Ludii General Game System for Modeling, Analyzing, and Designing Board Games

Cameron Browne, Éric Piette,
Matthew Stephenson and Dennis J. N. J. Soemers
Maastricht University, Maastricht, Netherlands

Definition

The Ludii computer program is a complete general game system for digitally modeling, analyzing, and designing a wide range of games. These include traditional tabletop games such as board games and dice games, in addition to card games, graph games, mathematical games, puzzles, simulations, and simple video games. Ludii supports stochastic (chance) elements, hidden information, adversarial and cooperative modes of play, and any number of players from 1 to 16.

The system differs from existing *general game playing* (GGP) programs in a number of ways. Its underlying ludemic model allows a wider range of games to be described more easily and succinctly than other approaches, and it is intended as a tool for game *design* as much as game *playing*. Ludii belongs the "hybrid" class of GGP approaches that allows extensible higher-level game descriptions (Kowalksi et al. 2020).

The Ludii distribution comes with over 1000 predefined games and a number of default *artificial intelligence* (AI) agents for playing and analyzing these, in addition to new games authored by users. An open Ludii AI API is provided to facilitate the system's use as a platform for general game-based AI research. Ludii's Java code base is freely available under a Creative Commons (CC BY-NC-ND 4.0) license.

Ludii was developed as part of the European Research Council (ERC) funded Digital Ludeme Project with one of its primary purposes being for the reconstruction of historical games from partial rulesets based upon the available evidence (Browne et al. 2019b). However, this is only one application, and Ludii provides a range of features intended to help the modern game designer prototype, fine tune, and discover new designs.

The Ludii System

The Ludii system is based on the notion of the *ludeme*, which can be described as a game related concept or element of play that is relevant to the equipment and/or rules of a game (Browne 2021). Ludemes constitute the fundamental building blocks of which games are composed.

Games are defined for Ludii as structured ludeme trees in the form of LISP-like symbolic expressions, according to a custom grammar that constitutes the *Ludii Game Description Language* (L-GDL). For example, the game Tic-Tac-Toe can be defined as follows:

© Springer Nature Switzerland AG 2023
N. Lee (ed.), *Encyclopedia of Computer Graphics and Games*,
https://doi.org/10.1007/978-3-319-08234-9_486-1

```
(game "Tic-Tac-Toe"
    (players 2)
    (equipment {
        (board (square 3))
        (piece "Disc" Each)
    })
    (rules
        (play (move Add (to (sites
        Empty))))
        (end
            (if
                (if Line 3)
                (result Mover Win)
            )
        )
    )
)
```

This *ludemic model* for describing games allows a wide range of games to be described simply and succinctly. The fact that ludemes encapsulate key game concepts makes such descriptions highly conducive to automated manipulations such as the evolution of new games from existing rulesets (Browne 2009).

Architecture

Figure 1 shows an overview of the main components of the Ludii system. Additional information on the relevant modules are provided in the following sections.

The game description (written using the L-GDL) is initially passed into the Grammar module, which first expands and parses the game description to check that it is syntactically valid. Once this initial check is done, the game description is then compiled, transforming it into its internal logic format within the Core module. This internal logic contains all the relevant information about the game, including the equipment, rules, and supplementary metadata. The metadata provides additional information about the game which is not related to how it is played. This includes textual information about the game – such as its rules, author, and historical details – but also graphical information about how the game should be visualized.

The rules section of the Core module is then used by the Manager module, which is responsible for coordinating the moves of the game. This module takes input from either a human user, via the Player module, or an Agent, via the AI module.

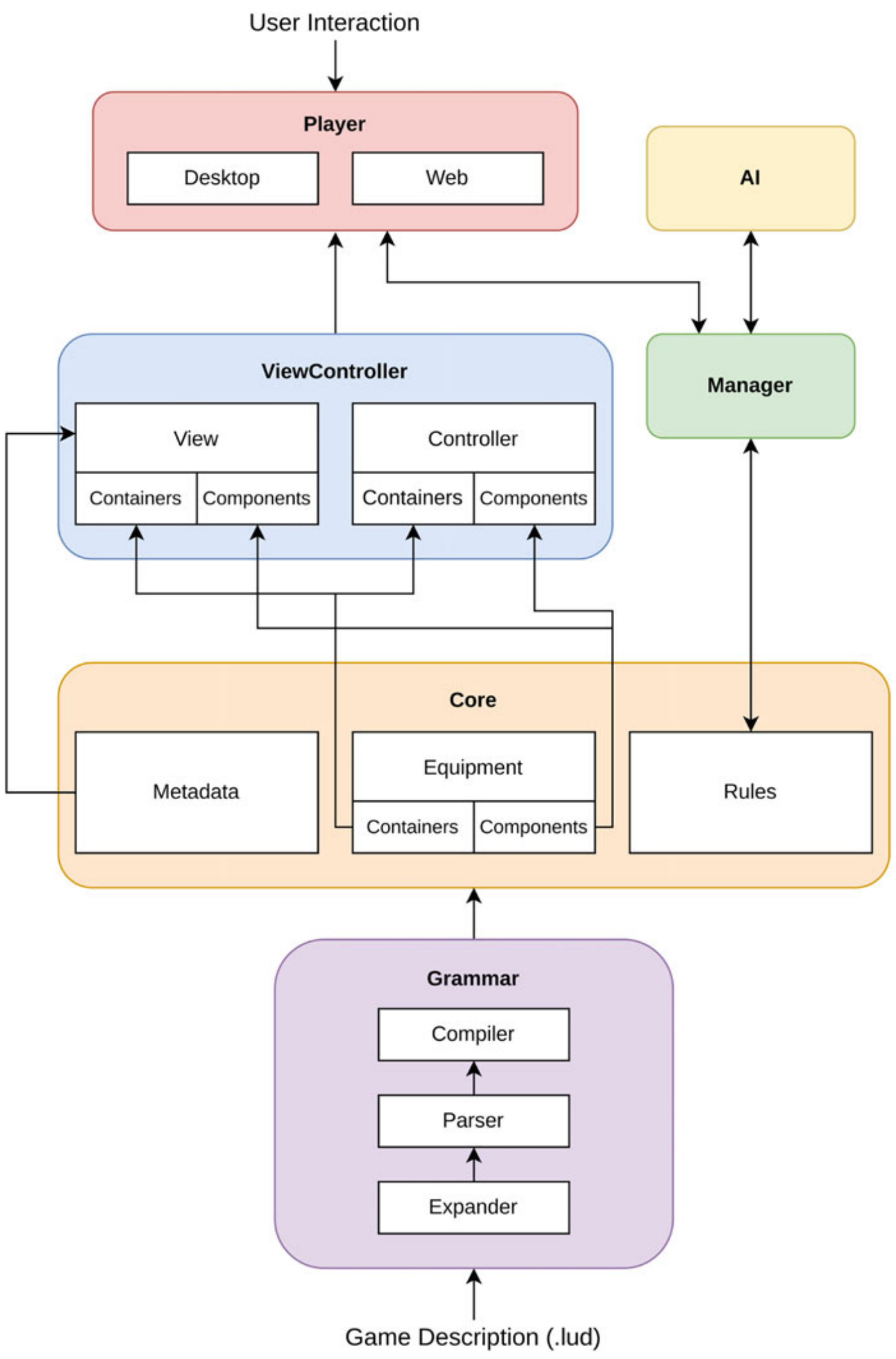
The equipment section of the Core module is used by the ViewController module, which is responsible for constructing the relevant graphics for the containers and components used in the game (view) as well as how interface inputs should be converted into game moves (controller). Any graphical options specified in metadata are also used by the view section of the ViewController module, to supplement or override what is defined in the equipment. For example, a metadata line could be added to the game that changes the size of a piece or the color of the board.

