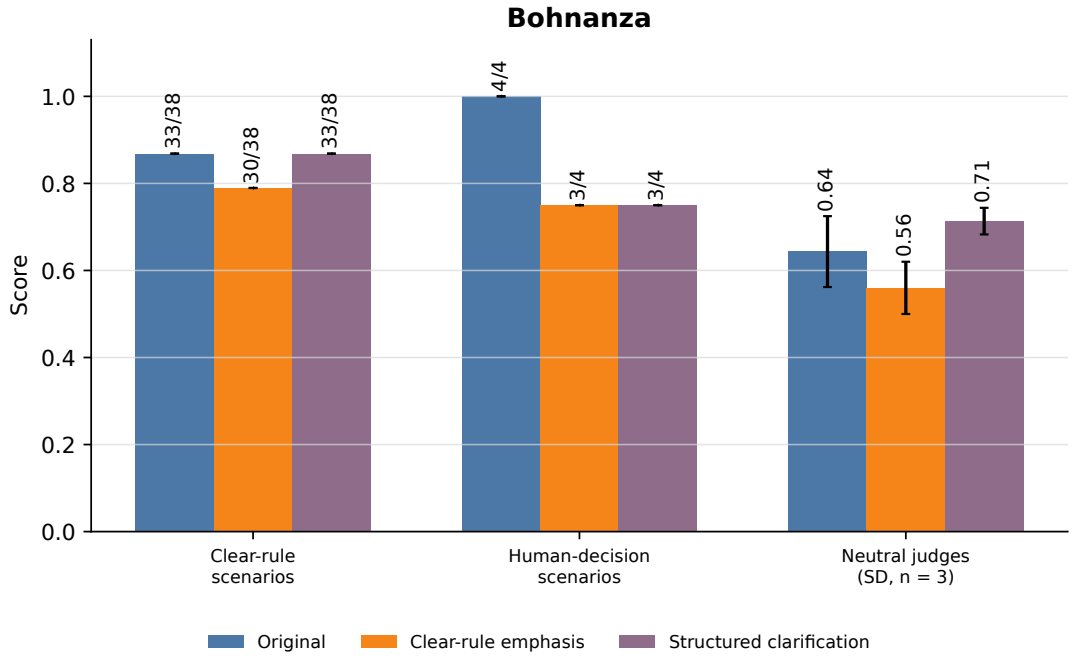


Figure 4.3: Bohnanza evidence profile.

The neutral judge mean was 0.643 for the original, 0.560 for the rule emphasis and 0.713 for the structured clarification. All three judges of the structured clarification reported missing unequal trades, a fixed planting order and trade actions that showed private cards in opponents' hands. Unlike the rule emphasis implementation, the structured clarification passed the scenarios for the Red bean value and final scoring. However, unequal trades were missing again and the restriction on the planting order also remained.

4.2.4 CATAN

The assigned sources for CATAN were the German publisher rulebook (Teuber, 2022a) and its matching official Almanac (Teuber, 2022b). All three conditions covered the illustrated beginner setup for three and four players with separate roll, trade and build phases. The rule emphasis repeated the requirements for Longest Road and the number of road pieces still available to a player, while the source gap clarification added four decisions for trade, discard, robbery and maritime exchange.

	Original	Rule emphasis	Clarification
Technical checks	4/4	4/4	4/4
Robustness rollouts	100/100	100/100	100/100
Interface checks	8 883 707/8 883 707	4 342 395/4 342 395	798 680/798 680
Clear-rule scenarios	37/40	38/40	32/40
Human-decision scenarios	10/15	13/15	11/15
Neutral judge mean (sample SD)	0.653 (0.070)	0.817 (0.067)	0.823 (0.038)

Table 4.8: CATAN evaluation results.

	Original	Rule emphasis	Clarification
Input tokens (cached)	4 705 367 (3 872 768)	2 949 963 (2 301 952)	3 551 441 (2 854 144)
Output tokens (reasoning)	42 879 (19 750)	39 937 (18 186)	40 131 (20 256)
Estimated equivalent API cost	USD 7.39	USD 5.59	USD 6.12
Generated Python lines	370	639	489

Table 4.9: CATAN resource usage.

All three implementations passed the four technical checks, passed all 100 sampled robustness rollouts and passed the sampled action conversions. The rule emphasis raised the clear-rule result from 37/40 to 38/40 and the human-decision result from 10/15 to 13/15. The source gap clarification passed 32/40 clear-rule scenarios and 11/15 human-decision scenarios.

Every implementation passed the technical gate without repairs. An earlier replay of the original implementation remained unscored, because three expectations required state details that the game data profile did not yet define. The same implementation was evaluated again after these expectations were removed. The first judge reviews of the original and rule emphasis conditions received the Almanac PDF but not rendered images of its pages. Those reviews remain archived and are not used in the table. New reviews received page images from both publisher sources without changing the generated implementations or the scenario results.

The Longest Road result depends on how a player's roads connect, including branches and interruptions, but the original implementation never updated the initial value. It therefore failed the tested threshold, branch, interruption, transfer and tie scenarios. The original implementation also allowed a second free road with the Road Building card after the player's road stock reached zero.

The rule emphasis implementation corrected every tested Longest Road behavior and enforced the remaining stock after the first free road. However, it did not allow Road Building to be played when no road remained, although the approved decision keeps the card playable even when no road can be placed. With three players, red is taken out of the game. The rule emphasis implementation started with only blue, orange and white, so red did not take part in the game. However, it did not list red as removed in the public state, which caused the scenario to fail. The tables keep this recorded result. In the same implementation, a player who already had ten points did not win when becoming active, so the turn continued with the dice roll.

The clarified implementation limited trade offers to the players' public hand sizes. It also protected cards already selected for discarding after a seven, when a development card interrupted the process before all players had submitted their selections. It also required maritime trades with the bank or a harbor to exchange one resource type for a different type. The implementation passed the scenarios for these three decisions.

However, the clarified implementation did not offer the Knight action before the roll, although the rules allow it. The scenario for choosing an opponent to steal from after moving the robber therefore stopped before reaching that choice and could not check whether it worked correctly.

Valid domestic trades could not be accepted because the implementation checked the offered resources against the responder's hand instead of the proposer's hand. An opponent's settlement also failed to update Longest Road, while some interrupted actions returned control to the wrong

player. These errors left the clarified implementation at 32/40 clear-rule scenarios despite its improvements in the targeted decisions.

Figure 4.4 shows the separate scenario and judge results for all three conditions.

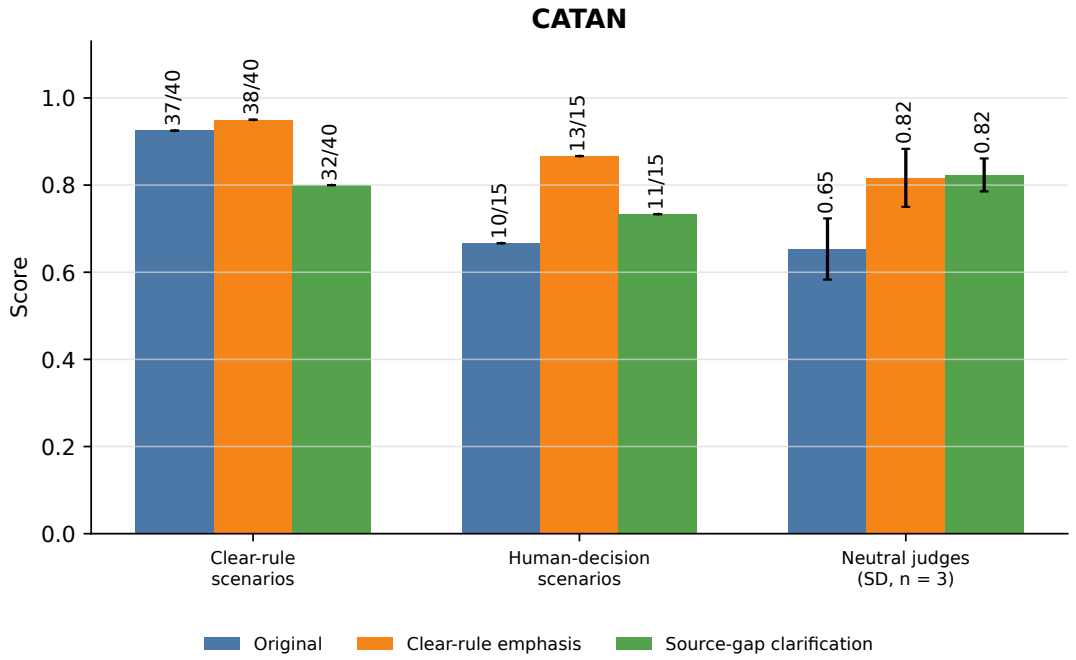


Figure 4.4: CATAN evidence profile.

