# A Practical Introduction to the Ludii General Game System

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

Department of Data Science and Knowledge Engineering,
Maastricht University,
Bouillonstraat 8-10, 6211 LH, Maastricht, The Netherlands
cameron.browne,matthew.stephenson,eric.piette,dennis.soemers
@maastrichtuniversity.nl

**Abstract.** Ludii is a new general game system, currently under development, which aims to support a wider range of games than existing systems and approaches. It is being developed primarily for the task of game design, but offers a number of other potential benefits for game and AI researchers, professionals and hobbyists. This paper is based on an interactive demonstration of Ludii at thuis year's Advances in Computer Games conference (ACG 2019). It describes the approach behind Ludii, how it works, how it is used, and what it can potentially do.

**Keywords:** General Game System, General Game Playing, Game Description Language, Ludeme, Ludii, Game Design, Artificial Intelligence

## 1 Introduction

Ludii is a *general game system* (GGS) [4] for modelling, playing, evaluating, optimising, reconstructing and generating a range of games in a digital format. It is distinct from existing GGSs in that its primary purpose is as a game design tool, with the focus being on the flexibility and expressiveness of its design language and the ease with which games can be defined.

### 1.1 The Digital Ludeme Project

Ludii is being developed as part of the Digital Ludeme Project (DLP)[1], a five-year research project which aims to model the world's traditional strategy games in a single, playable digital database. This database will be used to find relationships between games and their components, in order to develop a model for the evolution of games throughout recorded human history and to chart their spread across cultures worldwide. This project will establish a new field of research called Digital Archæoludology [2].

The DLP will model the thousand most influential traditional strategy games throughout history, each of which may have multiple interpretations and require

[1] Digital Ludeme Project: `http://ludeme.eu/`

hundreds of variant rule sets to be tested. These will mostly be board games but will also include card games, dice games, tile games, etc., and will involve games with non-deterministic elements of chance or hidden information. The Ludii system was developed for this purpose, as no existing general game approach would support the full range of games required for the execution of the DLP.

The following sections describe the approach behind Ludii, its game grammar and compilation mechanisms, how games are represented and played, how the user interacts with the system, and potential services Ludii might offer.

## 2 Ludemic Approach

Ludii is based on a *ludemic approach* that decomposes games into atomic constituents describing relevant equipment and rules.

### 2.1 Ludemes

*Ludemes* are "game memes" or units of game-related information that represent the building blocks of games; they are the conceptual units that game designers work with when developing their designs. The term was coined in the 1970s by Alain Borvo for his analysis of a novel card game [1].

The following example shows how the game of Tic-Tac-Toe might be described in ludemic form. All information required to play the game – the players, the equipment and the rules – are presented in a simple, structured format:

```
(game "Tic-Tac-Toe"
   (players 2)
   (equipment {
      (board (square 3))
      (piece "Nought" P1)
      (piece "Cross" P2)
   })
   (rules
      (play (to (empty)))
      (end (if (line 3) (result Mover Win)))
   )
)
```

The ludemic approach is a high-level approach to game description that encapsulates the key game-related concepts while hiding the complexity of the underlying implementation, making it well suited to the task of game description and design. This is in contrast with existing approaches, such as the Stanford Game Description Language (GDL) [13], that explicitly state the instructions for updating the game state in the descriptions themselves, yielding verbose and complex descriptions that do not encapsulate relevant concepts and are less amenable to the modifications required for game design.

### 2.2 Ludi

Ludii is based on similar principles to the first author's previous Ludi game system, which was used to evolve combinatorial board games in ludemic form [5]. However, Ludii has been completely redesigned to address shortcomings in its previous incarnation, in order to provide the generality, extensibility and efficiency required for the successful execution of the DLP. These improvements are due mainly to the class grammar approach for automated game grammar generation, and Monte Carlo-based move planning with a forward model only, yielding speed-ups in the order of 100 times faster for most games.