The Player module displays the visuals provided by the ViewController module to the user. When the user interacts with these visuals, the controller converts these inputs into a logical move and sends this to the Manager module. The Player module provides different visuals and interaction handling depending on if the user is operating on their own local version of Ludii (desktop) or the remote website version (web). Once the Manager module verifies that this move is valid, it is passed to the Core module, where the state of the game is updated.

Grammar

The custom Ludii Grammar is an Extended Backus-Naur Form (EBNF) style grammar consisting of a set of production rules used to generate game descriptions. The grammar is generated automatically from the Ludii code base using a *class grammar approach* in which all keywords, rules, and instantiations are derived directly from their corresponding Java classes (Browne 2016).

This approach provides a 1:1 correspondence between the L-GDL and the underlying Java code at all times, effectively making the Ludii Grammar a snapshot of the current class hierarchy and making the system easily extensible. New functionality can be added by simply implementing the relevant Java classes, which will then be automatically incorporated into the grammar the next time that Ludii is launched.



Ludii General Game System for Modeling, Analyzing, and Designing Board Games, Fig. 1 Architecture overview of the Ludii general game system

Each game description is contained in a plain text file with *.lud extension. When a game description is loaded into the Ludii system, the following steps are performed to compile the description into an executable Game object, as shown in Fig. 1:

1. *Expand:* The game description is expanded into a plain text string to resolve certain meta-language features and decorations.
2. *Parse:* The resulting string is parsed for correctness according to the current Ludii Grammar.
3. *Compile:* The Java classes corresponding to the game description keywords are recursively instantiated with the specified parameters to produce an executable Game object in Java bytecode.

Internal Model

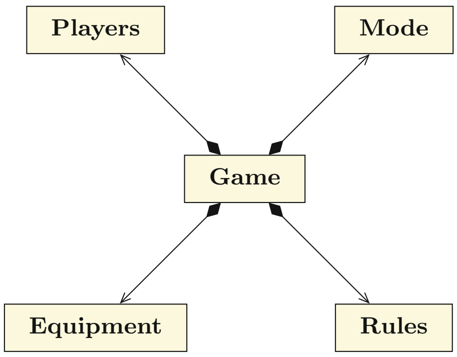
The game representation and the transitions between states are described in the following sections.

Game Representation

A game is defined as a 4-tuple of ludemes:

- **Players**: provides the information about the players (number of players, direction of each player, ...).
- **Mode**: corresponds to the game control and describes if the game is played alternatingly or simultaneously.
- **Equipment**: describes all the information about the containers used in the game (boards and hands) as well as the components (i.e., the game pieces).
- **Rules**: describes the initial state, how different components interact with each other, what moves can be made, and the conditions to reach a terminal state.

Figure 2 shows the components of a game in Ludii.



Ludii General Game System for Modeling, Analyzing, and Designing Board Games, Fig. 2 The game components

State Representation

A Ludii game state s encodes which player is to move in s as well as which player was moving in the previous state and which player is going to move in the next state.

Each container of a game is modeled as a graph defined by a set of cells C, a set of vertices V, and a set of edges E. Each playable *location* $l = \langle c_i, t_i, s_i, l_i \rangle$ is specified by its container $c = \langle C, V, E \rangle$, a site type $t_i \geq$.

$\in$ {Cell, Vertex, Edge}, a site index $s_i \geq 0$, and a *level* $l_i \geq 0$. Every location specifies a type of site in a specific container at a specific level. Each container of the game has its state representation called container state.

A container state *cs* is implemented as a collection of data vectors for each playable site. The different data vectors are:

- what[l]: The index of the component at l, or 0 if there is no component.
- who[l]: The index of the owner of the component at l, or 0 if there is no component.
- count[l]: The number components (of a single type) at l.
- state[l]: The local state of the component at l, or 0 if there is no component.
- value[l]: The value of the component at l, or 0 if there is no component.
- rotation[l]: The rotation index of the component at l, or 0 if there is no component.

Different representations are implemented to minimize the memory footprint and to optimize the time needed to access necessary data for reasoning on any game. These representations are:

- Flat state: For games played on one single site type without stacking.

- Graph state: For games played on multiple site types without stacking.
- Stack state: For stacking games played on one single site type.
- Graph Stack state: For stacking games played on multiple site types.
- Deduction Puzzle state: For puzzles corresponding to a Constraint Satisfaction Problem (Piette et al. 2019).