The neutral judge means were 0.653 for the original, 0.817 for the rule emphasis and 0.823 for the source gap clarification. All three original judges identified the missing Longest Road updates, while the clarification judges reported problems with development card interruptions and trade acceptance. One review also reported that discard cards were already set aside before they were submitted, although the submitted discard scenario passed. The clarification received the highest judge mean, but it passed fewer clear-rule scenarios than the other two implementations.

4.2.5 Exploding Kittens

Exploding Kittens used the German publisher rulebook (Exploding Kittens, 2018) with the base game for two to five players and with the extra card combinations. The source gap clarification added that Favor and Pair may target only living opponents who hold at least one card.

	Original	Clarification
Technical checks	4/4	4/4
Robustness rollouts	98/100	100/100
Interface checks	12 436/12 436	15 589/15 589
Clear-rule scenarios	32/34	33/34
Human-decision scenarios	3/4	4/4
Neutral judge mean (sample SD)	0.813 (0.012)	0.907 (0.012)

Table 4.10: Exploding Kittens evaluation results.

	Original	Clarification
Input tokens (cached)	1 704 563 (1 460 992)	1 857 484 (1 587 712)
Output tokens (reasoning)	32 106 (13 704)	31 738 (12 726)
Estimated equivalent API cost	USD 2.91	USD 3.09
Generated Python lines	504	486

Table 4.11: Exploding Kittens resource usage.

Both implementations passed all four technical checks and all sampled action conversions, but the clarified run required one repair before evaluation because the first generation did not produce the required files. The original passed 98/100 robustness rollouts, while the clarified implementation passed 100/100. The clear-rule result increased from 32/34 to 33/34 and the human-decision scenario result from 3/4 to 4/4, while the number of Python lines decreased from 504 to 486.

Favor requires the selected opponent to give a card, while Pair lets the player take a random card from an opponent's hand. The original implementation allowed opponents with empty hands as targets for both actions. After a player used Favor against an empty hand, the game waited for a card transfer, but the selected opponent had no card to give and therefore no legal action. Both failed robustness rollouts remained waiting for this card transfer with no legal action to continue. Pair also allowed the empty target but did not cause the same waiting state. The clarified implementation excluded empty targets for both actions and passed the corresponding human-decision scenario.

When a player draws an Exploding Kitten without a Defuse, both the Exploding Kitten and the player's remaining hand cards must be discarded. The original implementation eliminated the player but left the remaining hand outside that pile. This caused two scenario failures, one during a continuing game and one after the final elimination in a game with two players. The second failure did not change the winner, but the final state still violated the discard rule. The clarified implementation handled both situations correctly. The added decision concerned empty targets, but the clarification file also stated that an eliminated player's hand must be discarded.

When an attacked player also plays Attack, the next player must take exactly two turns. The original implementation followed this rule, but the clarified implementation assigned three turns instead and therefore failed its remaining clear-rule scenario.

Figure 4.5 shows the separate scenario and judge results for both conditions.

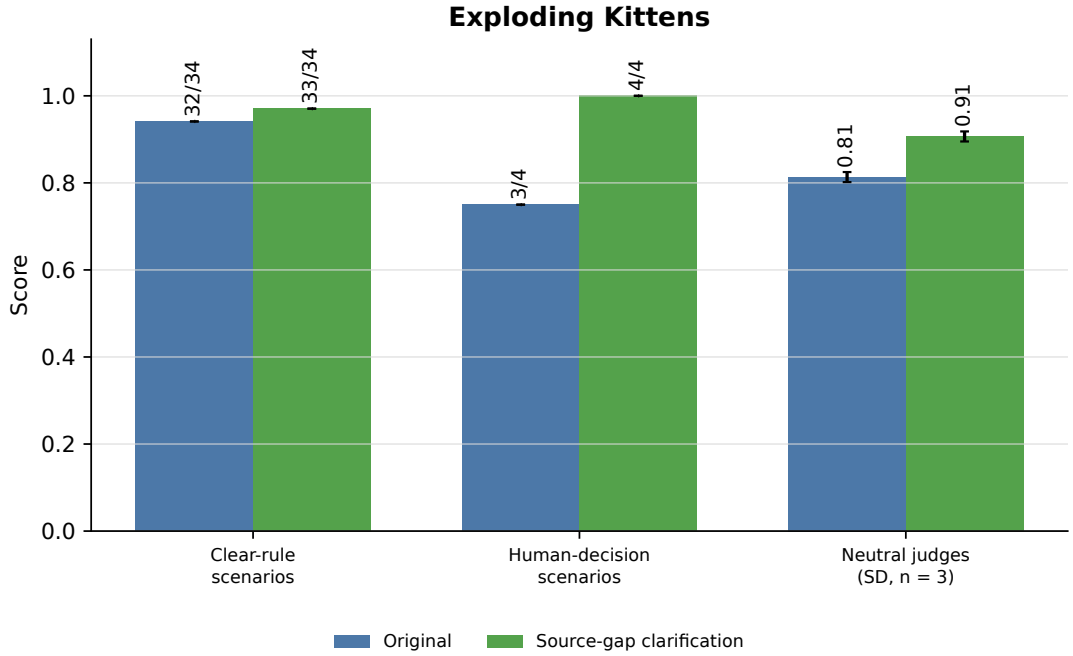


Figure 4.5: Exploding Kittens evidence profile.

The neutral judge mean increased from 0.813 to 0.907. All three original judges reported both the missing hand discard and the empty target problem, while all three clarification judges reported the incorrect Attack chain. The clarified implementation excluded empty targets and no longer waited for an impossible card transfer. It also handled elimination correctly but assigned an extra turn in the Attack chain. The added decision concerned only empty targets. The clarification file also mentions the discard requirement, but did not mention Attack chains.

4.2.6 Wizard

Wizard used the German publisher rulebook (AMIGO Spiel + Freizeit GmbH, 1996) and covered the base game for three to six players without any variants. The original condition received only the rulebook, while the clarification added seven decisions for card inventory, the first dealer, trump, bidding, the Jester Wizard sequence, tied scores and private hands.

	Original	Clarification
Technical checks	4/4	4/4
Robustness rollouts	100/100	100/100
Interface checks	226 118/226 118	226 148/226 148
Clear-rule scenarios	22/23	23/23
Human-decision scenarios	9/10	9/10
Neutral judge mean (sample SD)	0.740 (0.035)	0.897 (0.015)

Table 4.12: Wizard evaluation results.

	Original	Clarification
Input tokens (cached)	1 174 314 (968 192)	1 326 930 (1 132 800)
Output tokens (reasoning)	26 421 (10 215)	29 168 (13 446)
Estimated equivalent API cost	USD 2.31	USD 2.41
Generated Python lines	331	477

Table 4.13: Wizard resource usage.

Both implementations passed all four technical checks, passed all 100 sampled robustness rollouts and passed the sampled action conversions. The clear-rule result increased from 22/23 to 23/23, while the human-decision result remained at 9/10 and the number of Python lines increased from 331 to 477.

An earlier evaluator also required player 0 to be the first dealer, but this expectation was removed because the rulebook permits any player to be appointed and only requires the role to move clockwise after each round. Both implementations rotate the dealer correctly, so the original's random initial choice is no longer counted as a failure. The judge reviews were not repeated after this evaluation change and all three original reviews still criticize the initial dealer choice, so their reported mean includes an expectation that is no longer scored by the scenarios.

The original implementation failed when a Wizard led a trick. The first Wizard still won, but the first following ordinary card set a color. Later players then had to follow that color, although the rulebook keeps this trick colorless and every later card must remain legal. The clarified implementation kept a trick led by a Wizard colorless and allowed every later card.