## 3 Class Grammar

The Ludi *class grammar* is a set of production rules derived directly from the Java code implementation of the ludeme classes, in which sequences of symbols on the RHS are assigned to a nonterminal symbol on the LHS very much like an Extended Backus-Naur Form (EBNF) grammar [6]. The basic syntax is as follows:

```
<class> ::= { (class [{<arg>}]) | <subClass> | terminal }
```

where:

| | |
|---|---|
| `<class>` | denotes a LHS symbol that maps to a class in the code library. |
| `(class [{<arg>}])` | denotes a `class` constructor and its arguments. |
| `Terminal` | denotes a terminal symbol (fundamental data type or `enum`). |
| {...} | denotes a collection of one or more items. |
| `\|` | denotes a choice between options in the RHS sequence. |

The grammar is intrinsically bound to the underlying code library, but is *context-free* in that it is self-contained and can be used without knowledge of the underlying code. The mechanism for generating the grammar is similar to that of parsing C++ constructors described by Hall [11] to produce a form of *domain specific language* (DSL) [10]. The Ludii class grammar is effectively a snapshot of the class hierarchy of the program's current ludeme code base in Java. Ludii is implemented in Java for its cross-platform support, performance, flexible compilation and good Reflection library support.

### 3.1 Annotations

Custom annotations are used to decorate arguments in ludeme class constructors to help shape the resulting grammar. For example, the `@Opt` annotation is used to denote optional arguments for a ludeme, `@Named` is used to denote arguments that must be named in the grammar, and `@Or` denotes consecutive runs of arguments of which exactly one must be specified in the game description. For example, a `Line` class constructor with the following signature:

```
public Line(
            @Name final IntFunction length,
        @Opt       final Dirn   dirn,
   @Or @Opt @Name final IntFunction what,
   @Or @Opt       final Role    who
)
```

would generate the following rule with named and optional arguments:

```
<line> ::= (line length:<int> [<dirn>] [(what:<int> | <role>)])
```

### 3.2 Game Descriptions

Games are described as *symbolic expressions* or *s-expressions* expressed in the Ludii class grammar. The following example shows the game of Havannah, in which players win by connecting two corners, three board sides (not including corners) or form a ring with their pieces:

```
(game "Havannah"
  (players 2)
  (equipment {(board (hexagon 8)) (piece "Ball" Each)})
  (rules
    (play (to (empty)))
    (end
      (if (or {
        (connect 2 Corners)(connect 3 SidesNoCorners)(ring)
      })
      (result Mover Win)
    )
  )
)
```

### 3.3 Game Compilation

Game descriptions are processed using a *recursive descent parser* [7] in which LHS class names are matched to the actual classes they refer to. The (terminal or non-terminal) arguments to each (non-terminal) class are compiled, then the appropriate class constructor is found, compiled and passed up the compilation hierarchy. The object returned at the root of this compilation process is an executable `Game` object ready to run.

### 3.4 Advantages and Disadvantages

Advantages of the class grammar approach include its easy extensibility, as any required functionality can be simply implemented, added to the code base, and

automatically subsumed into the grammar. The system will theoretically support any functionality that can be implemented in Java, taking a step towards the ideal of the programming language becoming the game description language [15].

A drawback is that users adding ludemes to the code base must follow strict formatting guidelines for the ludeme constructors if they are to produce a well-behaved grammar. However, these are well documented for those who need them.

## 4 Game Representation

A game in Ludii is given by a 4-tuple = $\langle Players, Mode, Equipment, Rules \rangle$. *Players* is a finite set of $k$ players described by the numbers of players. *Mode* is the type of the game between: Alternating (by default if not specified), Simultaneous and Real Time. *Equipment* describes the *containers* and the *components* of the game. The containers are mainly a description of the main board by its shape and its tiling and if necessary the hands of the players. Each component is described by the ludeme *piece* specifying its name, its owner and if necessary how this component can be moved in the board. Finally, *Rules* defines the operations of the game which is split in three distinct parts: *start*, *play* and *end*.

For each container, the system builds a graph representation of the board according to its tiling and precomputes any useful data structure (neighbours of each vertex, corners of the board, etc.) in order to efficiently compute the legal moves from each game state.

### 4.1 Game States

When a game is compiled different flags corresponding to game types are automatically generated in function of the ludemic description. According to them, a game state in Ludii is built. A set of `ContainerState` objects associated with each container defines a game state. A Container state is defined using a custom `BitSet` class (called `ChunkSet`) that compresses the required state information into a minimal memory footprint. The `ChunkSet` encodes multiple data: `What(locn)` the index of the component located at `locn`, `Who(locn)` the owner of this piece, `count(locn)` the number of this component, an internal state of a component (direction, side, etc.) by `state(locn)` and if the information hidden to a player are given by `hidden(player, locn)`.