Figure 3 shows the relations between the different state representations. Thanks to these different state representations, Ludii is able to model a very large set of various games. Figure 4 shows an overview of the Ludii games library (1019 games in version 1.3.0).

Trial and State Transitions

A Ludii successor function is given by $\mathcal{T} : (\mathcal{S} \backslash S_{\text{ter}}) \times \mathcal{A} \mapsto S$, where $\mathcal{S}$ 6is the set of all the Ludii game states, S_{ter} the set of all the terminal states, and A the set of all possible lists of actions, where a single list of actions is a move that a player can select. Given a current state $s \in \mathcal{S} \backslash S_{ter}$, and a list of atomic actions $A = [a_i] \in \mathcal{A}$, $\mathcal{T}$ computes a successor state $s' \in \mathcal{S}$.

A trial τ is a sequence of states s_i and action lists A_i : $s_0, A_1, s_1, \ldots, s_{f-1}, A_f, s_f$ such that $f \geq 0$, and for all $i \in \{1, \ldots, f\}$,

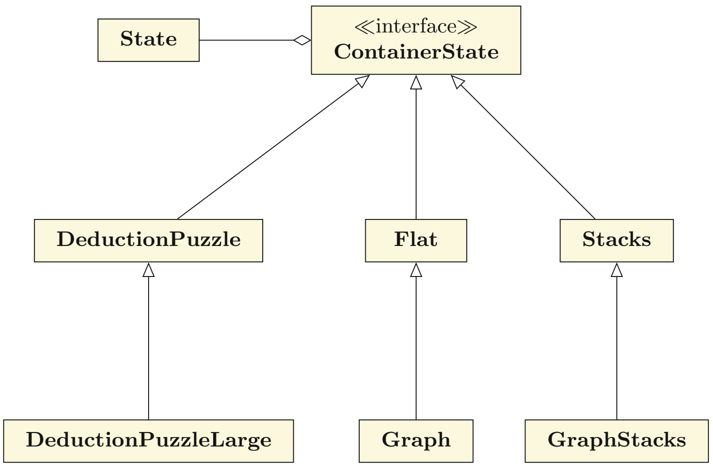
- The played action list A_i is legal for the mover (s_{i-1}).
- States are updated: $s_i = \mathcal{T}(s_{i-1}, A_i)$.
- Only s_f may be terminal: $\{s_0, \ldots, s_{f-1}\} \cap \mathcal{S}_{ter} = \emptyset$.

When a new state s_{i+1} is reached after applying A_i selected from the list of legal moves for a state s_i, Ludii computes the new list of legal moves of s_{i+1} and stores them in the trial for any caller to access them quickly without needing to compute them again.

A trial is over when all players are inactive and associated with a rank. The outcome of the game corresponds to the ranking of the players.

In short, a trial τ provides a complete record of a game played from start to end, including all the moves made stored in a list. Any reasoning on any game can be parallelized using separate trials per thread. All the data members of a game are constant and can therefore be shared between threads. A thread will be able to use a trial τ to compute any playout from any state.

In Ludii, a single object called *Context* is used to store references to the game, the state representation s, and the trial τ. For any operation such as computing the graph of a container, computing the initial state s_0, or computing the legal moves for a state s, Ludii evaluates a tree of ludemes by calling a method eval(context) to evaluate it according to the current state. The *Context* also contains the random number generator used for any stochastic operations in the corresponding trial and the value of the model – alternating or



Ludii General Game System for Modeling, Analyzing, and Designing Board Games, Fig. 3 The different state representations

Ludii General Game System for Modeling, Analyzing, and Designing Board Games, Fig. 4 Some example visualizations of games from the Ludii Library

simultaneous – used to apply moves or compute the legal moves in a specific game state. Alternating-move models expect only a single player to select a move at a time, whereas simultaneous-move models expect *all* active players to select a move at every time step, and simulations simply apply all legal moves automatically. More details about the internal model can be found in Piette et al. (2021).

Board Representation

In Ludii, the board shared by all players is represented internally as a finite graph defined by a triple of sets $\mathcal{G} = \langle V, E, C \rangle$ in which V is a set of *vertices*, E a set of *edges*, and C a set of *cells*.

For example, Fig. 5a shows a game with pieces played on the vertices, edges, and cells of the board graph. Figure 5b shows a board game played only on the cells but in which pieces may stack.

In any given game, a component (or a stack of components) can be placed on any location corresponding to a graph element and a level.