Under the approved decision, a trick beginning with a Jester followed by a Wizard before any ordinary card should remain colorless, but both implementations let a later ordinary card set a color that later players had to follow. The first Wizard still won as required, so that difference affected legal card choices rather than the trick winner. This remained the single human-decision scenario that failed in both implementations.

Figure 4.6 shows the separate scenario and judge results for both conditions.

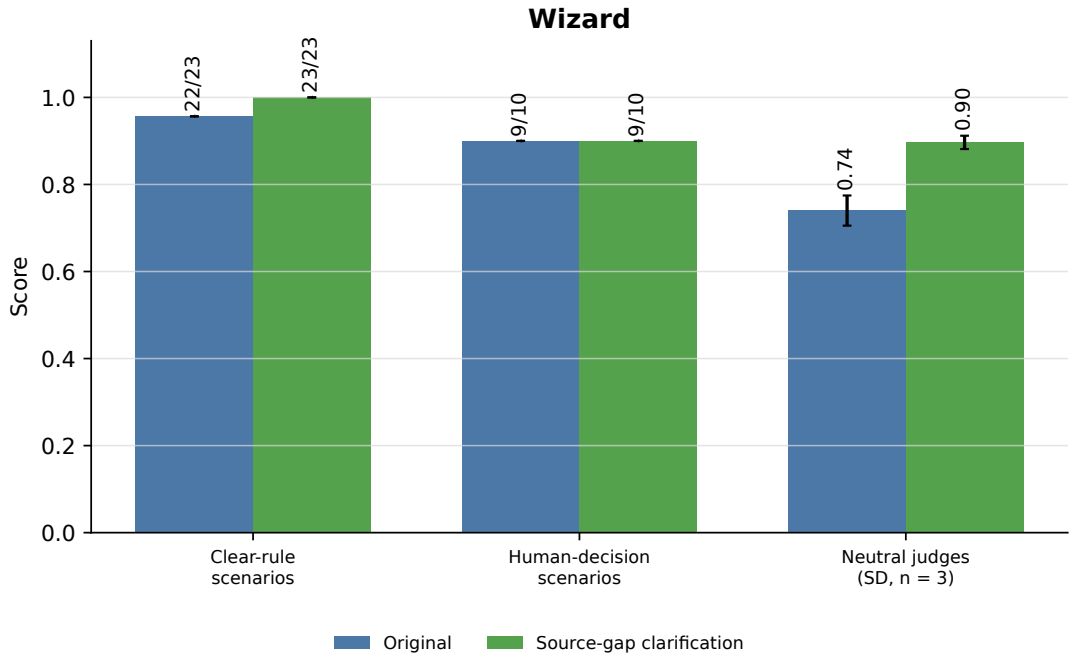


Figure 4.6: Wizard evidence profile.

The neutral judge mean increased from 0.740 to 0.897. All three original judges identified the incorrect color after a direct Wizard lead and all three clarification judges identified the remaining sequence with a leading Jester followed by a Wizard. Two original judges and one clarification judge also raised a possible information problem because cards from completed tricks did not remain visible. This finding remained unscored because the rulebook did not specify whether completed tricks had to remain visible, so no scenario had been configured for that expectation. The clarified implementation kept a direct Wizard lead colorless, while the same issue remained after a leading Jester followed by a Wizard.

Chapter 5

Discussion

This chapter discusses the problems found during rule translation, the changes after clarification and the Limitations of the evaluation. It then summarizes the findings in the Conclusion and describes possible Future work.

5.1 Rule translation problems

Abalone started with fewer marbles than required, Bohnanza used wrong bean values and CATAN did not update the Longest Road value, although the rulebooks already defined these requirements. The model did not need an additional decision for these rules, but it still had to find the information and use it correctly in the implementation. The Bohnanza rulebook showed the beanometers in small card images, while the physical cards were not supplied separately. The model had to read the numbers from these images and use them as harvest values, just as Abalone required reading the illustrated starting position. The wrong values and setup show that this translation failed, but the code alone does not show whether the model read the images incorrectly or made a later implementation mistake.

The Bohnanza rule emphasis implementation also allowed the optional second hand card to be planted but did not then advance to the reveal phase as required. In the CATAN clarification run, some interrupted actions returned control to the wrong player instead of continuing the interrupted turn correctly. These failures concerned how actions were connected to the next phase or player, not simply whether the individual actions existed.

The original Exploding Kittens implementation allowed Favor to target an empty hand, but then waited for a card transfer with no legal action to continue. The clarified implementation excluded these targets and removed the problem, but unlike the earlier examples, the rulebook did not clearly require this particular solution.

The Abalone setup emphasis repeated a clear rule, while the decision for empty targets supplied information that the publisher did not clearly provide. A different interpretation could fail the human-decision scenario without contradicting the publisher rule, while a clarified implementation is expected to follow the supplied decision.

In 6 nimmt!, an error affected what players could see, because a selected card was missing from the shared observation although all selected cards should have been visible together. The judges of the structured Bohnanza clarification reported the opposite problem, because available trade actions showed cards from opponents' private hands. A replay confirmed the first finding, while the second remained a judge finding in the reported evaluation. Checking turns and scores alone would miss whether players receive the public information they need or can access information that should remain hidden.

The CATAN emphasis implementation used the required three player colors, but did not list red as removed in the public state. The corresponding scenario failed because the required entry was missing, although red did not take part in the game. Unlike the 6 nimmt! error, this

failure concerned the required state representation rather than information that players should see during normal play. The game data contract allows the evaluator to use the same state fields across implementations of the same game, but a missing entry does not by itself show that a game rule was implemented incorrectly.

5.2 Corrections and new errors

The Bohnanza emphasis implementation corrected unequal trades and the Garden and Soy bean values, but also restricted the planting order of received and revealed cards and used an incorrect Red bean value. A second generation with the same emphasis reached the same clear-rule result through different failures, so the same score could hide different rule errors. The additional generations with the structured clarification also differed in their scenario results, so variation was not limited to the first emphasis.

The original Exploding Kittens implementation did not discard the remaining hand cards from an eliminated player, while the clarified implementation did. The added decision concerned empty Favor and Pair targets, but the same file also repeated the existing discard rule from the rulebook. The model received this extra reminder and the later implementation handled discarding correctly, although the reminder's contribution to that correction is uncertain. The later implementation also assigned three turns instead of two in an Attack chain that the original had handled correctly. Its higher pass rates in both clear-rule and human-decision scenarios reflect improvements in the tested rules alongside the new Attack error.

In the original Wizard implementation, a later ordinary card set the led color both after a direct Wizard lead and after a leading Jester followed by a Wizard. The rulebook already requires a trick led by a Wizard to stay colorless. The clarification file mentioned this rule and added the decision that the sequence with a leading Jester followed by a Wizard must also stay colorless. The clarified implementation corrected the direct Wizard lead, but still restricted later card choices in the second case. That error was already present in the original and remained even after the model received a decision explaining how to handle it.

The later runs corrected problems, including the Abalone setup and the empty Favor target in Exploding Kittens. The first Abalone rule emphasis implementation contained 344 Python lines rather than 245, although the only scored correction was the starting marble count. These cases show targeted corrections after additional explanations were supplied, but the comparison cannot separate their effect from variation between fresh generations or show how reliably the corrections would recur.

5.3 Scenario configuration and interpretation

A scenario can fail even when the implementation follows the tested rule if it expects the wrong result, starts from an unsuitable state or checks the wrong action. It can also pass despite a rule error if it misses an important part of the rule. The evaluator therefore needs checks of both the expectation, including any approved decision, and the actions and states used to test it.

The Exploding Kittens rulebook allows a combination of five cards with different card titles, including action cards rather than only cat cards. A scenario that accepted only five different cat cards would reject a legal move. Whether playing those action cards is a good move does not change whether the rules allow it. This illustrates why an evaluator must check what the source permits rather than limit play to the choices that seem useful or familiar. Calling the restriction a human decision would not justify it because the source already states which cards can be combined.