### 4.2 Moves and Actions

Legal moves are described by a `Moves` object which contains a list of component `Move` objects generated by the "play" rules of the game for the given state. Each move equates to a *complex instruction set* (CISC) command that decomposes into a set of atomic *reduced instruction set* (RISC) `Action` objects, each of which typically modifies a `ChunkSet` in the state.

For example, the move `To(1, 4)` sets the piece with index `1` at cell location `4` of the default container (i.e. the board), by applying the sequence of atomic actions: { `SetWhat(4,1), SetWho(4,1)`} .

## 5 AI Agents

One of the primary aims of Ludii is to facilitate the implementation of general game playing agents, and their evaluation in a wide variety of game. To this end, Ludii contains a number of default agent implementations, and provides an interface for the development of third-party agents.

### 5.1 Default AI Agents

The default agents implemented in Ludii are:

- **Random**: an agent that samples actions uniformly at random.
- **Monte-Carlo (flat)**: an agent that estimates the values of actions available in the root node using a flat Monte-Carlo search (i.e. uniformly random playouts), and selects the action with the maximum estimated value.
- **UCT**: a standard UCT implementation [12, 9, 3]. An open-loop Monte-Carlo tree search (MCTS) approach [14] is used in stochastic games.
- **MC-GRAVE**: an implementation of Generalized Rapid Action Value Estimation [8].
- **Biased MCTS**: a variant of MCTS that uses simple patterns as features for state-action pairs to bias [16] the selection and playout phases of MCTS.

### 5.2 Third-Party AI Support

Ludii provides an interface for the implementation of new agents, which can subsequently be imported into Ludii's GUI and used to play any Ludii game. Programmatic access to Ludii's game is also available, which allows for convenient evaluation of custom algorithms using Ludii's wide array of implemented games. Example implementations are available on github.[2]

## 6 Ludii Player

In this section we describe the Ludii player, that provides the front-end interface for accessing the complete functionality of the Ludii system. Some of the main highlights of the Ludii Player include:

- A graphical interface for playing hundreds of traditional and modern strategy games, both locally and online internationally, with other players from around the world.
- A variety of included general game playing algorithms (UCT, Flat-MC GRAVE, etc.) with comprehensive evaluation metrics and visualisation options, as well as the ability to integrate third-party agents.
- Tools for creating, playtesting, and sharing your own game designs, defined using the Ludii general game language.

[2] `https://github.com/Ludeme/LudiiExampleAI`.

### 6.1 Game Playing

One of the key advantages of Ludii over other previous general game playing systems such as GGP, is the ability to view and interact with all implemented games via a sophisticated graphical environment. This allows human users to play and enjoy any game created within Ludii, whilst also making tasks such as correcting bugs and identifying incorrect rule descriptions much easier to perform. The main graphical interface provided by the Ludii player is shown in Figure 1. The left side of the view shows the main playing area of the current game, the top right section provides information about the games's players (name, colour, score, components in hand, etc.), and the bottom right area provides additional supplementary information about the game (moves made, game description, AI analysis, etc.).

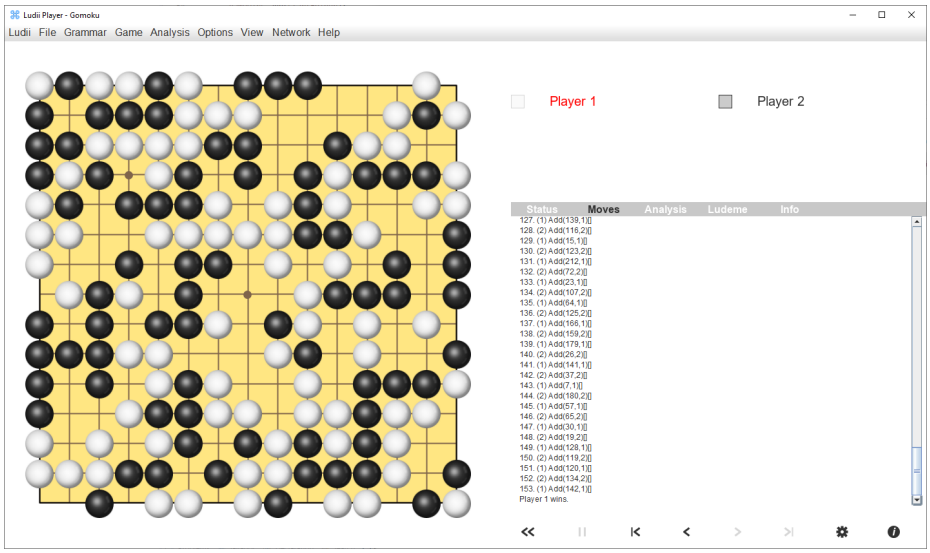



Fig. 1: The Ludii Player interface, showing a completed game of Gomoku.

