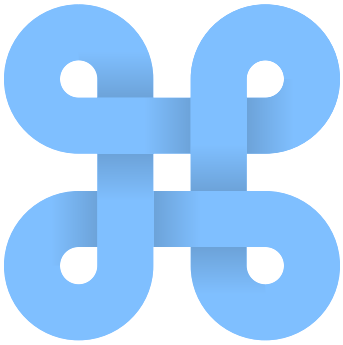


Ludii User Guide

Ludii Version 1.3.2

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

Department of Data Science and Knowledge Engineering (DKE)
Maastricht University
Maastricht, the Netherlands

February 18, 2022

Introduction

This user guide describes the basic functionality of Ludii { a \general game system" that can be used to play a wide variety of games. Ludii is a program developed for the ERC-funded Digital Ludeme Project, in which mathematical and computational approaches are used to study how games were played, and spread, throughout history.

The source code of Ludii is available at https://github.com/Ludeme/Ludii

Primary features of Ludii include:

- A large library of implemented games, which can be played by any combination of human and computer players, and displayed graphically in the application.
- A new *ludeme*-based game description language, which allows for new games to be modelled in more succinct and human-friendly formats than prior game description languages, and run more e ciently.
- Implementations of common game-playing algorithms from the Arti cial Intelligence literature, and the ability to write and import custom agents in the application.

Ludii primarily contains board game, dice games, card games, puzzles, and other abstract games.

To remain up-to-date with further developments of Ludii and the Digital Ludeme Project, keep an eye on the websites and twitter pages for each of them:

Ludii website: http://ludii.games/

Ludii Twitter: https://twitter.com/ludiigames

Digital Ludeme Project website: http://www.ludeme.eu/

Digital Ludeme Project Twitter: https://twitter.com/archaeoludology/

Please contact the authors with any comments or corrections at: ludii.games@gmail.com

Contents

1 Installation and Running

1.1 Prerequisites

Ludii requires Java version 8 or higher to be installed on your computer. Java can be downloaded from: https://www.java.com/download/. Ludii should run correctly on any major operating system. It has been verified to run correctly on the following operating systems:

Windows 10.

OS X El Capitan 10.11.6, OS X Mojave 10.14.3.

Ubuntu 16.04, Ubuntu 18.04, Ubuntu 19.04, Ubuntu 20.04.

1.2 Installation

The latest version of Ludii can be downloaded from: https://ludii.games/download. The downloaded file will be named `Ludii-X-Y-Z.jar`, where `X-Y-Z` describes the version.

Additional installation steps after downloading the file are not required. Because Ludii may write additional files in the directory containing the `.jar` file, we recommend placing it in a directory that is otherwise unused (for example: `C:/Downloads/Ludii/Ludii-X-Y-Z.jar`).

1.3 Running

The easiest way to launch Ludii is to double-click the downloaded `.jar` file. Alternatively, it may be launched by navigating to the directory containing the `.jar` file in a command prompt, and entering:

```
java -jar Ludii-X-Y-Z.jar
```

When launching the application for the first time, it will load the game *Surakarta* by default, and present the screen depicted in Figure 1.1.

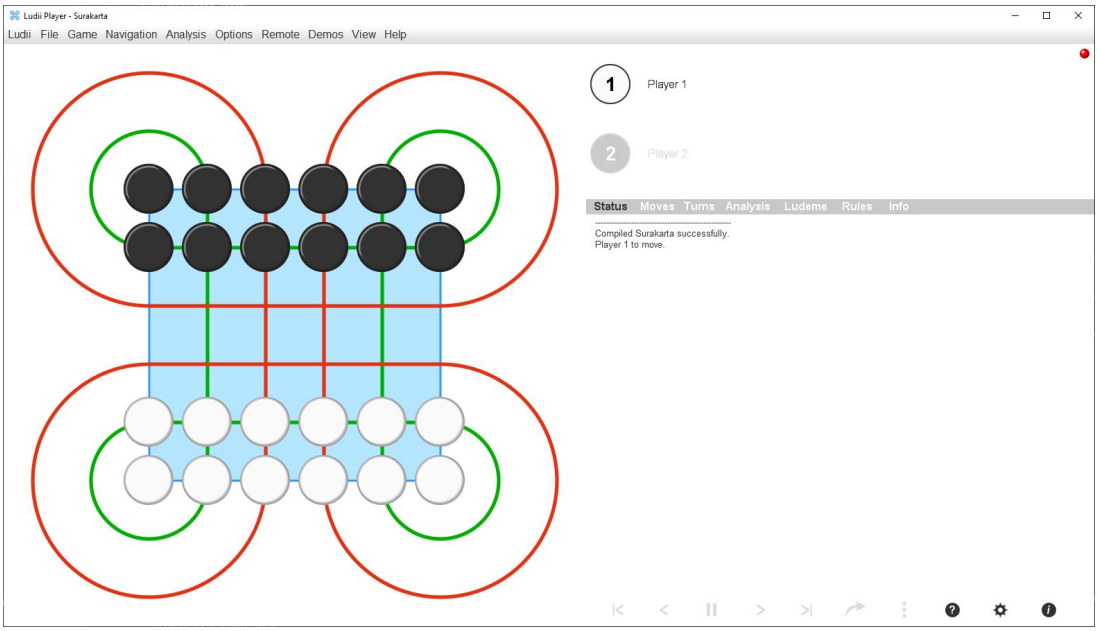


Figure 1.1: The initial screen presented by Ludii, with the game *Surakarta*.

2
Basic Usage

Basic usage of Ludii primarily involves loading and playing any of the games included in its default library of games.

2.1 Overview of the User Interface

The graphical user interface (GUI) of Ludii can be split up in four main areas, in addition to its menubar.

2.1.1 Main Areas

The four main areas are depicted in coloured (and numbered) rectangles in Figure 2.1.

Playing Area The *playing area* (area 1 in Figure 2.1) is the main area that players interact with when playing games. In most games, it depicts a board with any placed pieces. In games that do not use a board, it may just display pieces that have been placed on the \table". In most games, human players can select their moves by clicking and/or dragging pieces in this area.

Players' Area The *players' area* (area 2 in Figure 2.1) shows basic information about the players, such as their names and colours. Clicking any player's circle (displaying their number and colour) opens a dialog that can be used to adjust various preferences (see **??** and sections 2.1.3 and 2.1.4). In some games it may also contain the outcomes of dice rolls, or depict \hands" containing pieces that players may be able to drag to the playing area. An example of a game where this area contains dice (*Backgammon*) is depicted in Figure 2.2. An example of a game where this area contains pieces that may be dragged onto the board (*Three Men's Morris*) is depicted in Figure 2.3.

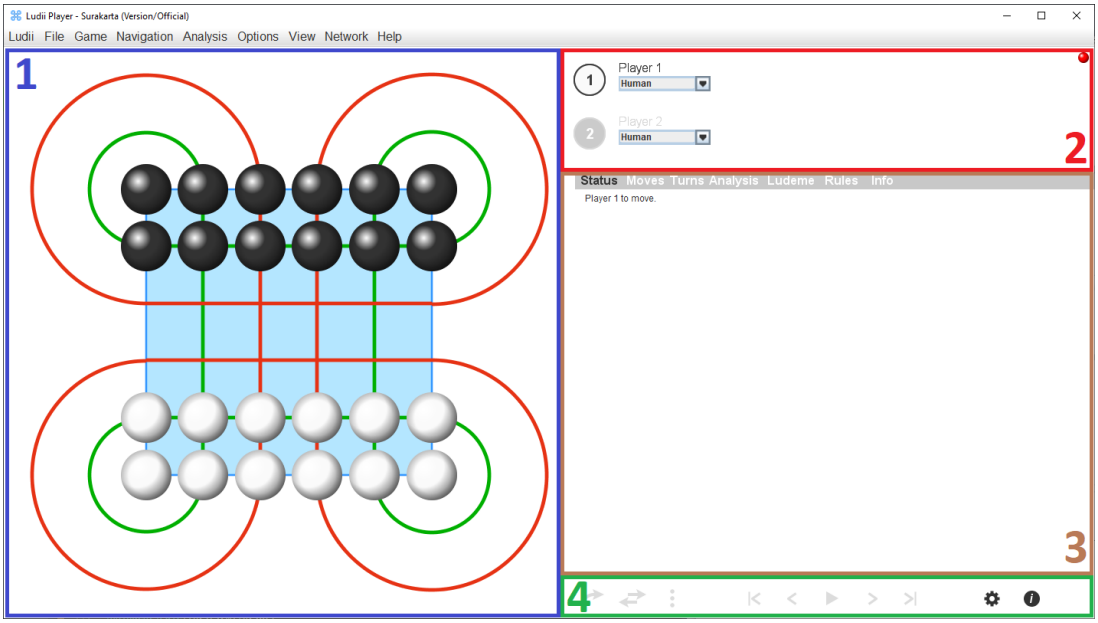


Figure 2.1: The graphical user interface (GUI) of Ludii. Coloured rectangles mark the playing area (1), the players' area (2), the panels area (3), and the buttons area (4).

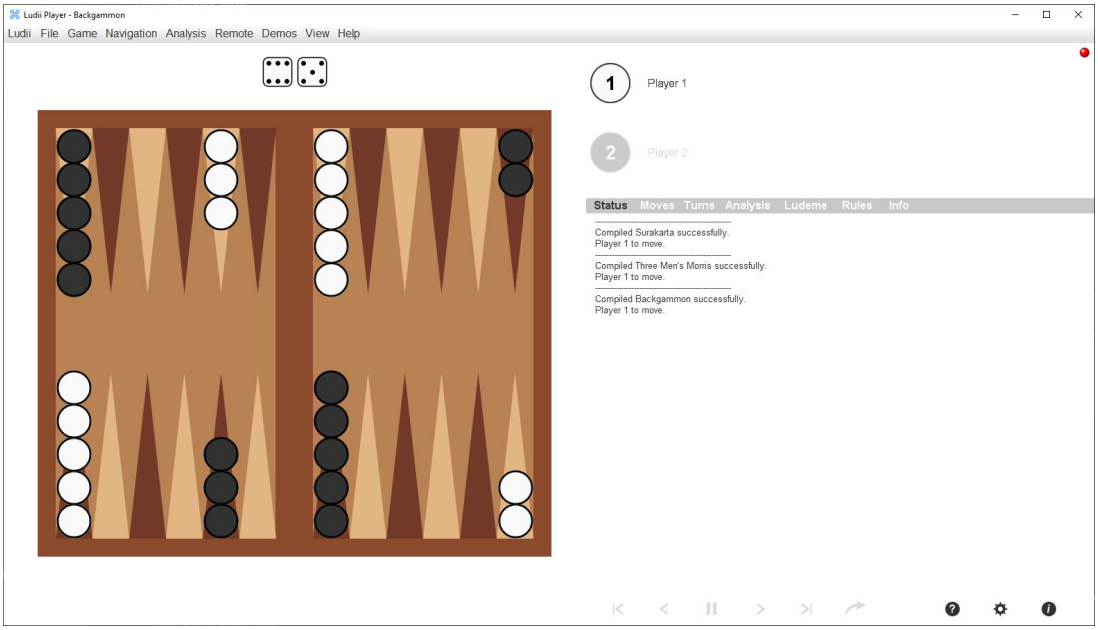


Figure 2.2: The game *Backgammon* in Ludii. The players' area shows two dice that have been rolled.