In Bohnanza, the problem was how legal actions were offered rather than which cards could be played. One scenario required separate harvest and plant actions, but the implementation also allowed the player to choose a field, harvest it and then plant within one action. Later checks

showed that the player could still choose which field to harvest and that the steps happened in the correct order for the tested situations. The scenario failed because it expected separate actions, not because those checks found a rule error. A combined action still needs to preserve the player's choices and the effects of each step, but it should not fail only because it represents the same legal sequence differently.

Unlike the Bohnanza scenario, which rejected a legal sequence, the CATAN scenario for choosing an opponent to steal from stopped because a required earlier action was missing. The original implementation passed the scenarios for playing a Knight before the roll and requiring a victim choice after moving the robber. The clarification file explicitly required choosing an adjacent opponent when one was present, while playing before the roll was already allowed by the publisher rules. The clarified implementation no longer allowed the Knight to be played before the roll, so the victim scenario failed at this earlier step and never reached the choice. The failure therefore shows a new error in when the Knight could be played, while the intended victim choice remained untested in this execution. Interpreting the failure correctly requires checking the step that failed, not only the scenario title and result.

An earlier Abalone evaluator required a forced pass when a player had no legal moves, although the rulebook leaves this situation open. Removing this expectation prevented an unspecified rule from being scored as an implementation error. Wizard's initial dealer choice was also no longer counted as an error because the rulebook did not require one fixed initial dealer. These changes improved the evaluation rather than changing game code. The earlier judge reviews still include these criticisms because they received the approved decisions for a forced pass and a fixed first dealer and were not repeated after the scenario evaluation changed.

The 6 nimmt! result shows the opposite problem, because all configured scenarios passed even though a later replay showed that a revealed card was missing from the shared observation. The scenarios had not checked enough of the reveal sequence to detect this error, so a passed scenario does not show that the relevant player views were checked.

Two Exploding Kittens scenarios failed because of the same discard error, while several Bohnanza scenarios found the same restriction on planting order. These scenarios check different situations, but within each game several scenario failures reflect the same error. The pass rates summarize the evaluated scenarios for each game, while differences in those scenarios limit comparison of overall rule fidelity between games. Clear-rule scenarios check publisher rules, while human-decision scenarios check the interpretations chosen for the thesis, so their pass rates remain separate.

5.4 Technical checks, rollouts and LLM reviews

The Bohnanza and CATAN implementations passed the technical checks and sampled robustness rollouts despite their errors in phases, trades and scoring. The technical and interface checks confirmed that the modules could be used through the tested interface, but not that their actions and scores followed the rules. The scope of these checks is described in Section 3.3.4.

Random play found the Favor problem in Exploding Kittens because the game waited for a card from an empty hand and had no legal action to continue. Successful rollouts provide evidence that the sampled play ran without any detected runtime errors. Other rule errors can leave the game running, and reaching the step limit also counts as a pass when no checked runtime error is detected. The pass rate describes how many sampled rollouts passed the runtime checks, not how many games finished and not whether every required action was available.

The judges identified the visibility error in 6 nimmt! that the scenarios missed, showing how reviewing the source and code can complement executable tests. However, the judges also agreed on unsupported expectations in Abalone and Wizard, so even findings shared by several judges need to be checked against the source and, if possible, through a replay. Differences between the three judge scores likewise describe disagreement between reviews, not certainty

that the implementation follows the rules. Keeping reviews, scenarios and technical checks separate makes their contributions and disagreements visible and allows one check to find errors that another missed.

5.5 Limitations

The six games provide examples of translation problems across different mechanics, but the small selection does not support broad conclusions about board games in general. Their rulebooks and supplied components also differ, making it difficult to separate the influence of game complexity from the information available to the model.

The technical results describe the final implementation after any allowed repairs, not how often the first attempt worked. Earlier pilots used different procedures and are described in Workflow development and pilot runs.

Implementation and judging used the same model with different thinking levels, so separate calls could still make similar mistakes or overlook the same error. Other models or different model versions may perform differently across game types. The numerical judge scores are also not calibrated measurements of rule fidelity. They summarize the individual reviews, but small score differences may result from variation between judge runs and are not treated as precise performance differences.

Rule analysis and scenario preparation also used model assistance together with human review, but there was no second person to check every claim independently. Some scenarios started from states prepared by adapters, and legal paths to all of these states were not verified. The scenarios checked behavior from the prepared states, while their relevance to normal play depends on whether those states can occur. Correcting the known evaluator errors improved the tests, although further review may still identify other problems.

The evaluation also depends on approved human decisions, which define expectations for this study but are not publisher rules. The original implementation runs did not receive these decisions as additional input, while later runs received only the decisions included in their clarification files. A failure against an unsupplied decision can therefore reflect a different interpretation rather than a contradiction of the publisher rules.

This thesis used German publisher PDFs and page images without comparing languages, editions or the model's previous knowledge of each game. Correct code may come from the supplied rules, the model's existing knowledge or both. Working with other sources or games may also require collecting component information not included in the rulebook and reviewing claims, decisions and scenarios. The resource tables record model usage and estimated equivalent API costs, but not the human preparation effort or the API costs of model assistance during preparation.

The final games were evaluated through BoardBench's interface and rule checks, without a comparison to a fully checked implementation or a matching OpenSpiel environment. This thesis did not train agents in these games, so it does not show whether they are suitable for reinforcement learning.

5.6 Conclusion

This thesis found several types of problems during the translation from rulebooks to executable Python environments. These included incorrect setup, missing or incorrect legal actions, phase transition errors, scoring errors and problems with the information available to players. Other situations needed an additional decision because the source did not clearly explain what should happen. Checking the source, the implementation and the scenario expectation separately helped distinguish missing information from errors in the code or evaluator.

Later implementations corrected some of the targeted problems, but other errors remained or new errors appeared. Because most conditions were generated only once, these comparisons do not reliably show how an explanation causes a correction.

The evaluation also showed that technical success and rule fidelity need to be considered separately. Implementations could pass the technical checks and random rollouts while still violating game rules, while some failed scenarios were caused by incorrect evaluator expectations rather than incorrect game code. BoardBench keeps sources, code, scenarios and reviews separate so these differences can be examined. This provides a way to locate translation and evaluation problems and to document both corrections and remaining errors.

5.7 Future work

More games could help show which game mechanics the agent or different models handle well and which cause problems more often. Several games could be selected for each feature, such as random events, hidden information, simultaneous choices or actions that interrupt a turn. Complexity would also need a clear description, for example through the number of phases and how rules depend on each other. Each game would need repeated generations so that one poor implementation does not stand for a whole type of game. Missing component information and differences between the rulebooks should also be recorded, so that these problems are not all treated as game complexity.

The corrections observed in this study could be tested through repeated generations under the same original and clarified conditions. This would show how often a selected explanation is followed, how often the target problem is corrected and how often other rules change. For a controlled comparison, source gaps should be selected before the implementation results are known, while the model, programming instructions, original sources, repair limits and evaluator stay the same. Each run should record its errors, scenario results and repair count. The number of runs and the stopping rule should be decided beforehand, with every completed result reported. Other implementation models could then be compared using the same procedure. Reviews by judges from different model families could also be compared to identify findings one model misses and disagreements that require source checks.

The agent can already run code and make its own checks, but another experiment could require it to play the game it creates, before submitting the implementation for evaluation. During play, it would compare available actions, state changes and player observations with the rulebook and repair errors it finds. Runs with and without this requirement should use the same sources and comparable resource limits, while the final evaluator scenarios remain hidden. Recording the detected errors and checking the final implementation would show whether this requirement improves rule fidelity. Independent evaluation remains important because the agent may overlook missing legal actions or make assumptions about situations that the source does not define.

Another question concerns how the rules are presented. In the structured Bohnanza clarification, additional information was added in a different format. A separate comparison could therefore keep the approved facts unchanged and add only a checked text version of the beanometer values alongside the unchanged PDF and page images. This would test whether making existing information easier to find and read improves how accurately it is implemented. The Exploding Kittens discard reminder also shows that the full added file should be checked for additional rule reminders, not only for the main decisions it was intended to clarify.