The Ludii player currently allows up to 8 players (both human and AI) to play games against each other, either on a single Ludii application or else using multiple applications within a single local network. By registering for a free Ludii user account, Ludii games can also be played internationally with other human players online. A large and active community of players from many different demographics and geographic regions, may also provide valuable insight into the game playing preferences and abilities of different cultures.

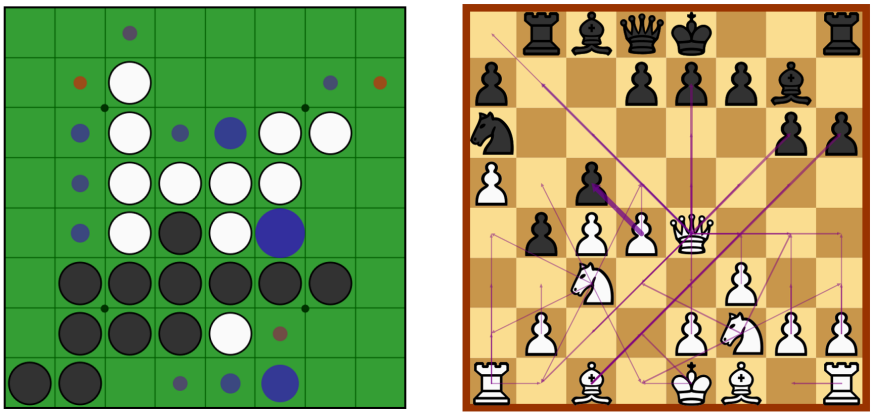
The Ludii Player also includes many customisable options for improving the overall game experience. Examples include changing the colours of the board and pieces, visualising the game's mathematical graph, showing the axes or coordinates of the game board, displaying the possible moves that can be made

by each player, providing a complete list of all moves made, the ability to undo moves, and analysis on the current winning likelihood of each player.

### 6.2 Agent Evaluation

As well as allowing humans to play games, the Ludii Player also contains several features that make it easier to evaluate and analyse the performance of different agents. Any player within a game can be controlled by one of the provided game playing algorithms that are included with Ludii. Analysis provided by these agents, such as their iteration count, child mode visits, and value estimates, is provided directly within the Ludii Player. It is also possible to visualise the distribution of possible moves for each agent at any given game state, see Figure 2, providing a graphical representation of the AI "thought process". Moves that involve adding a new piece into the game are represented by a dot, see Figure 2a, whilst those that involve changing the position of an existing piece are represented by arrows, see Figure 2b.

The size of either the dot or arrow for each possible move represents how much the AI is thinking about that move (playout number), whilst the colour indicates the average score obtained from playouts after making this move (red = low win likelihood, blue = high win likelihood, purple = neutral win likelihood). These features makes it easier to identify the different playing abilities of general game playing agents, and and can also provide a useful teaching tool for explaining how certain algorithms search the available action space.



(a) Reversi - black to move. (b) Chess - white to move.

Fig. 2: AI visualisations for two example games (Reversi and Chess) showing the outcome likelihood (colour) and number of playouts (size) for each move.

### 6.3 Manual Game Creation

The Ludii player also provides many useful tools for aiding with the creation and testing of games using the Ludii language. A complete game editor for the Ludii Player is currently under development and will allow designers to adjust certain properties or ludemes of the current game's description directly within the the Ludii app, with the resulting changes being compiled and applied automatically.

A large number of board and piece designs will be included within the Ludii player for game designers to use – see Figure 3 for examples – but it will also be possible to specify your own piece and board images within Ludii game descriptions. This will provide a wide range of possibilities for both the rules and visuals of a created game. These game descriptions can the be loaded into any Ludii application, allowing designers to easily share their created games with other Ludii users.