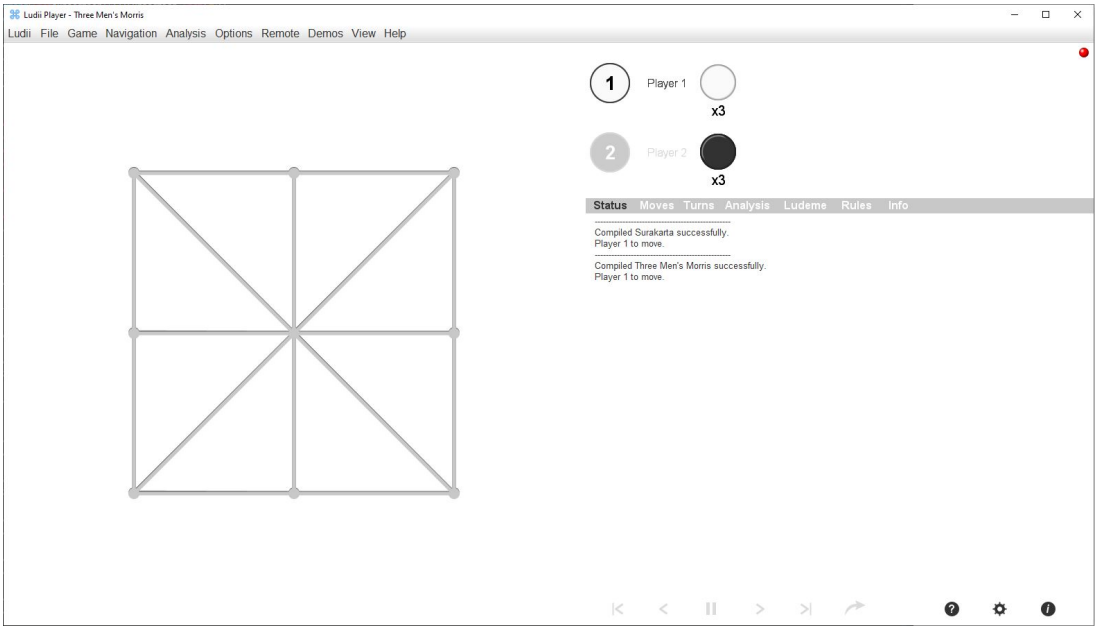


Figure 2.3: The game *Three Men's Morris* in Ludii. The players' area shows pieces that can be dragged onto the board by their owners.

Panels Area The *panels area* (area 3 in Figure 2.1) contains a panel in which Ludii displays information. The panel has the following tabs:

Status: this tab displays the status of the current game being played. This includes, for example, showing which player is the current player to move, or showing the winner of a game when it is nished.

Moves: this tab displays the sequence of moves that has been applied in the current game. This information is primarily intended for developers of Ludii or third-party AI players.

Turns: this tab displays the coordinates of positions where moves were made, along with the turn numbers in which they were made. If a player makes multiple moves in a row, all those moves together are treated as a single turn.

Analysis: some AI players print information about their processing to this tab.

Ludeme: this tab displays the game description of the currently loaded game, in its *ludeme*-based format (see Chapter 3).

Rules: this tab provides an English description of the rules as implemented and loaded in Ludii.

Info: this tab provides additional information about the currently loaded game.

Buttons Area The *buttons area* (area 4 in Figure 2.1) may, depending on the context, display any of the following buttons:

: this button can be used to pass, and move on to the next mover, in games where this is legal.

⋮ : this button opens up a dialog that allows for the selection between miscellaneous moves that cannot be made otherwise in the playing area (such as placing bets in *Morra*).

▶ : if the current player to move has been set as an AI player (see Subsection 2.1.3), this button can make it start playing. Any subsequent AI movers will automatically continue playing.

❚❚ : if any AI players are currently playing, this button can be used to pause them.

|< : if there is a current game in progress, this button winds back to the start of the game. The application will remember the moves that were selected thus far, and will allow re-applying them.

< : if there is a current game in progress, this button takes a single step back (e.g. to the previous turn in a turn-based game).

> : this button takes a step forward (e.g. re-applies one move) if we previously stepped backwards through a game in progress.

>| : this button re-applies any moves that we previously stepped backwards through in a game in progress.

⚙ : this button opens the same dialog with settings that can also be opened by clicking any player name (see sections 2.1.3 and 2.1.4).

ⓘ : this button opens the \About" dialog of Ludii, which provides information about the version number, authors, acknowledgements, etc.

2.1.2 Menu Bar Options

The menu bar at the top of Ludii's user interface contains various menus with additional options. This subsection provides short explanations for all options that can be found in these menus. Some of the more advanced options are explained in more detail in subsequent sections and chapters. Most of these options can also be accessed using keyboard shortcuts, which are listed in Appendix A. Some menus are not always accessible, but only in relevant contexts (e.g., the Puzzles menu is only accessible if a deduction puzzle is loaded).

Ludii The Ludii menu has two simple options:

Preferences: opens a dialog with various preferences and settings.

Quit: closes Ludii.

File The File menu contains options for loading games, as well as saving and loading games in progress:

Load Game: opens Ludii's game loader (depicted in Figure 2.4), which allows for any of Ludii's built-in games to be loaded. The games are sorted in di erent categories, very much like a le browser with les sorted in directories. The list of displayed games may be ltered by using the search bar, to search games by name. Many games also have a list of \aliases", which are typically alternative names or names in di erent languages. Games will still be visible in the game loader if the text typed in the search bar matches one of their aliases.

Load Recent: provides a list of games that were recently loaded on this computer, allowing for them to be easily selected and loaded again.

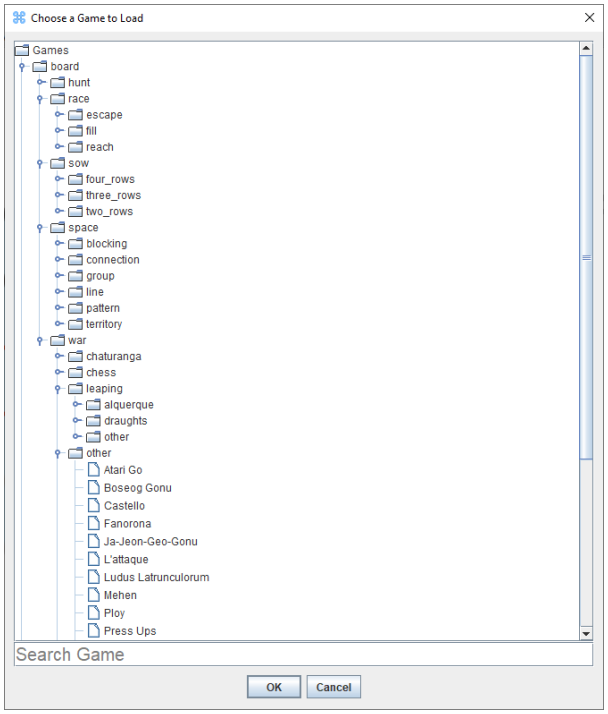


Figure 2.4: Ludii's Game Loader.

Load Game from File: opens a standard file browser, which can be used to navigate to any directory on your computer and load a game description from a `.lud` file. This may be used to load custom games that were not originally included in the Ludii download (see Chapter 3). Note that, for security reasons, we recommend only loading files provided by trusted sources!

Load Random Game: loads one randomly selected game from all the built-in games of Ludii. This can be a fun way to discover new games.

Load Trial: opens a file browser that can be used to load a previously saved \trial", which is a record of a game in progress. This may be used to resume playing a game that was previously saved.

Save Trial: opens a file browser that can be used to save the progress of the current game to a file, allowing it to be loaded and resumed in the future.

Editor (Expanded): opens an editor in which the expanded version of the currently loaded `.lud` file can be viewed, edited, and saved (optionally as a new file). In this expanded version, advanced features such as *de nes* have already been processed and expanded.

Editor (Short): opens an editor in which the short version of the currently loaded `.lud` file can be viewed, edited, and saved (optionally as a new file). In this short version, advanced features such as *de nes* remain unprocessed.

Game The Game menu contains some options for interaction with the current game:

Restart: restarts the current game.

Random Move: randomly selects one random move from all moves available in the current game state, and applies it.

Random Playout: starting from the current game state, rolls out all the way to a terminal game state by selecting a random sequence of legal moves.

Resign Game: resign current game (only available in network play).

Propose/Accept a Draw: propose to declare a game drawn, or accept such a proposal (only available in network play).

List Legal Moves: prints all the legal moves in the current game state to the Status tab.

Game Screenshot: takes a screenshot of the playing area, and saves it to a new `.png` file in the directory that contains Ludii's `.jar` file (or the current working directory if Ludii was launched from a command line).

Auto From Moves: if this option is toggled on, Ludii will automatically move a piece to a location that the user clicks on if there is only a single move possible towards the clicked location. In other cases (e.g. if there are multiple possible moves to the same location), the user will still be required to drag the piece to be moved, or click on the location to move from before clicking on the location to move to.

Auto To Moves: if this option is toggled on, Ludii will automatically move a piece that the user clicks on to a new location if there is only a single move possible from the location that was clicked on. In other cases (e.g. if there are multiple possible moves from the same location), the user will still be required to drag the piece to be moved, or click on the location to move from before clicking on the location to move to.