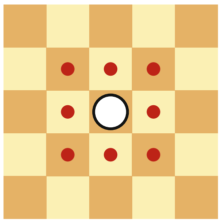
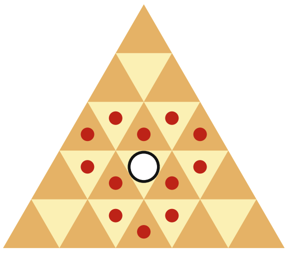
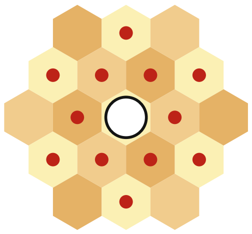
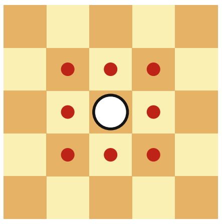
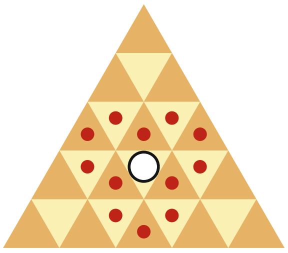
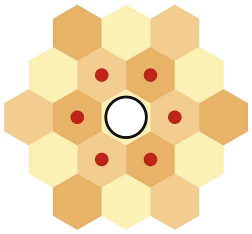
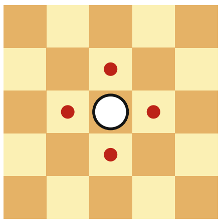
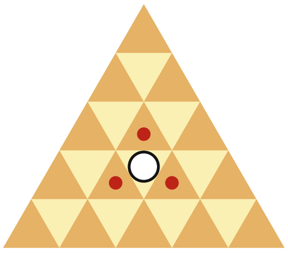
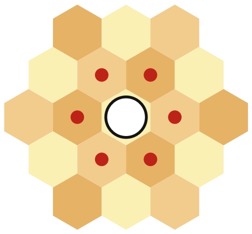
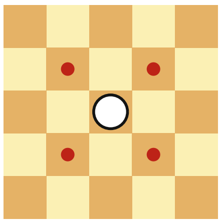
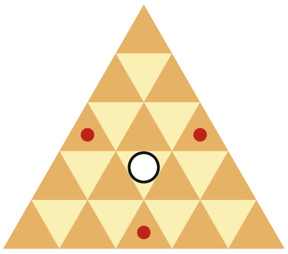
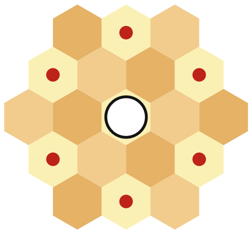
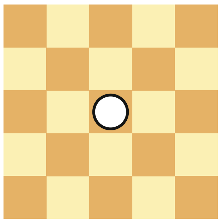
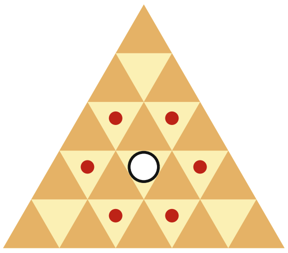
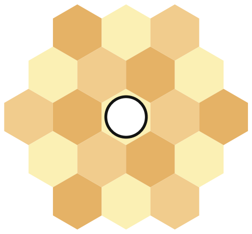
Two different graph elements can have different relations: Adjacent, Orthogonal, Diagonal, Off Diagonal, or All. The complete definition of each of these relations is provided in Browne et al. (2021). These relationships are summarized for the regular tilings in Table 1.

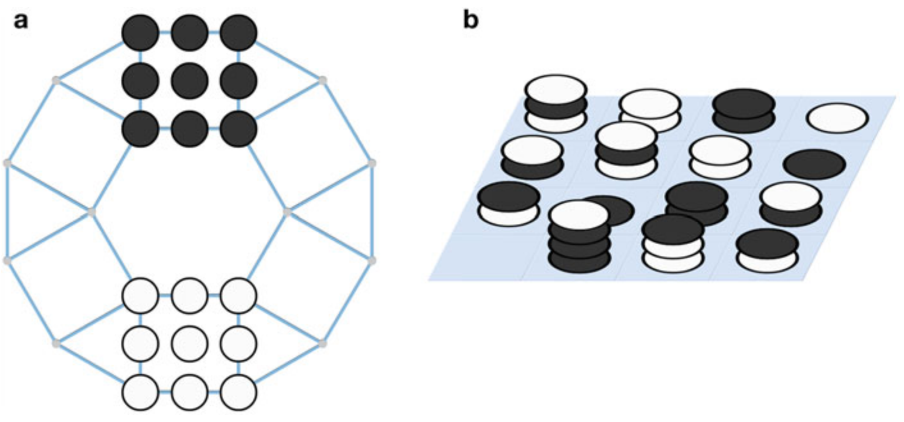
Ludii supports the following direction types:

- *Intercardinal* directions: `N`, `NNE`, `NE`, `ENE`, `E`, `ESE`, `SE`, `SSE`, `S`, `SSW`, `SW`, `WSW`, `W`, `WNW`, `NW`, and `NNW`.
- *Rotational* directions: `In`, `Out`, `CW` (clockwise), and `CCW` (counterclockwise).
- *Spatial* directions for 3D games: `D`, `DN`, `DNE`, `DE`, `DSE`, `DS`, `DSW`, `DW`, `DNW` and `U`, `UN`, `UNE`, `UE`, `USE`, `US`, `USW`, `UW`, and `UNW`.
- *Axial* directions subset (for convenience): `N`, `E`, `S`, and `W`.
- *Angled* directions subset (for convenience): `NE`, `SE`, `SW`, and `NW`.

Each graph element has a corresponding set of *absolute directions* and *relative directions* to associated graph elements of the same type. Absolute directions can be any of the above direction types in addition to any relation type (`Adjacent`,

Ludii General Game System for Modeling, Analyzing, and Designing Board Games, Table 1 Relations for the regular tilings

Relation	Square	Triangular	Hexagonal
All			
Adjacent			
Orthogonal			
Diagonal			
Off-Diagonal			



Ludii General Game System for Modeling, Analyzing, and Designing Board Games, Fig. 5 A game played on vertices, edges, and cells (**a**) and a game played only on cells (**b**)

`Orthogonal`, `Diagonal`, `Off Diagonal`, or `All`).

Relative directions from an element are defined by the direction in which a component is facing, the number of rightward steps of the component, and the graph relation to use at each step (Adjacent by default). Relative directions are: `Forward`, `Backward`, `Rightward`, `Leftward`, `FR`, `FRR`, `FRRR`, `FL`, `FLL`, `FLLL`, `BR`, `BRR`, `BRRR`, `BL`, `BLL`, or `BLLL`.

Game Logic

The logic of an L-GDL game (Piette et al. 2021) is computed from its *rules*, ludeme which is defined by the following ludemes which correspond to different rule types:

1. *meta*,
2. *start*,
3. *play*,
4. *end*.

For example, the following L-GDL code describes the Game of the Amazons (Amazons: ludii.games/details.php?keyword = Amazons) with an additional swap rule (a rule that allows the second player to swap colors after the first move to reduce any first move advantage). This game is used as example in the next sections.

```
(game "Amazons"
    (players 2)
    (equipment {
        (board (square 10))
        (piece “Queen” Each
         (move Slide (then (moveAgain)))
        )
        (piece "Dot" Neutral)
    })
    (rules
        (meta (swap))
        (start {
          (place "Queen1"
          {"A4" "D1" "G1" "J4"}
           )
          (place "Queen2"
             {"A7" "D10" "G10" "J7"}
          )
        })
        (play
          (if (is Even (count Moves))
            (forEach Piece)
            (move Shoot (piece "Dot0"))
          )
        )
        (end
          (if
            (no Moves Next)
            (result Mover Win)
           )
        )
    )
)
```

Metarules

In Ludii, a metarule, defined with the ludeme *meta*, is a global rule applied in each state

s reached after applying the move decided by the player. That rule can modify the state s or can add/remove some moves from the list of legal moves. In the Game of the Amazons, the swap rule is defined using the metarule (*swap*) (line 12). Here, after the first player has finished their turn, the second player has one more legal move allowing them to swap with the other player. The metarules are optional.