The comparisons above would still generate a new environment each time, but an agent could instead use the rulebook and an approved clarification to repair an existing implementation. Giving it failed scenarios or detailed evaluator output would add another source of feedback. Repair using this feedback should therefore be compared separately from repair using only code, rulebook and clarification. The final evaluation should also use scenarios that were not shown

during repair. The original implementation and results should remain available and the number of allowed repair attempts should be decided before each comparison.

To check whether prepared scenario states can occur during play, each scenario could record a legal sequence of actions from the normal setup to its starting state. For games with random events, the record would also need the outcomes required to reproduce that state. To test whether a scenario detects incorrect behavior, a known rule error could be deliberately added to the implementation. Conversely, checking equivalent action sequences could show whether a scenario rejects legal play only because the implementation combines or separates the steps differently. A second person could also review selected claims and scenarios independently, while disagreements are documented rather than resolved automatically.

Beyond these experiments, BoardBench could help create game environments for training reinforcement learning agents or for contributing to OpenSpiel. Before either use, the generated environments would need independent rule checks and an OpenSpiel contribution would also need to meet its interface and contribution requirements. In the OpenSpiel case, BoardBench could support contributions of new game environments rather than use OpenSpiel as an evaluation reference.

Appendix A

Used Language Models

The following models were used in this thesis.

Model	Used for
GPT-5.6-sol	game environments, rule analysis, LLM reviews, repository work
GPT-5.5	game environments, rule analysis, LLM reviews, repository work
Claude Opus 4.8	game environments, rule analysis, LLM reviews
GLM-5.2	game environments, rule analysis, LLM reviews
GPT-5.4	repository work
Composer 2.5	repository work
Grok 4.6	repository work
Grok 4.5	repository work

Table A.1: Language models used in this thesis.

Appendix B

LLM Prompts

B.1 Rulebook Analysis and Scenario Preparation Prompt

```
# Rulebook to BoardBench scenario candidates

Extract candidate black-box scenarios for ‘checks/run_scenarios.py‘ from the assigned source packet.

Rules:

- Use only the assigned sources; do not use remembered or web game knowledge.
- Every scenario needs a source ID, positive PDF page or JSON Pointer, and direct evidence. When an
    expectation genuinely combines pages or assigned sources, include ‘supporting_sources‘ with the same
    locator/quote shape.
- Mark ambiguous rules as questions instead of silently choosing an interpretation.
- Prefer observable assertions through the public BoardBench API: current player, terminal state, returns,
    absolute/relative legal-action count (‘legal_action_count‘ / ‘legal_action_delta‘), exact action name,
     or normalized action name.
- Start with setup, ordinary turn flow, occupied/illegal targets, and terminal conditions.
- For every card game, include cited inventory facts and a scenario that checks the exact total and per-
    card-type counts after setup; mark only genuinely unobservable portions ‘not_testable‘.
- Do not inspect or mutate private fields of a generated state.
- A human or independent second model must validate quotes and expected values before the scenario becomes
    benchmark evidence.

Return:

1. a short list of atomic rule facts (‘id‘, page, quote, interpretation, ambiguity status),
2. JSON candidates matching the version-3 format in ‘checks/scenarios/README.md‘,
3. rules that cannot be tested through the current public API and why.
```

B.2 Implementation Task

```
Implement the provided rulebook as one self-contained Python module.

Source rules:
- Use only the supplied rulebook as game-rule evidence.
- Do not use remembered or web rules.
- Do not invent missing rules. Keep unresolved gaps explicit.

Implementation rules:
- Read ‘ENVIRONMENT_CONTRACT.md‘ and ‘GAME_PROFILE.json‘ completely. They define evaluator representation
    only and are not additional rule sources.
- Standard library only; no files, network, subprocesses, API keys, or input prompts at runtime.
- Keep the code small and readable.
- Model setup, phases, legal actions, transitions, chance/private information, terminal conditions, and
    returns explicitly when required.
- Terminal states have no legal actions.

Provide ‘GameState‘ and ‘Game‘ with:
- ‘initial_state()‘
```

```
- `current_player(state)`
- `legal_actions(state)`
- `apply_action(state, action)`
- `is_terminal(state)`
- `returns(state)`
- `render(state)`
- `action_to_name(action)`
- `name_to_action(name)`
- `state_to_data(state)`
- `state_from_data(payload)`
- `action_to_data(action)`
- `action_from_data(payload)`
- `observation_to_data(state, player)`

`Game` must accept optional `num_players` and `seed` keyword arguments; a supplied seed controls every chance operation. Canonical state/action/observation payloads must follow `ENVIRONMENT_CONTRACT.md` and `GAME_PROFILE.json` exactly. Every legal action must be accepted by `apply_action`. Action names must be unique, stable, human-readable, reversible, and preserve source labels.

Work on the actual `implementation.py`. Before finishing, run:

```text
python -m py_compile implementation.py
python agentic_self_check.py
```

Do not modify `agentic_self_check.py`.

Create `rule_coverage.md`. Map every supplied section and named rule/card/combination to its implementing symbol, a source-only probe or reason it was not probed, and any assumption.

Create `assumptions.json` with exactly this shape:

```json
{
  "version": 1,
  "assumptions": [
    {
      "id": "A-01",
      "material": true,
      "source_location": "page/section or exact heading",
      "source_basis": "ambiguous",
      "alternatives": ["plausible behavior A", "plausible behavior B"],
      "selected": "implemented behavior",
      "affected_mechanics": ["legal_actions", "state_transition"]
    }
  ]
}
```

Allowed `source_basis` values are `ambiguous`, `missing`, and `conflicting`. Every `selected` value must exactly equal one complete string in the same assumption's `alternatives` array; put implementation detail in that selected alternative rather than paraphrasing it. Include only material assumptions that affect legal actions, transitions, private information, elimination, terminal results, or scoring. Use an empty list when there are none. This is a source audit, not evaluator access.

Final response only:
1. brief open questions/material assumptions;
2. files changed: `implementation.py`, `rule_coverage.md`, `assumptions.json`;
3. exact validation commands and outcomes.

Do not repeat the complete module in the final response.
```

B.3 Environment Contract

```
# BoardBench canonical environment contract v2

This file defines evaluator infrastructure, not game rules. The supplied rule sources remain authoritative for behavior.

```

````
Keep the existing public methods:

```text
initial_state, current_player, legal_actions, apply_action, is_terminal,
returns, render, action_to_name, name_to_action
```

Use a reproducible constructor `Game(num_players=None, seed=None)`. A supplied seed must determine all chance behavior. `GAME_PROFILE.json.player_counts.supported` is the source-approved player-count range: accept every listed count, apply its count-specific setup, and raise `ValueError` for listed unsupported counts.

Add these five methods:

```python
Game.state_to_data(state) -> dict
Game.state_from_data(payload: dict) -> GameState
Game.action_to_data(action) -> dict
Game.action_from_data(payload: dict) -> Action
Game.observation_to_data(state, player: int) -> dict
```

## Universal envelopes

State:

```json
{"schema": "boardbench/<game-slug>/state/1", "data": {}}
```

Action:

```json
{"schema": "boardbench/<game-slug>/action/1", "data": {"type": "stable_action_type"}}
```

Player observation:

```json
{"schema": "boardbench/<game-slug>/observation/1", "data": {}}
```

The root object has exactly `schema` and `data`. `data` is an object. Action `data.type` is a nonempty stable snake-case identifier. Follow the supplied `GAME_PROFILE.json` for the exact game-specific state fields, action vocabulary, types, and ordering.

## Required properties

- Payloads contain only JSON-domain values: `None`, booleans, integers, finite floats, strings, lists, and string-keyed dictionaries. No tuples, sets, enums, bytes, custom objects, NaN, or infinity.
- Returned payloads are detached copies. Mutating them must not mutate the original state or action.
- `state_to_data` exposes the complete privileged evaluator state, including private information and everything that can affect future behavior. It is not a player observation API.
- `observation_to_data` is the player-specific public view. It may expose the selected player's private information but must hide other players' private information according to the source.
- Required profile fields are always present. Reject unknown, missing, wrongly typed, or invalid fields instead of silently filling defaults or accepting aliases.
- `state_from_data` reconstructs every complete, schema-valid profile payload, including evaluator fixtures that are not naturally reached during a rollout. `zones.reserve` holds inventory intentionally removed from active play in such fixtures. Validate schema and types, but do not reject a payload merely because its phase/zones are unusual. It must not merge a sparse patch into `initial_state()`.
- Every legal action has unique canonical action data. `action_from_data` rejects unknown types and invalid parameters.
- Human-readable names remain separate from canonical machine data.

## Round trips

For every checked state and legal action:

```python
state_data = game.state_to_data(state)
rebuilt_state = game.state_from_data(state_data)
assert game.state_to_data(rebuilt_state) == state_data
````
````

````
65
66  action_data = game.action_to_data(action)
67  rebuilt_action = game.action_from_data(action_data)
68  assert game.action_to_data(rebuilt_action) == action_data
69  ```
70
71  The rebuilt state must agree on `current_player`, `is_terminal`, `returns`, canonical legal actions, and
        sampled successor states. Before finishing, also run `python profile_fixture_self_check.py`; this
        checks complete phase, pending-decision, zone, third-field, and five-player payloads without exposing
        rule expectations.
````

B.4 Independent Rule Review Prompt

````
1   # BoardBench rule review
2
3   Review one generated module against the complete supplied source condition and its provenance labels. Do
        not edit code, run generic checks, or use outside game knowledge.
4
5   ## Evidence rule
6
7   For every `critical` or `major` finding include:
8
9   - canonical fact ID;
10  - evidence type: `rule_quote`, `user_observation`, or `human_decision`;
11  - source ID, stable locator (PDF page or JSON Pointer), and exact source evidence;
12  - conflicting code symbol or transition;
13  - expected versus implemented behaviour.
14
15  Verify the complete approved fact before scoring. Do not penalize behaviour that the approved fact permits,
         including a five-card combination retrieving one of its own just-discarded components. If the packet
        does not decide the issue, use `question`; do not treat remembered rules as truth. Keep adjudication-
        dependent deviations visibly separate from contradictions of clear printed rules.
16
17  Severity:
18
19  - `critical`: core game cannot complete reliably, common crash/deadlock, or fundamentally wrong winner/game
        ;
20  - `major`: material setup, action, phase, chance, information, or terminal rule is absent/contradicted;
21  - `minor`: localized issue unlikely to change core flow or winner;
22  - `question`: rulebook evidence is insufficient or ambiguous.
23
24  ## Review
25
26  Check setup, turn flow, legal actions, transitions, chance/private information, terminal conditions,
        returns, and unsupported assumptions.
27
28  Return:
29
30  1. `score: 0.0-1.0` and `confidence: low|medium|high` with a brief rationale;
31  2. findings ordered by severity;
32  3. compact rule-area coverage table;
33  4. missing deterministic scenarios;
34  5. material questions for a human.
35
36  End exactly with:
37
38  ```text
39  score: <0.0-1.0>
40  confidence: low|medium|high
41  critical_issues: <number>
42  major_issues: <number>
43  minor_issues: <number>
44  needs_rulebook_clarification: true|false
45  needs_code_change: true|false
46  needs_more_tests: true|false
47  ```
````

Appendix C

Additional Source Material

C.1 6 nimmt! Component Observation

```
{
  "version": 1,
  "game": "6_nimmt",
  "role": "user_observation",
  "authorship": "User-confirmed physical component observation",
  "approved_on": "2026-07-31",
  "scope": "Component identity and printed bullhead inventory only; this artifact does not define or
     override gameplay rules.",
  "cards": {
    "total": 104,
    "identities": {
      "kind": "integer_range",
      "minimum": 1,
      "maximum": 104,
      "copies_each": 1
    },
    "bullheads": {
      "55": 7,
      "other_repeated_digit_values": 5,
      "other_multiples_of_10": 3,
      "other_multiples_of_5": 2,
      "all_remaining_values": 1
    }
  }
}
```

Appendix D

Experimental Interventions

D.1 Abalone Setup Emphasis

```
{
  "version": 1,
  "game": "abalone",
  "intervention_kind": "clear_rule_emphasis",
  "authorship": "BoardBench experimenter emphasis approved by the user after evaluation of v2_original_1 on 2026-07-26; not publisher text and not a source-gap clarification",
  "trigger": {
    "original_scenario": "ABAL-R01-exact-initial-setup",
    "original_result": "FAIL",
    "observed_behavior": "The implementation placed 13 black and 13 white marbles instead of 14 each."
  },
  "emphasis": [
    {
      "id": "ABAL-EMPH-SETUP",
      "claim_ids": ["ABAL-C-SETUP-FIGURE", "ABAL-C-BOARD-61", "ABAL-C-SETUP-COUNTS", "ABAL-C-SETUP-ROWS"],
      "publisher_locator": "game_rules.pdf page 1, Figure 1",
      "text": "Implement Figure 1 exactly: the board has 61 playable pits; the initial position has 14 black marbles, 14 white marbles, and 33 empty pits. Top-to-bottom rows are BBBBB / BBBBBB / ..BBB.. / empty / empty / empty / ..WWW.. / WWWWWW / WWWWW. Do not omit either outer marble from either six-marble row."
    }
  ]
}
```

D.2 Bohnanza Clear-Rule Emphasis

```
{
  "version": 1,
  "game": "bohnanza_base_2023",
  "intervention_kind": "clear_rule_emphasis",
  "authorship": "BoardBench experimenter emphasis approved by the user after evaluation of v2_original_1 on 2026-07-28; not publisher text and not a source-gap clarification",
  "guardrail": "This artifact repeats only publisher-clear rules that the Original implementation violated. It does not support a claim that the rulebook was missing these rules; it tests whether making already-clear rules more salient changes a fresh generation.",
  "emphasis": [
    {
      "id": "BOHN-EMPH-UNEQUAL-MULTICARD-TRADE",
      "claim_ids": ["BOHN-C-TRADE-ANY-HAND-POSITION", "BOHN-C-TRADE-UNEQUAL", "BOHN-C-TRADE-CONSENT", "BOHN-C-TRADE-TRANSFER-ON-ACCEPT"],
      "publisher_locator": "game_rules.pdf page 2, Regeln für den Bohnenhandel",
      "text": "A single agreed trade may contain multiple cards on either side and the bundle sizes may differ, for example two cards for one. Cards may come from arbitrary hand positions. Represent the complete proposed bundles atomically, leave both hands unchanged while awaiting consent, and transfer every referenced card only after acceptance. Do not reduce source-legal multi-card bundles to single-card-only proposals."
    },
```

```
    {
      "id": "BOHN-EMPH-GARDEN-METER",
      "claim_ids": ["BOHN-C-PAYOUT-GARTEN"],
      "publisher_locator": "game_rules.pdf page 1, Gartenbohne card Bohnometer",
      "text": "Apply the printed Garden Bean meter exactly: 1 Garden Bean pays 0 coins; 2 Garden Beans pay 2 coins; 3 or more Garden Beans pay 3 coins. There is no 1-coin Garden payout."
    },
    {
      "id": "BOHN-EMPH-SOY-METER",
      "claim_ids": ["BOHN-C-PAYOUT-SOJA"],
      "publisher_locator": "game_rules.pdf page 1, Sojabohne card Bohnometer",
      "text": "Apply the printed Soy Bean thresholds exactly: sizes 0-1 pay 0; 2-3 pay 1; 4-5 pay 2; size 6 pays 3; size 7 or more pays 4."
    },
    {
      "id": "BOHN-EMPH-THIRD-DEPLETION-PHASE2-END",
      "claim_ids": ["BOHN-C-END-THIRD", "BOHN-C-END-PHASE2-CONTINUE", "BOHN-C-FINAL-HARVEST"],
      "publisher_locator": "game_rules.pdf page 2, Ende des Spiels",
      "text": "If the draw pile becomes empty for the third time while revealing in phase 2, finish phases 2 and 3 only. As soon as all required phase-3 planting is complete, perform the final harvest and make the state terminal. Do not enter or require a phase-4 draw."
    }
  ]
}
```