Cycle Players: cycles through assignments of players (colours, names, types, etc.) to player numbers (1, 2, 3, etc.).

Generate Grammar: prints the grammar that defines the language used by Ludii to model games in an extended Backus-Naur form (EBNF)-like notation. The output is written to the standard output stream of the Ludii process, and to the status tab.

Count Ludemes: counts and prints the number of Ludemes available to Ludii. The output is written to the standard output stream of the Ludii process, and to the status tab.

Navigation The Navigation menus contain options to navigate through a single game (stepping backwards and forwards, starting and pausing play, etc.):

Play/Pause: starts or pauses gameplay.

Previous Move: if there is a current game in progress, takes a single step back (e.g. to the previous turn in a turn-based game).

Next Move: takes a step forward (e.g. re-applies one move) if we previously stepped backwards through a game in progress.

Go To Start: if there is a current game in progress, winds back to the start of the game. The application will remember the moves that were selected thus far, and will allow re-applying them.

Go To End: re-applies any moves that we previously stepped backwards through in a game in progress.

Puzzle The puzzle menu contains various options that are speci c to deduction puzzles. This menu is only available if the game currently loaded in Ludii is a deduction puzzle:

Value Selection: allows players to choose how they prefer to select values in deduction puzzles. Ludii can respond to clicks by cycling through possible values, opening a dialog with all possible values, or automatically deciding between these two options for the selected game.

Illegal Moves Allowed: if checked, Ludii will enable the user to select values that are illegal in a deduction puzzle.

Show Possible Values: if checked, Ludii will show the possible values that may be selected in deduction puzzles in the GUI.

Analysis The Analysis menu contains options to produce various measurements for purposes of analysis:

Estimate Branching Factor: estimates the average branching factor of the loaded game by running random playouts of the game for 30 seconds.

Estimate Game Length: estimates the average duration of the loaded game by running random playouts of the game for 30 seconds.

Estimate Game Tree Complexity: estimates the game tree complexity of the loaded game by running random playouts of the game for 30 seconds.

Estimate Game Tree Complexity (No State Repetition): estimates the game tree complexity of the loaded game by running random playouts of the game for 30 seconds. Within these playouts, players will not be allowed to make moves that lead to game states that have already been previously encountered in the same game.

Evaluation Dialog: opens a dialog with various options to evaluate various metrics of a game such as the average duration per game, win rates for every player number, etc.

Time Random Playouts: continuously runs full, random playouts in the selected game, over a period of fourty seconds. The average number of full playouts per second, measured over the last thirty[1] seconds, are reported in the Status tab of the panels area. This gives a rough[2] indication of the e ciency of any game's implementation in Ludii.

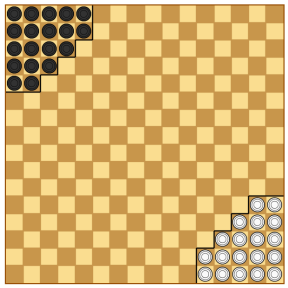
Time Random Playouts in Background: runs the same procedure as described above, but in a background thread such that the GUI is not blocked. Note that this can make the timings less reliable.

Evaluation UCT 10 trials: performs an analysis of the game by running 10 trials between UCT agents (such an evaluation could also be set up manually using the *Evaluation Dialog*).

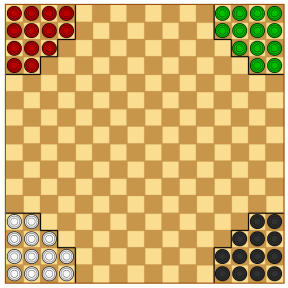
Evaluation UCT 30 trials: performs an analysis of the game by running 30 trials between UCT agents (such an evaluation could also be set up manually using the *Evaluation Dialog*).

[1] The rst ten seconds are used to \warm up" the Java Virtual Machine.

[2] To obtain more accurate numbers, for instance for academic publications, we recommend timing playouts without a GUI, and measuring over a longer period of time.



(a) *Halma* with two players.



(b) *Halma* with four players.

Figure 2.5: Ludii has two options available in *Halma*.

Evaluation UCT 50 trials: performs an analysis of the game by running 50 trials between UCT agents (such an evaluation could also be set up manually using the *Evaluation Dialog*).

Options The Options menu is not always visible, but only if the currently selected game has a set of options in its *ludeme*-based game description. The set of options that are available also varies from game to game. The most common options are to select di erent board sizes, rules, or numbers of players. For example, in the implementation of *Halma* included in Ludii, there are options to play with two players or with four players. This is depicted in Figure 2.5.

View The View menu provides a variety of settings to display extra information in Ludii's user interface:

Show Board: if turned o , Ludii will not draw game boards.

Show Pieces: if turned o , Ludii will not draw game pieces.

Show Graph: if turned on, Ludii draws the graph representation of the board.

Show Cell Connections: if turned on, Ludii draws the relations between cells (orthogonal relations as lines, and diagonal relations as dashed lines).

Show Axes: if turned on, Ludii adds extra labels around boards which may be useful to refer to speci c locations when playing with other humans. For instance, in *Chess* it labels the rows with numbers 1 through 8, and columns with letters A through H.

Show Possible Moves: if turned on, Ludii draws a blue dot on every location that may be clicked for a legal move in the current game state. If a move involves two locations (for instance to move a piece from one location to another in a game like *Chess*), Ludii will draw a red dot on every valid destination after a starting location has been selected by clicking on it. Note that valid locations may not only be located in the playing area, but sometimes also in the players' area (see, for instance, *Three Men's Morris*).

Show Last Move: if turned on, Ludii highlights the previous move made in a game with a yellow circle or arrow.

Show Indices: if turned on, Ludii draws a unique number in every cell or vertex (depending on which of the two is primarily used for gameplay) in the current game. This is primarily intended for developers of Ludii, third-party games, or AI players for Ludii.

Show Coordinates: if turned on, Ludii draws labels { constructed from the labels shown around boards by the \show axes" option { in every cell or vertex.

Show AI Distribution: if turned on, depending on which AI players are currently active, Ludii may visualise their \thought process".

Show Tracks: in games with tracks (such as many racing games), this can be used to visualise the tracks that di erent players move along.

View SVG: opens a dialog that can be used to view every SVG image that is included and available in Ludii.

Load SVG: opens a le chooser that can be used to select any arbitrary SVG image and view how it would look if rendered by Ludii.

Remote The Remote menu allows for users to connect to each other and play Ludii games over a network. A more detailed explanation of this can be found in Section 2.4.

Developer The developer menu provides access to advanced options that are primarily intended for Ludii developers or advanced users, such as game designers using Ludii to design and implement new games. This menu only becomes visible after enabling it in the \advanced" tab of the Settings dialog:

Compile Game (Debug): opens the game loader to select a game. The selected game will be compiled in debug mode, providing verbose output about the compilation process to the standard output stream.

Expanded Description: opens the game loader to select a game. The expanded description (with advanced features such as *de nes* already having been processed and expanded) of the selected will be printed to the *Status* tab and the standard output stream.

Metadata Description: opens the game loader to select a game. All of the metadata of the selected game's description le will be printed to the *Status* tab and the standard output stream.

Generate Symbols: prints a list of all the symbols that may be used in Ludii game description les to the *Status* tab and the standard output stream.

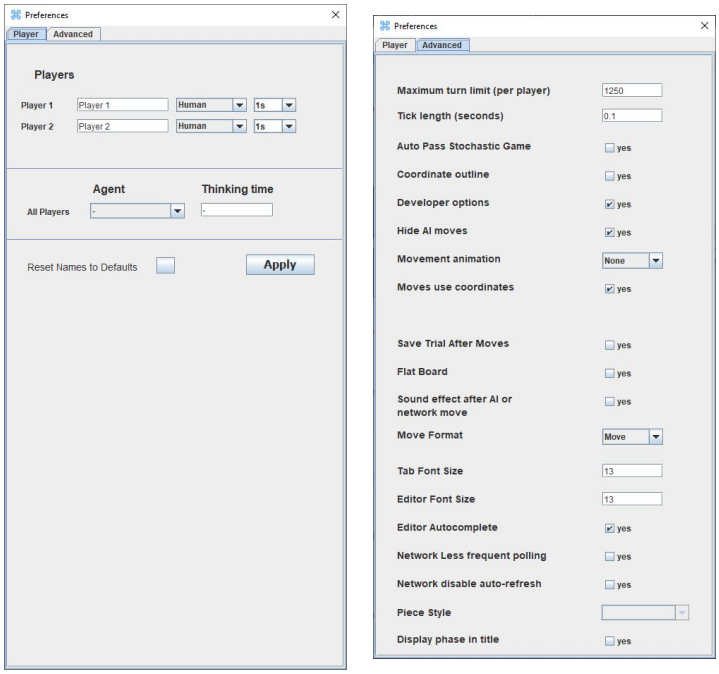
Jump to Move: shows a dialog prompting the user to enter a number. Ludii will jump (forwards or backwards) to the move corresponding to the entered number within the current trial.

Show Board Shape: if checked, draws an outline depicting the board shape used in the current game.

Show dev tooltip: if checked, shows tooltips with detailed information when mousing over pieces.

Swap Rule: if checked, enables the swap rule (or \pie rule") in the current game.

No Repetition Of Game State: if checked, the current game will not allow players to make moves that reproduce a previously-encountered game state.



(a) Player preferences.

(b) Advanced preferences.

Figure 2.6: Di erent dialogs for various preferences in Ludii.

No Repetition Within A Turn: if checked, the current game will not allow players to make moves that reproduce game states that were previously encountered within the same turn (i.e. no moving back and forth without allowing any other players to get any turns).

Sandbox Mode (Beta): enables the sandbox mode, in which the user can make almost any arbitrary modi cations to the game state. Note that this feature is in beta mode, and may fail in many situations. In particular, there is no guarantee that a game's rules will still be applied correctly after exiting the sandbox mode in game states created using the sandbox mode.

Show Cell Indices: if turned on, Ludii draws a unique number in every cell.

Show Edge Indices: if turned on, Ludii draws a unique number in every edge.