Starting Rules and Initial State

When a game is creating after being compiled, the state s_{-1} corresponds to all the variables set to their default values and no piece placed in any playable location. The starting rules (lines 13–20) of the Game of the Amazons are used to place the queens on the expected locations to create the initial state. These rules define a list of movements A_0 applied to the state s_{-1} to build s_0. The starting rules are optional.

Playing Rules and Move Generation

The playing rules of a game describe how to generate the legal moves of the mover for any current state s_i. These legal moves are defined in the Play ludeme through its Moves ludemes. The playing rules of the Game of the Amazons are described in lines 21–26.

At each state s_i, the Moves ludeme used to describe the playing rules are evaluated according to the state and return a list of k legal moves $\mathcal{M} : \langle m_1, \ldots, m_i, \ldots, m_k \rangle$ stored in the trial τ. As described in Section, the transition between two successive states s_i and s_{i+1} is possible as a sequence of atomic actions A_i applied to s_i. Such a sequence is modeled as a move $m : \langle a_1, \ldots, a_i, \ldots, a_n \rangle$, where n is the number of actions in A_i..

An atomic action a is the only operator able to modify the state after its creation. Consequently, when a player selects their move m from the list of legal moves available in the trial for the state s_i, this state is updated by applying successively each atomic action in the list of actions composing the move m.

Ending Rules and Terminal States

The ending rules describe when and how play can terminate for one or more of the players. In the Game of the Amazons, the ending rule (lines 27–32) checks whether the next player has no legal moves; if this is the case, the game is over and the current player wins.

In Ludii, any conditions to reach an ending state are described in the ending rules, followed by the description of the outcome of at least one player. In games with two players or fewer, an ending rule describes a terminal state s_{ter}, but for games with more players, the game can continue if play did not yet terminate for at least two of the players.

Functions

All the ludemes defining the rules are functions that are evaluated according to a state s returning a specific type of data. Five types of functions exist in Ludii:

- **Moves** functions return a list of moves. The Moves ludemes starting by (move ...) describes a decision move, all the other Moves ludemes are effect moves. To make the computation of the legal moves efficient, the effect moves which have to be applied before the decision action a_d are distinguished from those that have to be applied after, corresponding to the consequences of the decision and described using the ludeme (then ...). Due to that distinction, only the non-consequence moves are fully evaluated during the computation of the legal moves, and the consequences are evaluated only when a specific move has been selected to be applied by the player. In the context of the slide movement of the Game of Amazons, the effect Moves ludeme (moveAgain) is evaluated when a slide move is decided, setting the next player be the current mover.
- **Arithmetic** functions return one or many numerical values. The arithmetic functions are composed of many different functions according to the type of numerical values returned (array, integer, range, or real). As examples, the ludemes (count Sites “Board”) and (count Players) return the number of sites in the board, and the number of players, respectively.

- **Logic** functions return a Boolean value. The most common logic functions start by (is ...), such as (is Even (count Moves)), which returns true if the number of moves played so far is even.
- **Region** functions return one or many playable sites. The most common region functions start by (sites ...), such as (sites Board), which returns a list of all the sites on the board.
- **Direction** functions return one or many absolute directions. For example, (directions {Rightward, Leftward}) is returning the absolute directions corresponding to the right and left of the current direction of a piece.

Ludii Player

The Ludii Player provides the graphical user interface (GUI) aspect of Ludii. This includes both the visuals and controls for playing games, as well as additional software options to improve the user/developer experience (e.g., remote online games, a built-in editor, game analysis tools, advanced graphical settings, etc.). This is something that is either missing or severely lacking in most other general game systems.

An example screenshot of the main Ludii Player GUI is shown in Fig. 1. This example demonstrates an in progress game of Shogi. The left side of the player shows the current state of the game board. The top right area of the player displays details about each player, including who is controlling them and the contents of their hand. The bottom right area provides supplementary information about the game, such as the moves that have been made, ludeme description, agent analysis results, etc. A range of menu options at the top of the Ludii Player also provides many other alternative features.

A few of the user-friendly features offered by the Ludii Player, and their uses for research, are now described. Firstly, being able to visually see and play the games described using the L-GDL makes testing and verifying the correctness of game descriptions much easier. The benefit of this point should not be understated, as there have been several cases of games being described for alternative systems which were later found to be incorrect. Secondly, the heuristics and strategies of agents can be easily viewed to see their current performance and if there are any obvious weaknesses in their behavior. Humans can also play directly against agents to help determine if they are at a human-level playing strength. Lastly, providing a user-friendly interface is more inviting to the general public and encourages other game design enthusiasts to create their own games, leading to a larger range of games for research purposes. Ludii currently includes over 1000 games which were created by members of the general public, with new games being added frequently.