D.3 Bohnanza Structured Clarification

```
# Bohnanza Base Game 2023 - structured clarification and whole-game fidelity guide

## Provenance and use

This is a user-approved BoardBench experimenter supplement, not publisher text. Read it together with the unchanged publisher PDF. The PDF remains authoritative for every rule not explicitly clarified below. The checklist deliberately covers the whole game: do not optimize only one highlighted mechanic or treat an omitted PDF rule as optional.

Before finishing the implementation, trace every checklist item to concrete state, legal-action and transition logic. Check interactions at phase boundaries and game end, not only isolated helper functions.

## Clarified digital decisions

1. **Starting player:** use the supplied seed to select the starting player reproducibly. That player keeps the Start card for the whole game.
2. **Phase-three player order:** when several players have sideways staged cards, any affected player may plant next. Every affected owner chooses the order of their own cards. All staged cards must be planted before phase four.
3. **Anytime harvesting:** expose each player's source-legal harvest choices at stable decision boundaries, including off-turn boundaries. Do not interrupt one atomic draw, shuffle, transfer or planting transition.
4. **Private information:** a player sees their own complete ordered hand. For each opponent expose hand size and the source-visible front card, but hide deeper card identities. Legal-action data must not leak those deeper identities.

## Balanced whole-game implementation checklist

### Setup and information

- Support exactly 3-5 players; reject player counts outside that range.
- Use the complete 104-card base deck: Garden 6, Red 8, Black-eyed 10, Soy 12, Green 14, Stink 16, Chili 18, Blue 20.
- Deal five ordered hand cards to every player. Three-player games give every player three fields; four- and five-player games give every player two fields.
- Preserve hand order: plant only from the front, never sort, and append later draws at the back in draw order.
- Each opponent's front card is public; deeper opponent cards remain private as clarified above.

### Four-phase turn

```

```
1. **Plant from hand:** the first front card is mandatory. The newly exposed front card is optional as a second planting. Never allow a third hand planting. An empty hand skips this phase. If a mandatory bean fits no field, require a separate legal harvest choice before planting; do not silently combine harvest and plant.
2. **Reveal and trade:** reveal two cards. The active player owns them initially. Every trade includes the active player; two non-active players cannot trade with each other. A proposed trade may contain any positive number of cards on either side, may be unequal, and may use arbitrary hand positions plus the active player's revealed cards. Neither hand changes before both participants consent. On acceptance transfer every referenced card atomically. Gifts also require recipient consent. Field cards and already received sideways cards cannot be traded. The active player explicitly ends trading.
3. **Plant staged cards:** every received sideways card and every active-player revealed card not traded away must be planted. Received cards never enter a hand. Follow the clarified inter-player and per-owner order above. A card that fits no field requires a separate harvest choice before planting continues.
4. **Draw:** the active player draws three cards sequentially and appends them unchanged to the back of the hand, then play advances clockwise.

### Fields and harvesting

- A field contains one bean type, but the same type may occupy multiple fields.
- Harvesting is legal at the stable boundaries clarified above, including off-turn harvesting.
- Singleton protection: a one-card field cannot be harvested while another field of the same player has more than one card; it may be harvested when none does.
- A legal harvest empties the field. The payout number of cards becomes coins and every other harvested card enters the discard pile. Zero-coin harvesting remains legal when singleton protection permits it.
- Implement every printed beanometer exactly:
  - Garden: size 1 → 0; 2 → 2; 3+ → 3.
  - Red: thresholds 2/3/4/5 pay 1/2/3/4.
  - Black-eyed: thresholds 2/4/5/6 pay 1/2/3/4.
  - Soy: thresholds 2/4/6/7 pay 1/2/3/4.
  - Green: thresholds 3/5/6/7 pay 1/2/3/4.
  - Stink: thresholds 3/5/7/8 pay 1/2/3/4.
  - Chili: thresholds 3/6/8/9 pay 1/2/3/4.
  - Blue: thresholds 4/6/8/10 pay 1/2/3/4.

### Depletion and game end

- On the first and second depletion, immediately shuffle the discard pile into a new face-down draw pile and continue the interrupted multi-card reveal or draw when cards are available.
- The third depletion ends the game immediately except during phase-two reveal. If it occurs during phase two, complete phases two and three, then end without phase four.
- At game end, harvest every field of every player before calculating results. Hand cards do not score. Each coin-pile card is one coin; highest total wins. Among tied leaders, the tied player farthest clockwise from the fixed Start-card holder wins.

## Final consistency audit

Verify setup for 3, 4 and 5 players; one complete normal turn; optional second planting; forced harvest as a separate decision; unrestricted positive trade-bundle sizes and consent; every owner's planting choices; off-turn harvest availability; all eight beanometers; first/second recycling during interrupted draws; third depletion inside and outside phase two; final harvest, coins and tie-break; and private observations/legal actions. This audit is source-derived guidance, not evaluator output.
```

D.4 CATAN Clear-Rule Emphasis

```
{
  "version": 1,
  "game": "catan_2022",
  "intervention_kind": "clear_rule_emphasis",
  "authorship": "BoardBench experimenter emphasis approved after evaluation of v2_original_1; not publisher text and not a source-gap clarification",
  "guardrail": "This artifact repeats publisher-clear rules that the Original implementation omitted or violated. The unchanged rulebook and matching Almanac remain authoritative in full. Do not optimize only these bullets or neglect any other supplied rule.",
  "emphasis": [
    {
      "id": "CATAN-EMPH-LONGEST-ROAD",
      "publisher_locators": [
        "game_rules.pdf page 3, Längste Handelsstraße",
        "game_almanac.pdf page 8, Längste Handelsstraße"
```

```
      ],
      "text": "Implement and invoke a complete Longest Road recalculation whenever a road is placed or a settlement/city can interrupt a route. Measure one player's maximum continuous route without summing branches; an opponent building interrupts, an own building does not. At length at least five, award two points. A strictly longer route transfers the card immediately. After an interruption, a tied incumbent retains it; a leading tie excluding the incumbent leaves it unowned; no qualifying route also leaves it unowned. Keep owner and measured length synchronized, and check immediate victory after every resulting award change."
    },
    {
      "id": "CATAN-EMPH-ROAD-STOCK",
      "publisher_locators": [
        "game_almanac.pdf page 6, Bauen and Fortschrittskarten/Straßenbau",
        "game_almanac.pdf page 23, Spielfiguren"
      ],
      "text": "Physical road stock applies to paid roads and to both free Road Building placements. Never expose or apply a road action when that player's road stock is zero, never permit negative stock, and after each free placement recompute whether another normal-legal road remains feasible. Road Building places no more physical roads than are actually available."
    }
  ],
  "final_audit": "Before finishing, exercise threshold, branch, own/opponent interruption, longer-route transfer, both interruption tie outcomes, loop handling under the supplied profile, zero/one/two remaining road pieces, and victory triggered by a newly awarded Longest Road. Then audit the rest of the supplied game normally."
}
```

D.5 CATAN Source-Gap Clarification

```
{
  "version": 1,
  "game": "catan_2022",
  "intervention_kind": "source_gap_clarification",
  "authorship": "User-approved BoardBench digital clarification after evaluation of v2_original_1; not publisher text and not a clear-rule emphasis",
  "guardrail": "This artifact resolves only source-underspecified digital procedures. The unchanged publisher rulebook and matching Almanac remain authoritative for every clear rule. It does not repeat or prioritize the Original implementation's clear-rule failures.",
  "clarifications": [
    {
      "id": "CATAN-CLAR-TRADE-BOUND",
      "source_gap": "The tabletop sources permit freely negotiated quantities but prescribe no finite digital offer-building bound.",
      "decision": "During bilateral offer construction, the total number of cards on the give side may not exceed the active player's public resource-hand size, and the total on the take side may not exceed the partner's public resource-hand size. Resource types remain freely nameable so legal actions do not disclose private identities. Proposal requires both totals positive. Acceptance alone checks actual per-resource holdings and transfers atomically; an unaffordable offer cannot be accepted."
    },
    {
      "id": "CATAN-CLAR-DISCARD-ESCROW",
      "source_gap": "The sources do not define how a privately submitted seven-discard interacts with a permitted development-card interruption before all players submit.",
      "decision": "A submitted selection becomes private escrow represented by that player's counts in the pending discard frame. It remains included in the privileged aggregate resource counts until simultaneous settlement, but is unavailable to Monopoly, robbery, trade, building, maritime trade, or any other interrupt. After every required player submits, all escrowed cards move to the bank together. An interrupt must resume the exact pending selections without changing or invalidating them."
    },
    {
      "id": "CATAN-CLAR-KNIGHT-ROBBERY",
      "source_gap": "The primary says the mover robs a player while the Almanac says the mover may rob, leaving digital decline behavior unclear.",
      "decision": "After a Knight moves the robber, if at least one opponent has a settlement or city adjacent to the destination, the mover must choose one adjacent opponent; there is no separate decline action. An adjacent opponent with an empty resource hand remains selectable and then transfers nothing. With no adjacent opponent, movement completes without a robbery choice."
    },
    {
      "id": "CATAN-CLAR-MARITIME-SAME",
```

```
      "source_gap": "The 4:1 wording says a resource of choice, while harbor examples expressly describe receiving another resource.",
      "decision": "Every maritime exchange, including 4:1, 3:1 and 2:1, must receive a resource type different from the type given. Same-resource maritime actions are illegal."
    }
  ]
}
```