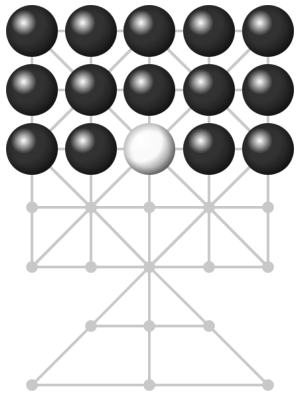
Show Vertex Indices: if turned on, Ludii draws a unique number in every vertex.

Show Cell Coordinates: if turned on, Ludii draws labels { corresponding to the axes { in every cell.

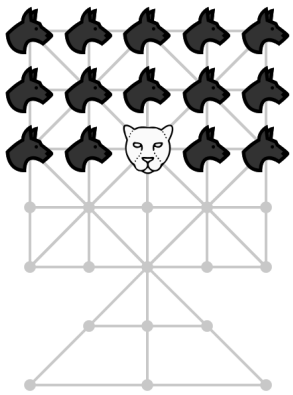
Show Edge Coordinates: if turned on, Ludii draws labels { corresponding to the axes { in every edge.

Show Vertex Coordinates: if turned on, Ludii draws labels { corresponding to the axes { in every vertex.

More Developer Options: opens a dialog with more advanced developer options.



(a) *Adugo* with \Abstract" style pieces.



(b) *Adugo* with \Themed" style pieces.

Figure 2.7: Ludii can draw pieces in *Adugo* in two di erent styles.

2.1.3 Player Preferences

The Player Preferences dialog can be used to modify player names, and to select AI algorithms and maximum thinking times for all players in a game simultaneously. AI algorithms and thinking times for individual players can be adjusted directly in the players' area of the main frame. We brie y summarise what players may expect in terms of playing strength from the di erent AI options in this section. More detailed and technical explanations of the di erent AI options can be found in Appendix B.

Ludii AI: this will automatically switch to whichever of the other AIs listed below was found to be the strongest player by our experiments given 1 second thinking time per move. This is generally the recommended AI for humans to try playing against.

Random: a simple computer player that randomly selects moves.

Flat MC: a weak AI which should be easily beaten by humans in almost every game.

UCT: a standard AI technique, for which the playing strength greatly depends on game complexity. In simple games like *Tic-Tac-Toe* it should easily be capable of perfect play with the default setting of 1 second per move, but in complex games like *Chess* it will still be easily beaten.

UCT (Uncapped): similar to the above (in most games likely slightly weaker, in others maybe slightly stronger).

MC-GRAVE: in most games, this player is expected to outperform UCT, but it is possible that it may be weaker in some games. It tends to be particularly strong in games where new pieces are only placed, and never moved or removed (such as *Hex* or *Gomoku*).

Progressive History: a variant of UCT that uses an action-based history heuristic in the selection phase of MCTS.

MAST: a variant of UCT that biases playouts based on action scores collected online.

Biased MCTS: a similar algorithm to UCT, but pre-trained to already have extra experience that biases its search process in all of the games that it is available. Therefore, it is generally expected to outperform the UCT variants (and also more advanced algorithms like MC-GRAVE), but may in some cases be weaker because it requires extra computation time for its trained biases.

MCTS (Biased Selection): a variant that is \in between" UCT and Biased MCTS. Sometimes it outperforms Biased MCTS due to being more computationally e cient, but also sometimes is outperformed by Biased MCTS due to using less bias.

Alpha-Beta: a standard AI technique (older than UCT). Tends to outperform the other approaches especially in \tactical" situations or games, such as *Chess*.

MCTS (Hybrid Selection): a variant of UCT that uses heuristic evaluation functions in playouts and as a part of the backpropagated outcomes.

Bandit Tree Search: a variant of UCT that uses no playouts, but immediately backpropagates scores from heuristic evaluation functions.

NST: a variant of UCT that uses scores collected online for action sequences (n-grams of actions) to bias playouts.

UCB1Tuned: a variant of UCT that accounts for the variance in backpropagated scores in its selection phase.

Progressive Bias: a variant of UCT that incorporates a term based on heuristic evaluation functions in its selection phase.

EPT: a variant of UCT that terminates playouts early (after 4 moves) and uses heuristic evaluation functions to backpropagate scores from non-terminal states.

EPT-QB: a variant of UCT that terminates playouts early (after 4 moves) and adds additional quality-based rewards (based on heuristic evaluation functions) to the backpropagated scores.

Score Bounded MCTS: a variant of UCT that tracks upper and lower bounds on nodes' values and can prune nodes that are provably inferior.

Heuristic Sampling: ignores a random sample of moves and directly evaluates the remaining moves with heuristic evaluation functions.

One Ply (No Heuristic): a 1-ply search without heuristic evaluation functions.

From JAR: this option may be used to load third-party AIs from `.jar` les (see Chapter 4).

2.1.4 Advanced Preferences

This dialog provides access to the following advanced settings:

Tick length (seconds): The tick time used for simulation games.

Maximum turn limit (per player): a game-agnostic turn limit. In any game run by Ludii, if max turn limit is reached, the game will end and be declared a draw for all remaining active players in the game.

Auto Pass Stochastic Game: To allow an auto pass in stochastic game when this is the only legal move.

Coordinate outline: if checked, Ludii draws an additional outline around coordinates when drawing coordinates. This can improve readability on some boards.

Developer options: if checked, the *Developer* menu with advanced developer options becomes available.

Hide AI Moves: if checked, moves made by AIs in games with hidden information are not printed to the Moves tab (except if they would be visible according to the game's rules).

Movement animation: if checked, the movement of pieces is animated instead of pieces moving instantly.

Moves use coordinates: if checked, moves printed to the Moves tab use coordinates to denote positions, rather than the unique indices of cells/vertices/edges.

Save Trial After Moves: if checked, the trial is saved after move applied

Flat Board: if checked, no 3D e ect are used for the drawing of the board.

Sound effect after AI or network move: if checked, Ludii plays a sound whenever an AI makes its move.

Move Format: to modify the move format in the Moves tab.

Tab Font Size: to modify font size of the tabs.

Editor Font Size: to modify font size in the editor.

Editor autocomplete: if checked, the autocompletion is activated for the editor.

Network Less frequent polling: if checked, less message will be printed when a game is played remotely.

Network disable auto-refresh: if checked, refreshing the game played remotely becomes manual.

Piece Style: to modify the style of the pieces.

Display phase in title: if checked, the name of the current phase of the game is printed in the title of the frame.

2.2 Playing Games with Human Players

By default, all players in Ludii are set to human-controlled. This means that humans who wish to play together on a shared screen can directly start playing after loading any game in Ludii. If any players were previously set to be AI-controlled, they may be switched back to Human using the drop-down menus in the players' area. If AIs are paused, human input can also be used to select moves for players that have been set to be AI-controlled without rst switching them back to Human.

The controls for most games should be intuitive; the mouse is typically used to place pieces by clicking in locations (in games like *Hex*, *Reversi*, etc.), or used to move pieces by dragging them or by clicking consecutively on the piece and the destination. It may be helpful to use the \Show Possible Moves" option from the View menu to visualise possible moves { in particular when trying new games for which you are not yet familiar with the rules!

2.2.1 Games with Hidden Information or Simultaneous Moves

Ludii contains some game with hidden information (like *L'attaque*), and games in which players are expected to select their moves simultaneously (like *Rock-Paper-Scissors*). Ludii allows for them to be played on a single screen, but this may not be ideal in practice because players may get access to information they should not have. Playing such games over the network circumvents these issues.

2.3 Playing Games with Computer Players

After setting one or more players to be AI-controlled, Ludii can be used to play against computer players (or even just watch as multiple computer players play against each other). If the Play button (▶) is visible in the bottom of the user interface, it can be used to make the AI player start playing. Any subsequent AI-controlled players will automatically continue playing when it's their turn to move, unless they are paused using the Pause button (❚❚). When playing against AIs, it may sometimes be useful to toggle on the \Show Last Move" option in the View menu, to avoid confusion about which move the previous AI-controlled player last made.

2.3.1 Games with Hidden Information

When AI-controlled players play games with hidden information, currently they will still have access to all information (even the information that should be hidden). The two simplest AI players (*Random* and *Flat MC*) are not able to abuse this, but all others are { and will likely easily beat most human players as a result. Ensuring that AI players only receive the information that they should have is still work in progress.[3]

2.4 Remote Play Over a Network

Ludii supports online network play between multiple registered users through the Remote Play Dialog, shown in Figure 2.8, which can be accessed through the Remote/Remote Play menu item. The Remote Play Dialog has three tabs, as follows.

2.4.1 Remote Play Tab

The Remote Play Tab, shown in Figure 2.8, lets you log in using your Ludii Forum account username and password,[4] in order to create and join online games and tournaments. Selecting the \Remember Me" check box will store your username locally in your preferences, but you will still need to re-enter your password each time you login. Once you have logged in, you are able to apply the remote actions and access the Games and Tournaments tabs. The remote actions include:

Send a private message to a speci c player.

[3] More precisely, we are still investigating what the most sensible way to implement a forward model in this category of games would be.

[4] If you do not have a Ludii Forum account, you can register for one at https://ludii.games/forums.

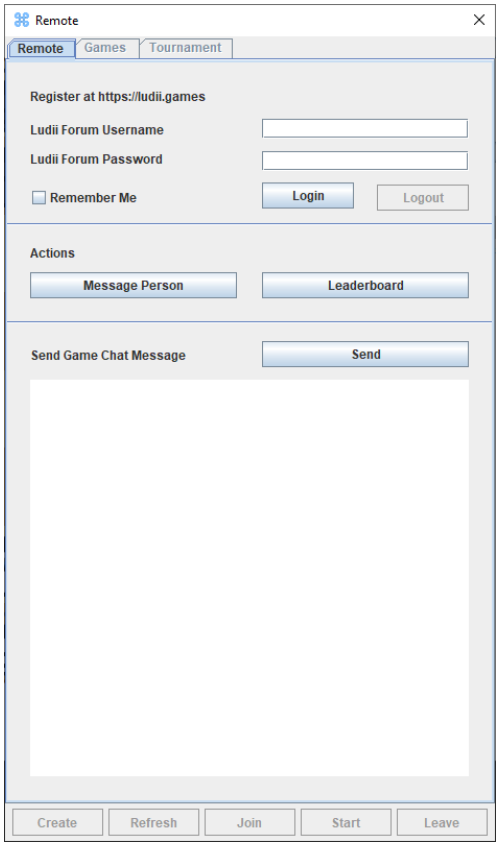


Figure 2.8: Remote Play Dialog.