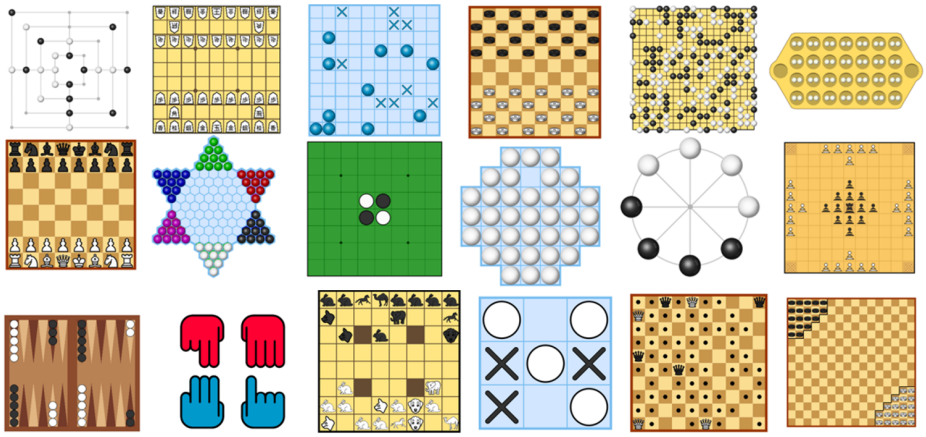


Fig. 3: Thumbnails for some of the games provided with the Ludii Player.

## 7 Ludii Portal

The Ludii Portal website, hosted at the URL `www.ludii.games`, provides additional information and services beyond those offered by the main Ludii system. Some of these services are not yet available at the time of writing, but will be added to the Ludii Portal over the coming months.

**Library** The Ludii Game Library provides a wide range of computational and historical information on the complete collection of official Ludii games, see Figure 4. This includes diagrams, rule descriptions, strategies, tutorials, mathematical and social profiles, geographical regions, time periods, cultural importance, game reconstructions, and much more.

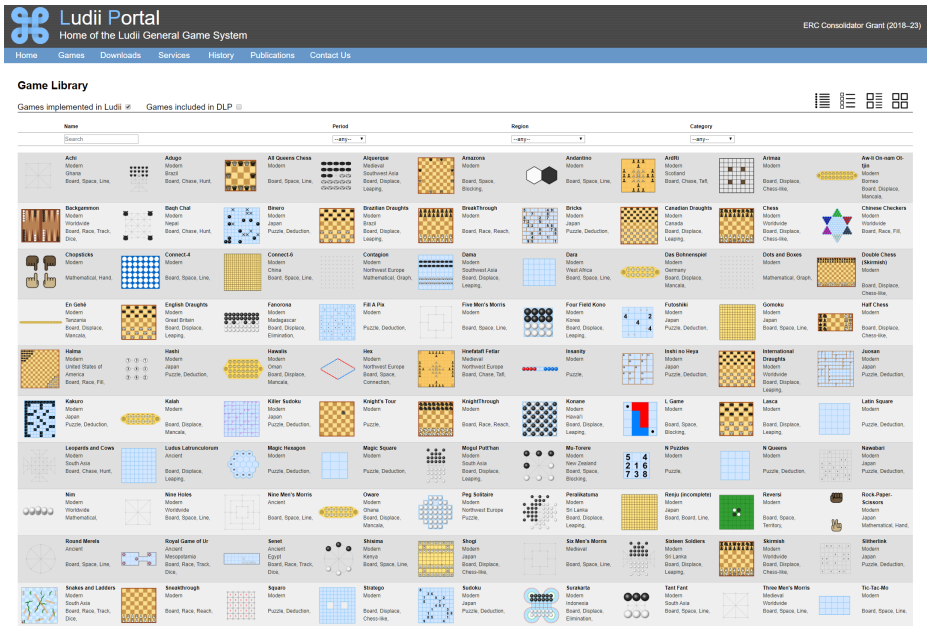



Fig. 4: The Game Library page of the Ludii Portal.

**Forum** The Ludii Forum offers a dedicated space to discuss any subject related to Ludii and the DLP. This may include discussions about the latest research on general game AI, archaeological finds from recent excavations, promotion and sharing of new game descriptions, the results of Ludii competitions, recommendations for new games to include within the official Ludii game repository, and whatever else the Ludii community feels is important to discuss.

**Competitions** We plan to run several general game AI competitions using Ludii over the following years [17]. This includes many AI competitions that focus on the development of autonomous general game playing agents, procedural content generators, and data mining algorithms. In addition to this, we aim to organise a handful of non-AI related competitions focusing on human playing abilities and game design. Such competitions will hopefully stimulate conversation on the Ludii forum, and would likely rely on the cooperation of a large number of Ludii users to compete, playtest and evaluate submitted entries.