D.6 Exploding Kittens Source-Gap Clarification

```
{
  "version": 1,
  "game": "exploding_kittens",
  "role": "experimenter_clarification",
  "authorship": "user-approved BoardBench clarification",
  "approved_at": "2026-07-27",
  "base_source": {
    "source_id": "EXPL-NSFW-DE-2018-RULES",
    "sha256": "f15c85be6345ff0101d01059509bc07e4989896f4f1927ace4248bba4ce1e853"
  },
  "guardrail": "This artifact resolves only the observed empty-target source gap. It does not restate or weaken the publisher-clear requirement that an eliminated player's complete remaining hand and Exploding Kitten enter the discard pile.",
  "clarifications": [
    {
      "id": "EXPL-D-EMPTY-TARGET",
      "claim_ids": ["EXPL-X-EMPTY-TARGET"],
      "decision": "Favor and Pair may target only a living opponent who currently holds at least one card. A living opponent with an empty hand is not a legal target.",
      "affected_mechanics": ["legal actions", "Favor targeting", "Pair targeting", "pending choice phases"]
    }
  ]
}
```

D.7 Wizard Source-Gap Clarification

```
{
  "version": 1,
  "game": "wizard",
  "role": "experimenter_clarification",
  "authorship": "user-approved BoardBench clarification",
  "approved_at": "2026-07-26",
  "base_source": {
    "source_id": "WIZARD-RULES",
    "sha256": "167254a64b0866266746833c0b98836db51c08171b5b96e25f7461d5bf3dee79"
  },
  "guardrail": "These decisions resolve only source gaps. They do not weaken the publisher-clear rule that after a Wizard opens a trick every later card is legal and no led suit arises.",
  "clarifications": [
    {
      "id": "WIZ-DEC-INV",
      "claim_ids": ["WIZ-G-EXACT-INVENTORY"],
      "decision": "Use exactly one ordinary card of every rank 1 through 13 in each of the four named suits, plus four Wizards and four Jesters. Magier and Zauberer denote the same Wizard type.",
      "affected_mechanics": ["component inventory", "dealing", "state reconstruction"]
    },
    {
      "id": "WIZ-DEC-FLOW",
      "claim_ids": ["WIZ-G-FIRST-DEALER-RESET"],
      "decision": "Player 0 is the first dealer and scorekeeper. At each non-final round boundary collect and reshuffle all 60 cards, then rotate the dealer one player clockwise and deal the next round.",
      "affected_mechanics": ["initial dealer", "round reset", "dealer rotation"]
    },
    {
      "id": "WIZ-DEC-TRUMP",
      "claim_ids": ["WIZ-G-WIZARD-CHOICE-MANDATORY"],
```

```
      "decision": "When the revealed card is a Wizard, the dealer must choose exactly one of the four suits as trump after receiving and viewing their hand. Declining to choose is not legal.",
      "affected_mechanics": ["trump selection", "legal actions", "phase transition"]
    },
    {
      "id": "WIZ-DEC-BID",
      "claim_ids": ["WIZ-G-BID-DOMAIN", "WIZ-G-BID-SUM"],
      "decision": "Predictions are integers from zero through the current hand size, made clockwise beginning with the dealer's left neighbor. The base game imposes no restriction on the sum of predictions.",
      "affected_mechanics": ["prediction order", "prediction actions", "base-game scope"]
    },
    {
      "id": "WIZ-DEC-JESTER",
      "claim_ids": ["WIZ-G-JESTER-CHAIN", "WIZ-G-JESTER-WIZARD"],
      "decision": "After one or more leading Jesters, the first subsequently played ordinary card establishes the led suit. If a Wizard is played before any ordinary card, the trick remains colorless, every later card remains legal, and the first Wizard wins.",
      "affected_mechanics": ["led suit", "legal plays", "trick winner"]
    },
    {
      "id": "WIZ-DEC-TIE",
      "claim_ids": ["WIZ-G-TIE"],
      "decision": "Players tied for the highest final score are joint winners.",
      "affected_mechanics": ["terminal returns", "winner set"]
    },
    {
      "id": "WIZ-DEC-PRIVACY",
      "claim_ids": ["WIZ-G-PRIVACY"],
      "decision": "Each player observes their own cards and only opponents' hand sizes. Predictions, revealed trump, played cards, turn information, and scores are public.",
      "affected_mechanics": ["observations", "private hands", "public state"]
    }
  ]
}
```

Bibliography

AMIGO Spiel + Freizeit GmbH. *Wizard*. AMIGO Spiel + Freizeit GmbH, Dietzenbach, 1996. German rulebook, Version 1.0. Printed copyright MCMXCVI. https://www.brettspiele-report.de/images/wizard/Spielanleitung_Wizard.pdf.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Exploding Kittens. *Exploding Kittens NSFW Edition*, 2018. German rulebook, 2–5 players. https://www.brettspielversand.de/mediafiles/spieleanleitungen/asmodee/16-0404_explodingkittensnsfw_anleitung.pdf.

Wolfgang Kramer. *6 nimmt!* AMIGO Spiel + Freizeit GmbH, Dietzenbach, 2011. German rulebook, Version 2.3. Printed copyright MCMXCIV, MMXI. https://www.ggnf.de/spielanleitungen/6-nimmt/6-nimmt.pdf.

Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, Daniel Hennes, Dustin Morrill, Paul Muller, Timo Ewalds, Ryan Faulkner, János Kramár, Bart De Vylder, Brennan Saeta, James Bradbury, David Ding, Sebastian Borgeaud, Matthew Lai, Julian Schrittwieser, Thomas Anthony, Edward Hughes, Ivo Danihelka, and Jonah Ryan-Davis. Openspiel: A framework for reinforcement learning in games, 2020. URL https://arxiv.org/abs/1908.09453.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. doi: 10.1038/nature14539.

Laurent Lévi and Michel Lalet. *Abalone*. Schmidt Spiele GmbH, Berlin, n.d. German rulebook. The supplied PDF has no printed publication year. https://www.spiele4us.de/wp-content/uploads/2022/10/abalone-mini-gebraucht_g003801400.pdf.

Uwe Rosenberg. *Bohnanza*. AMIGO Spiel + Freizeit GmbH, Dietzenbach, 2016. German basegame rulebook, Version 5.4. Printed copyright 1997, 2016. https://blog.amigo-spiele.de/content/ap/rule/01661-DE-AmigoRule.pdf.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. Paperbench: Evaluating ai's ability to replicate ai research, 2025. URL https://arxiv.org/abs/2504.01848.

Klaus Teuber. *CATAN: Spielanleitung*. Franckh-Kosmos Verlags-GmbH & Co. KG, Stuttgart, 2022a. German base-game rulebook, Art.-Nr. 68 26 82. https://www.catan.de/sites/default/files/2022-08/CATAN-Das_Spiel-Manual2022.pdf.

Klaus Teuber. *CATAN-Almanach*. Franckh-Kosmos Verlags-GmbH & Co. KG, Stuttgart, 2022b. Official companion to the 2022 German base game, Art.-Nr. 68 26 82. https://www.catan.de/sites/default/files/2022-08/CATAN-Das_Spiel-Almanach2022.pdf.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering, 2024. URL https://arxiv.org/abs/2405.15793.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023. URL https://arxiv.org/abs/2306.05685.