Show the current Ludii leaderboard over all games, which includes basic statistics and ratings for each registered Ludii user. This information is calculated from all online games that the user has played. Statistics for speci c games can be viewed on https://ludii.games, by navigating to the game page that you wish to view, and selecting the \Leaderboard" link under the game's heading.

Send a message to all players of the current game (if you have joined one).

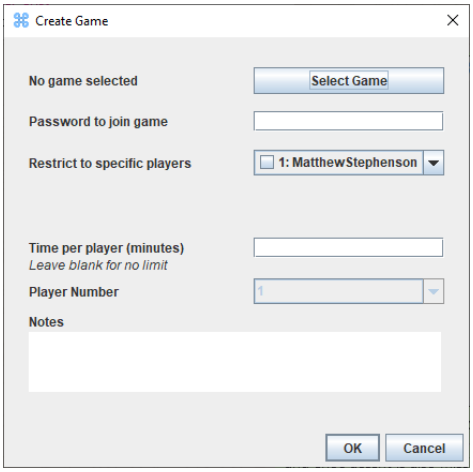
Users will see messages that are sent to them in the Status tab of their Ludii app if they are currently logged in, otherwise they will see the messages the next time they log in.

2.4.2 Remote Games Tab

The Remote Games Tab, shown in Figure 2.9a, gives you access to the remote games currently being played, and the ability to create new remote games. Three tables are shown:

1. **Games You Are Playing** Lists the remote games that you are currently an active player in. If it is your turn to move in a game, it will be highlighted in light red. To continue playing a game, select it in the table and then press the \Join" button.

(a) Remote Games Tab.



(b) Create Game Dialog.

Figure 2.9: The Remote Games Tab and Create Game Dialog.

2. **Games You Can Join** Lists the remote games that are currently waiting for more players to join. These will include:

   Public games that are open to all players.

   Private games that you have been invited to join.

   To join a game, select it in the table and then press the \Join" button.

3. **Other Games** Lists all other currently running remote games.

The columns in these tables include:

Id The unique network Id of the game.

Game The name of the game.

Time The amount of time you have left, if the game has a time limit.

Click this to select your desired player number from those remaining.

Players The number of players who have already joined the game, as well as their usernames and Elo ratings for this game.

Details Click this to view additional information about the game.

Once you have joined a remote game, some menu items will be disabled to prevent cheating. Users can resign or propose a draw for the current game by selecting the corresponding menu options in the \Game" menu. Closing the Ludii application, logging out of your account, or joining a di erent remote game will cause you to leave the current remote game.

Create Game Dialog Pressing the Create button while in the Remote Games Tab will launch the Create Game Dialog, shown in Figure 2.9b. Within this dialog you can:

Select the game (or match) you wish to create. If the game has options, then an \Options" button will appear at the top of the dialog so that you can select the desired options (e.g. choosing the desired board size).

Set a password that players must enter to join the game (optional).

Allow only speci c registered Ludii users to be able to join the game (optional).

Allow players to use AI agents to make their moves (no by default). We recommend not allowing this option for games that involve hidden information, as AI agents may be able to access details that should be kept from them (e.g. the identity or face down tiles).

Set a time limit per player for the game, in minutes (optional). Any player who runs out of time in a timed game is automatically removed from the game, as if they had resigned.

The player number that you will play in the game (Player 1 by default).

Any notes about the game that you would like prospective players or onlookers to see (optional).

Select \OK" to create the online game, with yourself as the host (and participating player). *Note:* a single user cannot host more than 100 games at a time.

2.4.3 Remote Tournaments Tab

The Remote Tournaments Tab, shown in Figure 2.10a, gives you access to the (remote) tournaments currently being played, and the ability to create new tournaments. Three tables are shown:

1. **Tournaments You Are Playing In** Lists the tournaments that you are currently participating in. To leave a joined tournament, select it in the table and then press the \Leave" button. *Note:* you cannot leave a tournament once it has started.
2. **Tournaments You Can Join** Lists the tournaments currently inviting participants to join, that you are able to join. These will include:

   Public tournaments that are open to all players.

   Private tournaments that you have been invited to join.

   To join a tournament, select it in the table and then press the \Join" button.
3. **Tournaments Hosted by You** Lists the tournaments that you created and are the host of. To start your tournament, select it in this list then press the \Start" button.

The columns in these tables include:

Id The unique network Id of the tournament.

Game The name of the game.

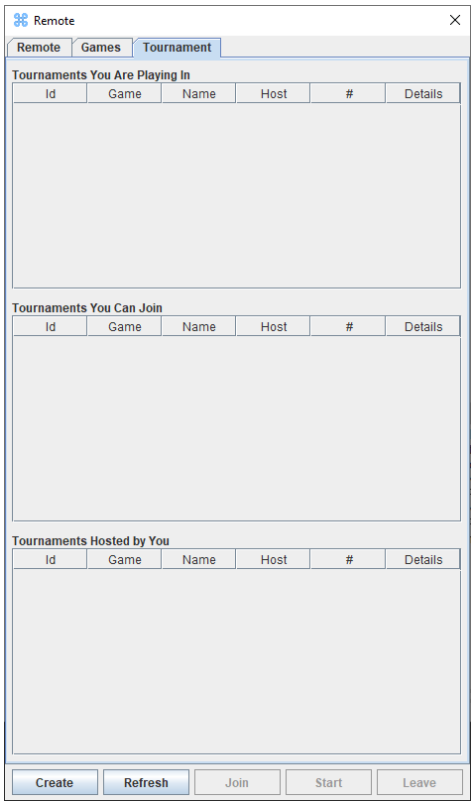
Time The name of the tournament.

Host The host of the tournament.

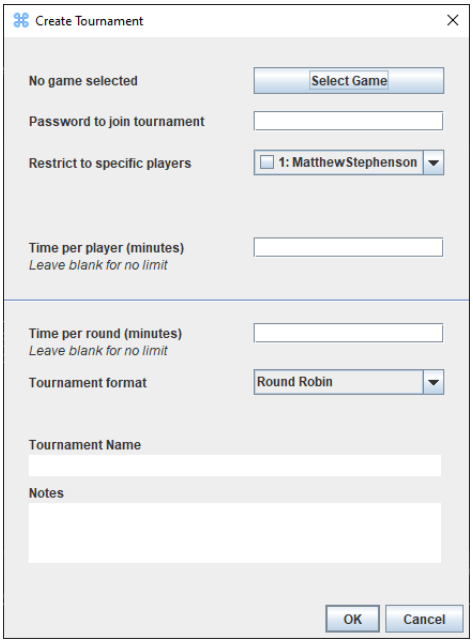
The number of players in the tournament.

Details Click this to view additional information about the tournament.

Once the host has started a tournament that you have joined, one or more games will be automatically created for the first round of the competition. The players involved in these games will be able to find them under the Remote Games Tab within the "Games You Are Playing" table. Visit the Tournaments page on ludii.games[5] to see the list of all current tournaments, and an archive of all previous tournaments.



(a) Remote Tournaments Tab.



(b) Create Tournament Dialog.

Figure 2.10: The Remote Tournaments Tab and Create Tournament Dialog.

[5] https://ludii.games/browseTournaments

Create Tournament Dialog Pressing the Create button while in the Remote Tournaments Tab will launch the Create Tournament Dialog, shown in Figure 2.10b. Within this dialog you can:

Select the game (or match) you want the tournament to be based on. If the game has options, then an \Options" button will appear at the top of the dialog so that you can select the desired options (e.g. board size, starting position, rule variant, etc.). *Note:* Currently only two-player games are supported for tournaments. We hope to add support for N-player games in the future.

Set a password that players must enter to join the tournament (optional).

Allow only speci c registered Ludii users to be able to join the tournament (optional).

Allow players to use AI agents to make their moves (no by default). We recommend not allowing this option for games that involve hidden information, as AI agents may be able to access details that should be kept from them (e.g. the identity or face down tiles).

Set a time limit per player for each tournament game, in minutes (optional). Any player who runs out of time in a timed game is automatically removed from that game, as if they had resigned.

Set a time limit for each tournament round, in minutes (optional). The rst round of the tournament starts as soon as its host presses the \Start" button. *Note:* once the time left for a round reaches a value less than the combined time limits for each player (i.e. there is a risk of the round ending before every game in the round has been completed), any games that have not yet started will begin automatically (i.e. the timer for Player 1 will begin counting down). It is therefore up to the rst player in each game to make sure they make their rst move in a game, while there is still su cient time left in the round. If your tournament is a multi-round format, then the next round is automatically started and the round timer is reset when all games in a round are completed.

Supported tournament formats include:

- **Round Robin** Every player plays every other player twice (as both Player 1 and Player 2) in a single round.
- **Swiss System** Every player plays N games against P opponents, matched according to current skill rating, over a speci ed number of rounds. An odd number of players will result in one player getting a bye each round.
- **Elimination** Players compete in a knockout tournament over multiple rounds until a single winner is left. An odd number of players in a round will result in one player getting a bye.

A name for the tournament to describe what it is, e.g. \Taikyoku Shogi Endurance Championships".

Any notes about the tournament that you would like prospective participants or onlookers to see (optional).

Selecting \OK" to create the tournament, with yourself as the host. If you want to participate in your own tournament { which is usually frowned upon! { press the \Join" button to join it. *Note:* a single user cannot host more than 20 tournaments at a time.

2.4.4 Running a Tournament

This section details the complete process for hosting, joining and managing a tournament using Ludii.

Preparing for Tournaments

The tournament Host (i.e. the person running the tournament) and each participant should download and run a Ludii client on their machine [https://ludii.games/download].

The tournament Host and each participant should register for an account on the Ludii Forum, if they do not already have one [https://ludii.games/forums].

The tournament Host and each participant should open the Remote Play Dialog in their Ludii app and log in with their Ludii Forum username and password.

Hosting Tournaments

The tournament Host should create their tournament, using the Create Tournament Dialog as described in Section 2.4.3 above.

If the tournament is private, the tournament Host should contact the desired players and invite them to join the tournament. If the tournament is public, the tournament Host should announce the tournament on a suitable forum and wait for the desired number of players to join. The Host can join their own tournament if they want to.