**Game Recommendations** Another service that will be offered by the Ludii Portal will be personalised game recommendations. These recommendations will be based on user personal information, prior game results within the Ludii Player, regional and cultural data, and other factors that may influence an individual's game preference. The more users that participate in this game recommendation system, the more accurate our suggestions will be.

## 8 Planned Services

Ludii provides a platform for many potential game design services, including the following, which we plan to provide over the course of the DLP.

### 8.1 Automated Game Design

Games might be generated automatically in a number of modes:

- **Unconstrained:** New games might be generated through standard search techniques (e.g. evolutionary methods or hill-climbing techniques) using the provided database of known games as a starting point.
- **Directed:** New games might be generated by directed search according to metrics, conditions or desired behaviours specified by the user.
- **Bespoke:** Games might be generated for individual users based on implicit preferences inferred from player behaviour.

### 8.2 Game Optimisation

Ludii has already proven to be a useful tool for automated play-testing to detect imbalances and other flaws in candidate rule sets. This may in future be coupled with intelligent rule modification to optimse rule sets in order to reduce or (ideally) remove flaws.

### 8.3 Historical Game Reconstruction

One of the most important services offered by Ludii will be the facility to perform reconstructions of historical games based on partial or unreliable information. This includes taking material evidence in the form of (possibly partial) game boards and pieces, and inferring likely rule sets based on the geographical, historical and cultural context of the evidence, using historical data accumulated during the course of the DLP through archival and on-site research [2]. The aim is to produce likely reconstructions that maximise historical authenticity as well as quality of play, and to provide a tool to help traditional games researchers in the difficult reconstruction process.

## 9 Conclusion

The Ludii general game system, while being developed to address the needs of the larger Digital Ludeme Project, has the potential to be a significant and useful software tool in its own right. It has been designed to allow the description of as wide a range of (mostly traditional) games as easily as possible, and to provide a platform for a range of game analysis and design services that games researchers will hopefully benefit from. Ludii will continue to mature and expand in functionality as the DLP progresses.

**Acknowledgements.** This research is part of the European Research Council-funded Digital Ludeme Project (ERC Consolidator Grant #771292) run by Cameron Browne at Maastricht University's Department of Data Science and Knowledge Engineering.

## References

1. Borvo, A.: Anatomie D'un Jeu de Cartes: L'Aluette ou le Jeu de Vache. Librarie Nantaise Yves Vachon, Nantes (1977)
2. Browne, C.: AI for ancient games. Künstliche Intelligenz (2019)
3. 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 Transactions on Computational Intelligence and AI in Games 4(1), 1–49 (2012)
4. Browne, C., Togelius, J., Sturtevant, N.: Guest editorial: General games. IEEE Transactions on Computational Intelligence and AI in Games 6(4), 1–3 (2014)
5. Browne, C.B.: Automatic Generation and Evaluation of Recombination Games. Phd thesis, Faculty of Information Technology, Queensland University of Technology, Queensland, Australia (2009)
6. Browne, C.B.: A class grammar for general games. In: Advances in Computer Games. LNCS, vol. 10068, pp. 167–182. Leiden (2016)
7. Burge, W.H.: Recursive Programming Techniques. Addison-Wesley, Boston (1975)
8. 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 (2015)
9. 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. LNCS, vol. 4630, pp. 72–83. Springer Berlin Heidelberg (2007)
10. Fowler, M., Parsons, R.: Domain-Specific Languages. Addison-Wesley, Boston (2011)
11. Hall, P.W.: Parsing with C++ constructors. ACM SIGPLAN Notices 28(4), 67–69 (1993)
12. Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) Machine Learning: ECML 2006, LNCS, vol. 4212, pp. 282–293. Springer, Berlin, Heidelberg (2006)
13. Love, N., Hinrichs, T., Genesereth, M.: General game playing: Game description language specification. Tech. Rep. LG-2006-01, Stanford Logic Group (2008)
14. Perez, D., Dieskau, J., Hünermund, M., Mostaghim, S., Lucas, S.M.: Open Loop Search for General Video Game Playing. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 337–344. ACM (2015)
15. Schaul, T., Togelius, J., Schmidhuber, J.: Measuring intelligence through games. CoRR abs/1109.1314 (2011), `http://arxiv.org/abs/1109.1314`
16. Soemers, D.J.N.J., Piette, É., Browne, C.: Biasing MCTS with features for general games. In: Proceedings of the 2019 IEEE Congress on Evolutionary Computation (CEC 2019). pp. 442–449 (2019)
17. 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)