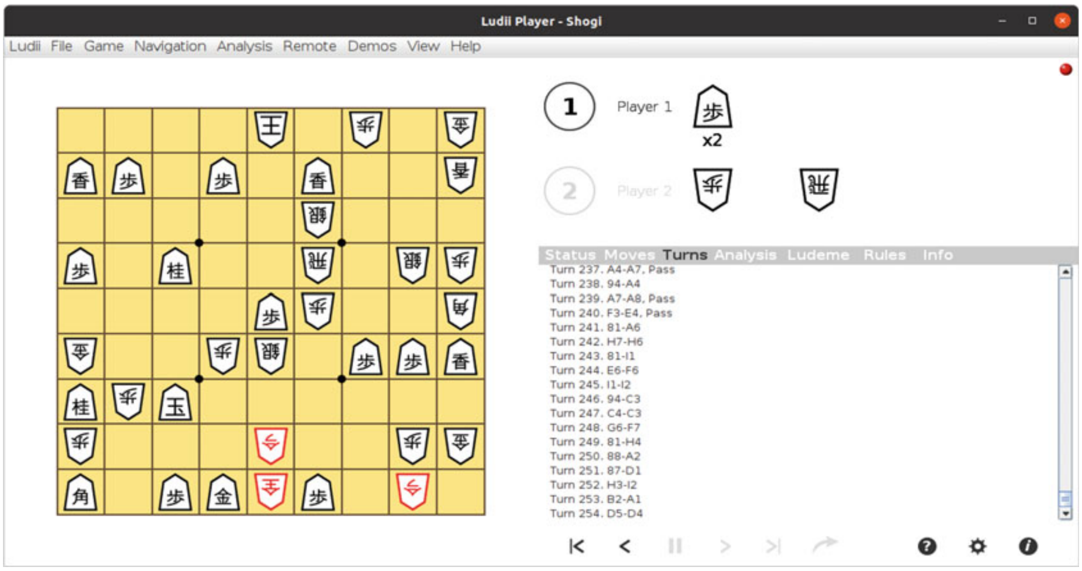
User Interface

The Ludii Player provides the graphical user interface (GUI) aspect of Ludii, including both the visuals and controls for playing games. There are currently two different version of the Ludii Player: the Desktop Player, which is used when running Ludii locally on any standard PC, and the Web Player, which is used when interacting via the Ludii Portal Website.

Ludii Desktop Player

An example screenshot of the Ludii Desktop Player GUI is shown in Fig. 6. The left side of the player shows the current state of the game board. The top right area of the player displays details about each player, including who is controlling them and the contents of their hand. The bottom right area provides supplementary information about the game, such as the moves that have been made, ludeme description, agent analysis results, etc. A range of menu options at the top of the Ludii Desktop Player also provides many other additional features, including but not limited to:

- The ability to play remote games and tournaments with other Ludii users online.
- A built-in editor for creating, modifying, and saving Ludii game descriptions.
- Game analysis tools for evaluating and comparing games across a variety of metrics.
- Multiple graphical settings, such as animations, move highlighting, cell coordinates, etc.


Ludii Player - Shogi
Ludii File Game Navigation Analysis Remote Demos View Help
1 Player 1
x2
2 Player 2
Status Moves Turns Analysis Ludeme Rules Info

```
Turn 237. A4-A7, Pass
Turn 238. 94-A4
Turn 239. A7-A8, Pass
Turn 240. F3-E4, Pass
Turn 241. 81-A6
Turn 242. H7-H6
Turn 243. 81-I1
Turn 244. E6-F6
Turn 245. I1-I2
Turn 246. 94-C3
Turn 247. C4-C3
Turn 248. G6-F7
Turn 249. 81-H4
Turn 250. 88-A2
Turn 251. 87-D1
Turn 252. H3-I2
Turn 253. B2-A1
Turn 254. D5-D4
```

Ludii General Game System for Modeling, Analyzing, and Designing Board Games, Fig. 6 The graphical user interface (GUI) of the Ludii Desktop Player for an in progress game of Shogi

- The ability to select different game options and rule sets.

Ludii Web Player

An example screenshot of the Ludii Web Player GUI is shown in Fig. 7. This picture was taken from a mobile smartphone device in portrait mode. Other devices may arrange certain elements such as the player hands differently, but are otherwise functionally identical. As can be seen, the Web Player contains less features that the Desktop Player, essentially only allowing the user to play the game against other AI or human opponents. The benefit of the web version, however, is that it can be played on almost any device with an internet connection and does not require the user to install Java beforehand.

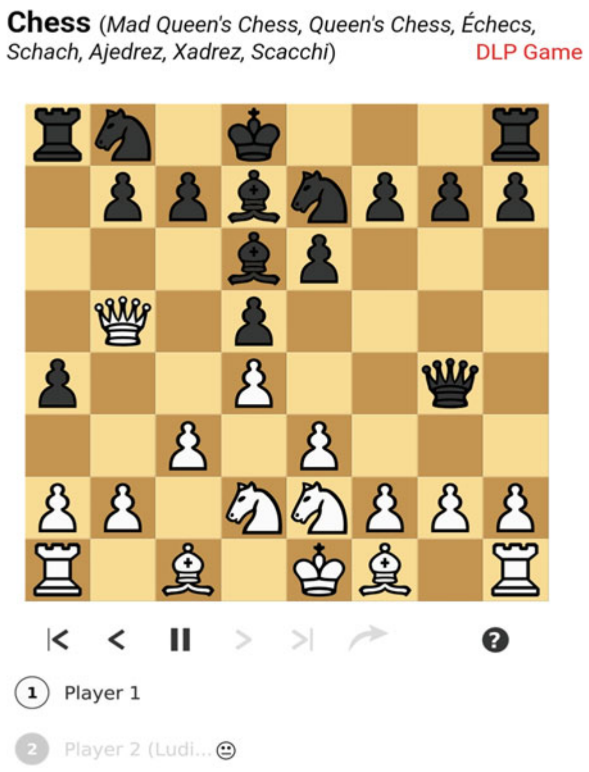
Ludii Portal

Both the Ludii Web Player and the download link for the Ludii Desktop Player can be accessed via the Ludii Portal (ludii.games). This portal also provides additional information about Ludii and the games within it. Some of main web pages that can be accessed from this portal include:

- The Ludii Game Library (ludii.games/library), which displays images and category information for all games within Ludii. Selecting a game from this library will open a Web Player instance of that game.
- The Ludii Downloads Page (ludii.games/download), which contains links for downloading the Ludii Desktop Player, as well as other Ludii documentation.
- The Game Concepts Search Page (ludii.games/searchConcepts), which can be used to search for games with a specific combination of over 700 defined concepts.
- The Ludeme Tree Page (ludii.games/ludemeTree), which displays an interactive hierarchy tree for all the ludemes within the L-GDL.