Once all players have joined the tournament, the host can start the tournament by pressing the \Start" button on the Remote Tournaments Tab Figure 2.10a.

Joining Tournaments

Participants should wait for the Host to create the tournament.

Once the Host has created the tournament, participants can join it from the Remote Tournaments Tab Figure 2.10a.

Once the Host has started the tournament, participants should find their games for the first round in the Remote Games Tab Figure 2.9a, under the \Games You Are Playing" table (unless the participant has been a bye for this round. A message will be sent to each participant that the tournament has started and what games they are playing.

Participants can join and play any game in any order they like. Participants can also leave a game and come back to it later.

Participants can see their active games in the \Games You Are Playing" table in the Remote Games Tab Figure 2.9a. Games in which it is the your turn to move will be highlighted in light red; if you see your time counter ticking down for any games, then you should double click on those games to go to them at once!

Once the current round for the tournament has finished, either the next round of games will be created or the tournament will end, depending on its format.

Observing Tournaments

The current state of each tournament can be viewed at https://ludii.games/browseTournaments.

Complete trials for each game in the tournament can also be downloaded.

A complete report on each tournament is available for download after it has finished.

Managing

The tournament host can manage their tournament by selecting it at https://ludii.games/browseTournaments.

The tournament host must make sure they are logged into their Ludii Forum account. If not, a link to sign in will be displayed at the top right corner of the page.

The tournament host can update certain tournament values by selecting the "Host Control Settings" link at the bottom of the page. Be aware that this will allow the tournament host to directly alter values in the database. Tournament hosts should be fully aware of what they are doing before using these control settings. Additional details and more user-friendly controls will be provided on this page in the future.

3
Modelling New Games

Game descriptions for Ludii are written in text files with a `.lud` extension. The language used to describe games for Ludii is defined by a *class grammar* (Browne, 2016) approach; it is automatically derived from the *ludeme* classes available in Ludii's `.jar` file. The full grammar { with full documentation { is available in the Ludii Language Reference, which may be downloaded from https://ludii.games/downloads/LudiiLanguageReference.pdf.

3.1 Modelling Games as Trees of Ludemes

The basic premise of the language is that ludemes are described as their name, followed by a whitespace-separated list of arguments, all wrapped up in a pair of parentheses:

Ludeme 1

```
(ludemeName arg1 arg2 arg3 ...)
```

Some of those arguments may themselves also be ludemes, leading to a tree of ludemes. The \outer" ludeme of a full game description is always the game ludeme. The following example shows the full game description of *Tic-Tac-Toe*:

Ludeme 2

```
(game "Tic-Tac-Toe"
    (players 2)
    (equipment
        {
            (board (square 3))
            (piece "Disc" P1)
            (piece "Cross" P2)
        }
```

```
        )
        (rules
            (play (move Add (to (sites Empty))))
            (end (if (is Line 3) (result Mover Win)))
        )
    )
```

The five parameters for the game ludeme are `name`, `players`, `mode`, `equipment`, and `rules`. Only four of them are explicitly used in this *Tic-Tac-Toe* example, with the unused `mode` parameter implicitly left at its default value.

3.1.1 Name

The name of the game, simply supplied as a `"String"` (in quotes), should always be equal to the name of the `.lud` file (without that extension. It does not affect gameplay, but may be displayed in various places in the GUI.

3.1.2 Players

The `(players)` ludeme allows for the players of a game to be described. In the vast majority of games, it is sufficient to only specify the number of players that participate in a game, and leave any other player data at their defaults. For example, for two-player games such as the *Tic-Tac-Toe* example, this may be achieved using `(players 2)`.

3.1.3 Mode

The `mode` ludeme is used to define the basic control flow of a game. Alternating-move games are assumed to be the default type of games in Ludii. Therefore, alternating-move games such as *Tic-Tac-Toe* do not need to specify the `mode` argument at all. Simultaneous-move games can be specified by using this argument explicitly.

3.1.4 Equipment

The `equipment` ludeme is used to define equipment required to play a game. Broad categories of equipment are:

Components: pieces (e.g. discs and crosses in *Tic-Tac-Toe*, pawns, kings, queens, etc. in *Chess*), cards, tiles, dice, etc.

Containers: boards, hands, decks, pools, etc.

Other: important regions (e.g. opposite sides of boards to be connected to each other for a victory in *Hex*), tracks to move along (in race games), etc.

For example, the *Tic-Tac-Toe* description above defines its board – a square board of size 3, tiled with square cells – and pieces to be placed by players – a disc for player 1, and a cross for player 2 – as equipment.

3.1.5 Rules

Finally, the `rules` ludeme is used to define the rules according to which a specific game is played. These are split up in two or three sets of rules; `start` (optional), `play` (required), and `end` (required).

`start`: This can be used to define any rules to be applied whenever a new game starts. Defining `start` rules is optional; many games do not need any `start` rules. If a game has no `start` rules, any containers (such as boards, or player hands) will be completely empty when a game starts. Examples of games that require `start` rules are *Chess* or *Breakthrough*, because they require pieces to be set up on the board before gameplay can start.

`play`: This is used to define how Ludii should generate its list of legal moves for any game state. Adding `play` rules is compulsory.

`end`: This is used to define when Ludii should declare a game to be over, and what the outcome is (e.g. which player is a winner or loser, or how are the players ranked). Adding `end` rules is compulsory.

For example, the *Tic-Tac-Toe* description above does not require any `start` rules { gameplay starts with an empty board. Every turn, the current mover in *Tic-Tac-Toe* chooses one empty cell of the board to place one of its pieces. This is defined by `(play (move Add (to (sites Empty))))`. Because there is only one container (one board), and every player only owns a single type of piece, it is not required to explicitly define the type of piece to place { or which container to place them in { in the `play` rules. Whenever a player completes a line of three pieces, they win the game. This is defined by `(end (if (is Line 3) (result Mover Win)))`.

By default, Ludii declares any game in which no player is able to make a legal move a draw. It is not necessary to explicitly define this rule for *Tic-Tac-Toe* (or any other game).

3.2 Walkthrough: Implementing *Amazons* from Scratch

In this section, we provide a step-by-step guide for how a game like *Amazons* may be described for Ludii. *Amazons* is played on a 10 10 board. Each player has four amazons (chess queens), with other pieces used as arrows. Every turn consists of two moves. First, a player moves one of their amazons like a Chess queen, without crossing or entering a space occupied by another amazon or arrow. Second, it shoots an arrow to any space on the board that is along the unobstructed path of a queen's move from that place. The last player able to make a move wins.

3.2.1 Step 1: A Minimum Legal Game Description

We start out with the minimum description that results in a legal game description that may be loaded in Ludii by defining the number of players, the board by its shape and its size, the most used playing rules, and a basic ending rule.

Ludeme 3

```
1  (game "Amazons"
2      (players 2)
3      (equipment
4          {
5              (board (square 10))
6          }
7      )
8      (rules
9          (play
10             (forEach Piece)
11         )
12
13         (end
14             (if
15                 (no Moves Next)
16                 (result Mover Win)
17             )
18         )
19     )
20 )
```

Line 2 de nes that we wish to play a two-player game, where it is implied by default to be an alternating-move game. Line 3 de nes the equipment, and is used to list all the items used in the game. Line 5 de nes that we wish to use a square board of size 10. By default, the square board is tiled by squares. Line 8 is used to de ne the rules of the game; the minimum rules to compile are the playing and the ending rules. Lines 9-11 describe the playing rules by using one of the simplest `play` rules available in Ludii; `(forEach Piece)`, which simply de nes that Ludii should loop through all pieces owned by a player, and extract legal moves from the piece types to generate the list of legal moves for a mover. Finally, lines 13-18 describe the ending rules. Here we want the player who last made a move to win the game whenever the next player has no move.

3.2.2 Step 2: Defining the Pieces

In this step, we add the pieces to the equipment.

Ludeme 4

```
1      (game "Amazons"
2          (players 2)
3          (equipment
4              {
5                  (board (square 10))
6                  (piece "Queen" Each)
7                  (piece "Dot" Neutral)
8              }
9          )
10         (rules
11             (play
```

```
12                  (forEach Piece)
13              )
14
15              (end
16                  (if
17                      (no Moves Next)
18                      (result Mover Win)
19                  )
20              )
21          )
22      )
```

Line 6 de nes that each player should have a piece type labelled \Queen". Ludii will automatically label these as "Queen1" and "Queen2" for players 1 and 2, respectively. Additionally, in line 7 we de ne a \Dot" piece type, which is not owned by any player. This is the piece type that we will use in locations that players block by shooting their arrows.

3.2.3 Step 3: Defining the Starting Rules

We extend the game description listed above by adding `start` rules to place the pieces on the board:

Ludeme 5

```
1  (game "Amazons"
2      (players 2)
3      (equipment
4          {
5              (board (square 10))
6              (piece "Queen" Each)
7              (piece "Dot" Neutral)
8          }
9      )
10     (rules
11         (start
12             {
13                 (place "Queen1" {"A4" "D1" "G1" "J4"})
14                 (place "Queen2" {"A7" "D10" "G10" "J7"})
15             }
16         )
17         (play
18             (forEach Piece)
19         )
20
21         (end
22             (if
23                 (no Moves Next)
24                 (result Mover Win)
25             )
26         )
```

```
27      )
28  )
```

Lines 11{16 ensure that any game is started by placing objects of the two di erent types of queens in the correct starting locations. The labels used to specify these locations can be seen in Ludii by enabling \Show Coordinates" in Ludii's *View* menu.

3.2.4 Step 4: Adding the Final Rules for *Amazons*

To complete the game of *Amazons*, we need to allow players to move their queens and to shoot an arrow after moving a queen. This is implemented in the following game description:

Ludeme 6

```
1  (game "Amazons"
2      (players 2)
3      (equipment
4          {
5              (board (square 10))
6              (piece "Queen" Each (move Slide (then (moveAgain))))
7              (piece "Dot" Neutral)
8          }
9      )
10     (rules
11         (start
12             {
13                 (place "Queen1" {"A4" "D1" "G1" "J4"})
14                 (place "Queen2" {"A7" "D10" "G10" "J7"})
15             }
16         )
17         (play
18             (if (is Even (count Moves))
19                 (forEach Piece)
20                 (move Shoot (piece "Dot0"))
21             )
22         )
23
24         (end
25             (if
26                 (no Moves Next)
27                 (result Mover Win)
28             )
29         )
30     )
31 )
```

To make the queens able to move, inside the queen pieces, we have added the following: `(move Slide (then (moveAgain))))`. By default, the `(move Slide)` ludeme de nes that the piece is permitted to slide along any axis of the used board, as long as we keep moving through locations that are empty. No additional restrictions { in terms of direction or distance, for example {

are required for queen moves. We have appended (then (moveAgain)) in the queen moves. This means that, after any queen move, the same player gets to make another move.

In lines 18-21, the play rules have been changed to no longer exclusively extract their moves from the pieces. Only at even move counts (0, 2, 4, etc.) do we still make a queen move (using (forEach Piece). At odd move counts, the moves are de ned by (move Shoot (piece "Dot0")). This rule lets us shoot a piece of type "Dot0" into any empty position, starting from the location that we last moved to { this is the location that our last queen move ended up in. This game description implements the full game of *Amazons* for Ludii.

Once pieces are de ned, their names are internally appended with the index of the owning player. For example, the above description de nes a \Queen piece for players 1 and 2, then the subsequent description refers to \Queen1 for \Queen pieces belonging to Player 1 and \Queen2 for \Queen pieces belonging to Player 2. The \Dot piece is referred to as Dot0, indicating that this is a neutral piece not owned by any player. Note that pieces can also be referred to by their undecorated names in the game description, e.g. \Queen or \Dot, in which case the reference applies to all pieces with that name belonging to any player.

3.2.5 Step 5: Improving Graphics

The game description above plays correctly, but does not look appealing because it uses Ludii's default colours for the board. This can be easily improved by adding graphics metadata:

Ludeme 7

```
(game "Amazons"
    (players 2)
    (equipment
        {
            (board (square 10))
            (piece "Queen" Each (move Slide (then (moveAgain))))
            (piece "Dot" Neutral)
        }
    )
    (rules
        (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)
```

```
28                (result Mover Win)
29              )
30          )
31      )
32  )
33
34  (metadata
35      (graphics
36          {
37              (piece Scale "Dot" 0.333)
38              (board Style Chess)
39          }
40      )
41  )
```

Line 37 makes the \Dot" pieces smaller, and line 38 applies a Chess style to the board.

3.3 Metadata

Following the de nition of a game ludeme in a `.lud` le, additional metadata may be provided using a similar, *ludeme*-like format. The language for metadata is also described in detail in the Ludii Language Reference. The above walkthrough, for *Amazons*, already ended with a short example of metadata for graphics. Apart from modifying graphics, metadata can also be used to provide additional information about a game, and to provide various hints for that game to Ludii's AI players. The full metadata included in Ludii's `Amazons.lud` le is as follows:

Ludeme 8

```
(metadata

    (info
        {
            (description "Invented in 1988 by Walter Zamkauskas and
                first published in the Argentine magzine El
               Acertijo in December 1992.")
            (rules "Played on a 10x10 board. Each player has four
               Amazons (chess queens), with other pieces used as
               arrows. Two things happen on a turn: an amazon moves
                like a Chess queen, but cannot cross or enter a
               space occupied by another amazon or arrow. Then, it
               shoots an arrow to any space on the board that is
               along the path of a queen's move from that place.
               The last player able to make a move wins.")
            (source "https://en.wikipedia.org/wiki/
               Game_of_the_Amazons")
            (version "1.0.0")
            (classification "board/space/blocking")
            (author "Walter Zamkauskas")
            (credit "Eric Piette")
```

```
            (origin  "This game is from Argentina, on 1988.")
        }
    )

    (graphics {
        (piece Scale "Dot" 0.333)
        (board Style Chess)
    })

    (ai
        "Amazons_ai"
    )

)
```

The AI metadata can often be quite verbose and long. Therefore, this is stored in a separately le, which is referenced by the "Amazons_ai" string in (ai "Amazons_ai").

3.4 Creating Your Own Games

A full reference of the language used by Ludii to describe games is available at https://ludii.games/downloads/LudiiLanguageReference.pdf. However, this document is quite large and can be di cult to immediately use when getting started with writing new games for Ludii. For getting started, we recommend to extract all the .lud for all the built-in games from the Ludii.jar archive. These can all serve readily as examples, and be tweaked to develop new (but similar) games.

4
Implementing Custom Agents

Ludii's `.jar` file provides an abstract `AI` class which may be extended by third-party users to develop their own AI players. We first provide instructions for how to develop your own AI for Ludii, and describe how to load them into the Ludii application afterwards. Finally, we explain how to run Ludii games programmatically (circumventing its graphical user interface), which we expect to be particularly interesting for AI developers and AI researchers. Many of these instructions can also be found on https://github.com/Ludeme/LudiiExampleAI, which hosts a repository with some example AIs for Ludii.

4.1 Implementing a Ludii AI Player

Ludii's `.jar` file cannot only be used to run the application, but also as a library for other Java projects. Any Java project that uses it as a library can implement new AI players for Ludii by creating a class that extends Ludii's `util.AI` abstract class.

The most important abstract method to be overridden from this class is `selectAction`, for which the signature is given in the following box:

Java 1

```
public abstract Move selectAction
(
    final Game game,
    final Context context,
    final double maxSeconds,
    final int maxIterations,
    final int maxDepth
);
```

Any AI for Ludii is expected to implement this method such that it returns the move to be played in a given game state. The `game` object is an object that contains the rules of the loaded

game, and methods required for implementing a forward model (such as methods to obtain the list of legal moves in a game state, or methods to apply a move or roll out a full playout). The `context` object contains data for the current game state, the full list of moves applied previously, etc. The nal three parameters may be used to restrict an AI player's \thinking time" (in seconds), maximum number of iterations (for instance, MCTS iterations), and search depth. Ludii does not currently enforce any such restrictions, but we expect they may be useful for AI researchers to run experiments. All of Ludii's built-in AIs respect the maximum number of seconds, and all MCTS-based built-in AIs also respect restrictions on their iteration count. Of course, future competitions run using Ludii (Stephenson, Piette, Soemers, & Browne, 2019) will more strictly enforce such restrictions.

An example implementation, taken from the example \Random AI" from our example repository, is given in the following box:

Java 2

```
1  @Override
2  public Move selectAction
3  (
4      final Game game,
5      final Context context,
6      final double maxSeconds,
7      final int maxIterations,
8      final int maxDepth
9  )
10 {
11     FastArrayList<Move> legalMoves = game.moves(context).moves();
12
13     if (legalMoves.isEmpty())
14         return Game.createPassMove(context);
15
16     // If we're playing a simultaneous move game, some of the legal
17     // moves may be  for different players. Extract only the ones
18     // that we can choose.
19     if (!game.isAlternatingMoveGame())
20         legalMoves = AIUtils.extractMovesForMover(legalMoves, player);
21
22     final int r =
23         ThreadLocalRandom.current().nextInt(legalMoves.size());
24     return legalMoves.get(r);
25 }
```

In lines 13-14, we handle the special case where no legal moves are available by simply passing. In lines 19-20, we extract only those legal moves that belong to the player given by the `player` variable. This is required in simultaneous-move games, because those games may simultaneously have di erent legal moves for di erent players at any given point in time.

The `player` variable mentioned above is how the AI knows which player it is currently controlling. This value is supplied to AIs in an `initAI()` method which is called whenever an AI starts playing a new game, and can be overridden to perform any desired initialisation. For example, the example Random AI implements it as follows:

Java 3

```
@Override
public void initAI(final Game game, final int playerID)
{
    this.player = playerID;
}
```

Ludii's `AI` abstract class also has a `String` member named `\friendlyName", which may sometimes be displayed in the GUI. The example Random AI sets this property in its constructor:

Java 4

```
public RandomAI()
{
    this.friendlyName = "Example Random AI";
}
```

Finally, the class has a method that may be overridden to inform the Ludii application if a certain (type of) game cannot be played by an AI. For example, our example implementation of a simple UCT agent[6] (Browne et al., 2012) overrides this method as follows:

Java 5

```
@Override
public boolean supportsGame(final Game game)
{
    if (game.isStochasticGame())
        return false;

    if (!game.isAlternatingMoveGame())
        return false;

    return true;
}
```

This implementation ensures that, if the agent is loaded into the Ludii application, it will never try to make them play any games with stochasticity or simultaneous moves; instead, Ludii will automatically replace it with one of its own built-in AIs that does support those types of games.

4.2 Loading Third-party AI Players in Ludii

Ludii can import any custom AI controller that extends Ludii's `AI` abstract class from `.jar` les. The steps to do this are:

1. Create a `.jar` le containing the `.class` les for your AI implementation, or obtain one from a third party. **Note**: as with third-party `.lud` game description les, we recommend only using les from trusted sources!

[6] https://github.com/Ludeme/LudiiExampleAI/blob/master/src/mcts/ExampleUCT.java

2. Select \From JAR" for one or more players in the Player Preferences dialog.
3. Navigate to, and select, your `.jar` file in the file chooser that pops up.
4. A dialog will pop up with a drop-down menu listing all the AI classes that were found in the selected `.jar` file. Whichever class is selected from this menu will be used as an AI controller by Ludii.

4.3 Programmatically Running Ludii Games

Apart from loading custom AI controllers into the Ludii application, it is also possible to programmatically run Ludii games from any Java project using Ludii's `.jar` file as a library. We expect this to be particularly convenient for AI developers to test their implementations during development without having to use the GUI, and for AI researchers to run large-scale experiments.

Ludii's repository with example AIs also contains a Tutorial file with extensive comments, which showcases how some of the basic functionality of Ludii may be accessed programmatically. This file can be found at the using the following URL: https://github.com/Ludeme/LudiiExampleAI/blob/master/src/experiments/Tutorial.java. Two additional, shorter examples for running games programmatically can be found at:

https://github.com/Ludeme/LudiiExampleAI/blob/master/src/experiments/RunLudiiEvalGamesSet.java

https://github.com/Ludeme/LudiiExampleAI/blob/master/src/experiments/RunCustomMatch.java

5 Troubleshooting

5.1 Cannot Get Ludii to Run

If double-clicking Ludii's `.jar` file does not make the application run, it is possible that your operating system is not using the correct executable to run it. See https://stackoverflow.com/questions/8511063/how-to-run-jar-file-by-double-click-on-windows-7-64-bit for Windows, or search for similar solutions for other operating systems.