Artificial Intelligence

Ludii provides an API for game-playing agents using any artificial intelligence (AI) techniques to be developed and used to play any of Ludii's games from within its GUI-based player as well as command-line programs and competitions (Stephenson et al. 2019). The API for agents provides them with a *forward model*; given any (current) game state, this may be used to generate


Chess (Mad Queen's Chess, Queen's Chess, Échecs, Schach, Ajedrez, Xadrez, Scacchi)
DLP Game
Player 1
Player 2 (Ludi...

Ludii General Game System for Modeling, Analyzing, and Designing Board Games, Fig. 7 Ludii Web Player GUI for mobile devices, showing Chess

lists of legal moves, generate successor states resulting from the application of moves, query whether or not a game state is terminal or any rankings have already been determined, and so on. This is similar to the API provided by the General Video Game AI (GVGAI) framework (Perez-Liebana et al. 2019) for its collection of video games. This interface is sufficient for typical tree search algorithms as commonly used for GGP, such as Monte-Carlo Tree Search (MCTS) (Kocsis and Szepesvári 2006; Browne et al. 2012; Coulom 2007). There is also support for tensor representations of states and actions to be generated, the use of which has been demonstrated in a bridge between Ludii and the Polygames (Cazenave et al. 2020) framework of deep learning approaches for games. Various types of constraints can be specified for agents, such as processing time per move, maximum iteration count, and maximum search depth; different constraints may be more or less suitable for different experiments or use cases.

Based on this interface, several standard algorithms have already been implemented and included directly in Ludii, as well as new techniques developed and proposed specifically in the context of DLP and Ludii. In GGP, one of the most commonly used search algorithms is MCTS. Ludii includes implementations of several variants and common extensions, such as UCT (Browne et al. 2012), GRAVE (Cazenave 2015), MAST (Finnsson and Björnsson 2008), Progressive History (Nijssen and Winands 2011), and NST (Tak et al. 2012). It also includes training techniques and variants of MCTS that are guided by trained features, which are described in other publications (Browne et al. 2019a; Soemers et al. 2020).

Another search technique implemented in Ludii is $\alpha\beta$-search (Knuth and Moore 1975), with Max^N (Luckhardt and Irani 1986), Paranoid search (Sturtevant and Korf 2000), and BRS+ (Esser et al. 2014) extensions for games with more than two players. Unlike MCTS, these techniques require heuristic evaluation functions – generally based on domain knowledge – to compare the “desirability” of various states. A variety of heuristics, most of which were found to be fairly generally useful across multiple games (Browne 2009), are included in Ludii for this purpose. Typical examples include a *material* heuristic to count weighted sums of types of pieces owned by players or terms that compute proximity to board centers, corners, sides, and so on.

Ludii Database

All relevant information about each official Ludii game (i.e., those which are included within the code Ludii software and repository) is stored inside the Ludii Game Database (LGD). This data can be decomposed into two main types, *game-related* and *evidence-related*. The evidence-related data is primarily stored only for games that are relevant to the goals of the DLP. As such, a large portion of games in the LGD does not have any evidence-related information. As this information is unlikely to be useful outside of this archaeological context, this section will focus primarily on the game-related data. This game-related data can further be split into three subsections: *games*, *rulesets*, and *ludemes*. Each game can be thought of as being composed of one or more rulesets, with each rulesets being made up of multiple ludemes.

While the distinction between a game and a ruleset is not exact, two sets of rules/equipment can be considered different games if they come from different places or existed in different time periods. If they cannot be separated, however, then they are considered different rulesets of the same base game. This can lead to two rulesets of the same game with very different rules/equipment, such as two different sets of rules that have been suggested for a historical game with largely unknown rules. It can also lead to two distinct games with very similar rules/equipment, to consider the possibility that these games were created independently in different places and times.

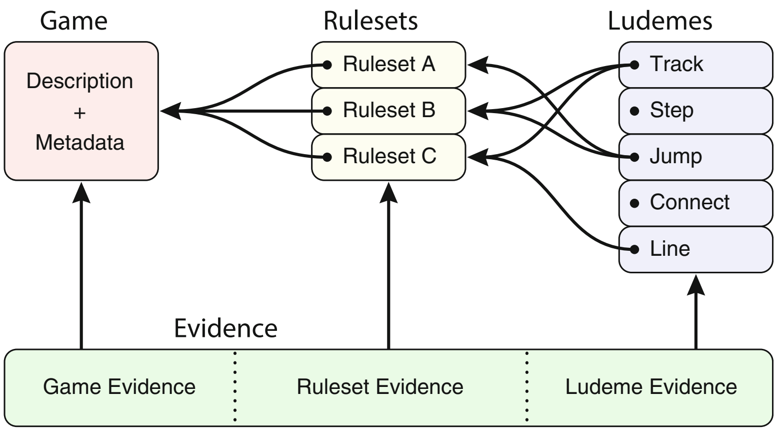
A rough outline of the LGD structure is shown in Fig. 8.

Games

Each game entry in the LGD describes a specific game (i.e., a single .lud game description) in Ludii. Each of these game entries will also have at least one ruleset associated with it, although it can have more. Auxiliary metadata information about each game is also stored, such as plain English descriptions of the game and its rules, any aliases, publication details, and so on.

Rulesets

A ruleset is a defined set of ludemes which describe the specific rules and equipment that is used to play a certain game. These rulesets could be speculative in nature or can simply be a known variant of an established game (e.g., the different scoring systems for the game Go). Playing the same game with a different ruleset can often lead to different gameplay experiences. One ruleset



Ludii General Game System for Modeling, Analyzing, and Designing Board Games, Fig. 8 Overview of the main Ludii Game Database groups and their relationships could make a game long and biased, with little room for strategic play while another could provide the complete opposite. As a result of this change, it is these rulesets that are evaluated and analyzed when it comes to gameplay, rather than the game itself.

Ludemes

A ludeme is single elemental building block of a game. Multiple ludemes can be combined together to create a description of a specific piece of equipment or rule that a game uses. Each ludeme is stored in the LGD and is associated with the rulesets which use it. If a game has any ruleset that uses a particular ludeme, then by extension, that game will also be considered as using that ludeme.