Alternatively, if you just installed (a new version of) Java, it may help to unplug and replug your monitor.

5.2 Poor Performance User Interface

If Ludii appears slow or unresponsive on your device then we recommend the following.

Ensure you have Java 8 or higher installed (preferably the latest version).

Turn off high-DPI scaling on your device. Depending on your operating system, this can be achieved as follows:

1. Windows: Open "Display settings" by searching for it in the Windows search bar, or by right clicking on the desktop. Under the "Scale and layout" section, change the option labelled "Change the size of text, apps, and other items" to 100%.
2. Windows (advanced): This option is more complicated, but will ensure that high-DPI scaling is only removed from Java applications. Navigate to you Java installation directory, and locate the file "javaw.exe" inside the "bin" directory. Right click on this file, select "Properties", select "Compatibility", select "Change high DPI settings" if present, activate High DPI scaling override and override the scaling behaviour performed by System.
3. macOS: Open "System preferences" and select "Displays". Under "Resolution" select "Default for display".

4. Ubuntu: Open \Settings" and select \Screen Display". Change the Scale option to 100%.

Acknowledgement

This work is part of the *Digital Ludeme Project*, funded by €2m European Research Council (ERC) Consolidator Grant #771292, being run by Cameron Browne at Maastricht University's Department of Data Science and Knowledge Engineering over 2018{23.

A
Keyboard Shortcuts

Keystroke(s)	Action
Ctrl+L	Load Game
Ctrl+F	Load Game from File
Ctrl+T	Load Trial
Ctrl+S	Save Trial
Space	Start / Pause Playing
Left	Go back one step (undo one move)
Right	Go forward one step (redo one move)
Down	Jump back to start
Up	Jump forwards to end of game (played so far)
Ctrl+R	Restart Game
Ctrl+M	Make a Random Move
Ctrl+P	Make a Random Playout
Alt+P	Open Preferences / Settings Dialog
Shift+L	List All Legal Moves (in Status tab)
Shift+O	Open Editor (Expanded)
Shift+S	Open Editor (Short)
Shift+F	Toggle Auto From Moves
Shift+T	Toggle Auto To Moves
Alt+R	Cycle Players
Shift+J	Toggle Swap Rule

Shift+N	Toggle No Repetition of Game States Rule
Shift+W	Toggle No Repetition Within a Turn Rule
Shift+I	Allow \illegal" moves (in deduction puzzles)
Shift+V	Show possible values (in deduction puzzles)
Shift+E	Open Evaluation Dialog
Ctrl+O	Time Random Playouts
Shift+R	Time Random Playouts (in a background thread)
Shift+B	Toggle Display Board
Shift+P	Toggle Display Pieces
Shift+G	Toggle Display Graph
Shift+D	Toggle Display Cell Connections
Shift+A	Toggle Display Axes
Alt+M	Toggle Display Legal Moves
Alt+L	Toggle Display Last Move
Alt+A	Toggle AI Visualisations
Alt+I	Toggle Display Cell Indices
Alt+E	Toggle Display Edge Indices
Alt+F	Toggle Display Vertex Indices
Alt+C	Toggle Display Cell Coordinates
Alt+B	Toggle Display Edge Coordinates
Alt+W	Toggle Display Vertex Coordinates
Shift+0, . . . , Shift+9	Toggle Display Track
Alt+N	Open Network Dialog
Alt+1, Alt+2, . . . , Alt+0	Load Recent Game
Alt+V	Display Built-in SVG Images
Alt+U	Load SVG image le to display
Shift+C	Take a Screenshot (saved in directory from which Ludii is run)
Shift+K	Clear Board (in sandbox mode)
Shift+Y	Next Player (in sandbox mode)
Shift+Q	Exit Sandbox Mode
Ctrl+G	Generate Grammar
Alt+G	Generate Symbols
Ctrl+H	Open About Dialog
Ctrl+Q	Quit Ludii

B Technical Details Built-in AIs

This appendix provides technical details for the di erent built-in AI controllers of Ludii.

B.1 Flat MC

This is a at Monte-Carlo search agent. Let $M(s)$ denote the set of legal actions for the agent in a game state s. For as many iterations as possible, it runs full playouts from the current game state s, where all moves are selected uniformly at random. At the end of every playout, the outcome is mapped to a value in $[-1, 1]$. Whichever move $m \in M(s)$ resulted in the greatest average score among all playouts in which it was selected at the start of the playout is nally selected as move to play in s.

B.2 UCT

This is a fairly standard implementation of the UCT variant of MCTS (Browne et al., 2012). Some implementation details:

- The standard UCB1 policy (Auer, Cesa-Bianchi, & Fischer, 2002) is used in the MCTS selection phase, with an exploration constant $C = \sqrt{2}$ (i.e. equivalent to the standard formulation of UCB1 without an exploration hyperparameter).
- Moves in playouts are selected uniformly at random.
- Playouts are capped at 200 moves (after the selection phase).
- The search tree is expanded by one node per iteration.
- Relevant parts of the search tree from a previous search process are reused when a new search process starts.
- In deterministic games, game states are stored in the nodes. For nondeterministic games, we use an \open-loop" approach (Perez, Dieskau, Hunermund, Mostaghim, & Lucas,

2015); game states are not stored in nodes, but re-generated (by a possibly nondeterministic forward model) as we traverse from a root node to a leaf node in every iteration.

The selection phase may select children that have already been visited before, even if there are also children without any visits available. The \mean value" estimate for children without any visits is equal to the value estimate of their parent node.

The nal move selected by the algorithm is that corresponding to the child of the root node with the highest visit count (the \robust child").

The implementation maps all game outcomes to lie in [1,1] for its value estimates.

No transposition table is used.

This implementation does not support simultaneous-move games.

B.3 UCT (Uncapped)

Identical to the agent described above, except playout durations are uncapped (except for any caps in the game itself).

B.4 MC-GRAVE

An implementation of Generalized Rapid Action Value Estimation (GRAVE) (Cazenave, 2015). Most implementation details are identical to those of UCT, as described above. Di erences are:

The tree is traversed using GRAVE's strategy in the selection phase of MCTS. We use ref = 100 (this is the threshold determining which node's AMAF values are used), and $bias$ = 10^{-6} (this is the hyperparameter used in the denominator when computing the β_m weight for GRAVE). These hyperparameter values are loosely based on the experiments reported by Cazenave (2015).

The selection phase does not incorporate any explicit exploration terms. In the future we may add a UCT-GRAVE variant with an explicit exploration term (this is why we named the current implementation MC-GRAVE instead of GRAVE).

The value estimate for unvisited children is set to 10,000 (deliberately far outside the normal range of value estimates, which is upper bounded by 1.0), instead of setting it to be equal to the value estimate of the parent node. This is required due to the lack of an explicit exploration term in MC-GRAVE.

AMAF statistics are updated once *per occurrence* of a move in a playout. This means that, if the same move (or one with the same hash code) occurred more than once in a single playout, the AMAF statistics for that move are updated more than once for a single playout.

B.5 Biased MCTS

A variant of MCTS that is biased by *features* (local patterns for state-action pairs). Most implementation details are identical to those of UCT, as described above. Di erences are:

The tree is traversed using the same PUCT strategy as AlphaGo Zero (Silver et al., 2017) in the selection phase of MCTS. We use an exploration constant of 2.5.

The policy used to bias PUCT is a softmax function of logits { one logit per state action pair, which is computed as the dot product between a vector of weights and a vector of binary features for the state-action pair (Browne, Soemers, & Piette, 2019; Soemers, Piette, & Browne, 2019).

The same policy is also used to select moves in the playout phase of MCTS.

B.6 MCTS (Biased Selection)

This algorithm is similar to the Biased MCTS described above, but uses unbiased (uniform) playouts as used by UCT.

B.7 Alpha-Beta

In two-player games, this agent uses the well-known minimax algorithm with alpha-beta pruning. Speci c implementation details are:

Iterative deepening is used to obtain \anytime" search behaviour under time restrictions.

Before starting the very rst iteration of iterative deepening (with a depth limit of 1), the legal moves at the root are shu ed { this leads to random tie-breaking between moves with equal evaluations.

Before starting a new search with a new depth limit in iterative deepening, the agent orders the moves at the root node based on the evaluations from the previous search with the previous depth limit.

If the agent proves a win, it will immediately return the move leading to that win without starting subsequent iterations of iterative deepening.

If the agent proves a loss, it will immediately return the move that was best according to the previous search with the previous search depth.

Heuristic terms with an absolute weight below 0.01 are ignored in the heuristic evaluation function, for improved computational e ciency.

In games with more than two players, the following additional implementation details apply:

In general, the agent runs a paranoid search, where all opponents are assumed to \collaborate" against the searching agent. This allows for alpha-beta pruning to be used.

If the agent proves a win despite the paranoid assumption, it plays accordingly.

If the agent proves a loss under the paranoid assumption, and still has search time left, it will re-start a Max^N search { without the paranoid assumption, and without alpha-beta pruning. This may perform better than the paranoid search in end-game situations or games with trivially small search spaces.

References

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, *47*(2{3), 235{256.

Browne, C. (2016). A Class Grammar for General Games. *Computers and Games*, *LNCS 10068*, 167{182.

Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P. I., Rohlfshagen, P., ... Colton, S. (2012). A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, *4*(1), 1{49.

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

Cazenave, T. (2015). Generalized Rapid Action Value Estimation. In Q. Yang & M. Woolridge (Eds.), *Proceedings of the Twenty-Fourth International Joint Conference on Arti cial Intelligence (IJCAI 2015)* (pp. 754{760). AAAI Press.

Perez, D., Dieskau, J., Hünermund, M., Mostaghim, S., & Lucas, S. M. (2015). Open Loop Search for General Video Game Playing. In *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 337{344).

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... Hassabis, D. (2017). Mastering the game of Go without human knowledge. *Nature*, *550*, 354{359.

Soemers, D. J. N. J., Piette, E., & Browne, C. (2019). Biasing MCTS with features for general games. In *Proc. 2019 ieee congr. evol. computation* (pp. 442{449). IEEE.

Stephenson, M., Piette, E., Soemers, D. J. N. J., & Browne, C. (2019). Ludii as a Competition Platform. In *Proceedings of the IEEE Conference on Games (COG 2019).*