Cross-References

► Monte-Carlo Tree Search

References

Browne, C.B.: Automatic generation and evaluation of recombination games. Phd thesis, Faculty of Information Technology, Queensland University of Technology, Queensland, Australia (2009)

Browne, C.: A class grammar for general games. In: Advances in Computer Games, vol. 10068 of Lecture Notes in Computer Science, pp. 167–182, Leiden (2016)

Browne, C.: Everything's a ludeme: well, almost everything. In: Proceedings of the XIIIrd Board Game Studies Colloquium (BGS 2021), Paris (2021)

Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of Monte Carlo tree search methods. IEEE Trans. Comput. Intell. AI Games. **4**(1), 1–49 (2012)

Browne, C., Soemers, D.J.N.J., Piette, E.: Strategic features for general games. In: Proceedings of the 2nd Workshop on Knowledge Extraction from Games (KEG), pp. 70–75 (2019a)

Browne, C., Soemers, D.J.N.J., Piette, È., Stephenson, M., Conrad, M., Crist, W., Depaulis, T., Duggan, E., Horn, F., Kelk, S., Lucas, S.M., Neto, J.P., Parlett, D., Saffidine, A., Schädler, U., Silva, J.N., de Voogt, A., Winands, M.H.M.: Foundations of digital archæoludology. Technical report, Schloss Dagstuhl Research Meeting, Germany (2019b)

Browne, C., Piette, É., Stephenson, M., Soemers, D.J.N.J.: General board geometry. In: Advances in Computer Games (ACG 2021) (2021)

Cazenave, T.: Generalized rapid action value estimation. In: Yang, Q., Woolridge, M. (eds.) Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015), pp. 754–760. AAAI Press, Buenos Aires, Argentina (2015)

Cazenave, T., Chen, Y.-C., Chen, G.W., Chen, S.-Y., Chiu, X.-D., Dehos, J., Elsa, M., Gong, Q., Hu, H., Khalidov, V., Li, C.-L., Lin, H.-I., Lin, Y.-J., Martinet, X., Mella, V., Rapin, J., Roziere, B., Synnaeve, G., Teytaud, F., Teytaud, O., Ye, S.-C., Ye, Y.-J., Yen, S.-J., Zagoruyko, S.: Polygames: improved zero learning. ICGA J. **42**(4), 244–256 (2020)

Coulom, R.: Efficient selectivity and backup operators in Monte-Carlo tree search. In: van den Herik, H.J., Ciancarini, P., Donkers, H.H.L.M. (eds.) Computers and Games, vol. 4630 of LNCS, pp. 72–83. Springer, Turin, Italy (2007)

Esser, M., Gras, M., Winands, M.H.M., Schadd, M.P.D., Lanctot, M.: Improving best-reply search. In: van den Herik, H., Iida, H., Plaat, A. (eds.) Computers and Games. CG 2013, vol. 8427 of Lecture Notes in Computer Science, pp. 125–137. Springer, Cham (2014)

Finnsson, H., Björnsson, Y.: Simulation-based approach to general game playing. In: The Twenty-Third AAAI Conference on Artificial Intelligence, pp. 259–264. AAAI Press, Chicago, Illinois (2008)

Knuth, D.E., Moore, R.W.: An analysis of alpha-beta pruning. Artif. Intell. **6**(4), 293–326 (1975)

Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) Machine Learning: ECML 2006, vol. 4212 of Lecture Notes in Computer Science (LNCS), pp. 282–293. Springer, Berlin, Heidelberg (2006)

Kowalksi, J., Miernik, R., Mika, M., Pawlik, W., Sutowicz, J., Szykula, M., Tkaczyk, A.: Efficient reasoning in regular boardgames. In: Proceedings of the 2020 IEEE Conference on Games, pp. 455–462. IEEE, Osaka, Japan (2020)

Luckhardt, C.A., Irani, K.B.: An algorithmic solution of n-person games. In: Proceedings of the Fifth AAAI National Conference on Artificial Intelligence, pp. 158–162. AAAI Press, Philadelphia, Pennsylvania (1986)

Nijssen, J.A.M., Winands, M.H.M.: Enhancements for multi-player Monte-Carlo tree search. In: van den Herik, H.J., Iida, H., Plaat, A. (eds.) Computers and Games (CG 2010), vol. 6515 of Lecture Notes in Computer Science, pp. 238–249. Springer, Kanazawa, Japan (2011)

Perez-Liebana, D., Liu, J., Khalifa, A., Gaina, R.D., Togelius, J., Lucas, S.M.: General video game AI: a multitrack framework for evaluating agents, games, and content generation algorithms. IEEE Trans. Games. **11**(3), 195–214 (2019)

Piette, C., Piette, É., Stephenson, M., Soemers, D.J.N.J., Browne, C.: Ludii and XCSP: playing and solving logic puzzles. In: 2019 IEEE Conference on Games (CoG), pp. 630–633 (2019)

Piette, É., Browne, C., Soemers, D.J.N.J.: Ludii game logic guide. https://arxiv.org/abs/2101.02120 (2021)

Soemers, D.J.N.J., Piette, É., Stephenson, M., Browne, C.: Manipulating the distributions of experience used for self-play learning in expert iteration. In: Proceedings of the 2020 IEEE Conference on Games, Osaka, Japan, pp. 245–252. IEEE (2020)

Stephenson, M., Piette, É., Soemers, D.J.N.J., Browne, C.: Ludii as a competition platform. In: Proceedings of the 2019 IEEE Conference on Games (COG 2019), pp. 634–641, London (2019)

Sturtevant, N.R., Korf, R.E.: On pruning techniques for multi-player games. In: Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, pp. 201–207. AAAI Press, Austin, Texas (2000)

Tak, M.J.W., Winands, M.H.M., Björnsson, Y.: N-grams and the last-goodreply policy applied in general game playing. IEEE Trans. Comput. Intell. AI Games. **4**(2), 73–83 (2012)