Digital
Ludeme
Project

The Ludii System, Database and Portal: Functional Requirements

Version 1.0

Cameron Browne Eric Piette

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

15 January 2019

European Research Council
Established by the European Commission

Abstract

This document provides an overview of the functionality required by the software developed as part of the Digial Ludeme Project in order to achieve the project's aims. Key components will include the Ludii System (a complete general game system), the Ludii Server (which stores private databases required for the project) and the Ludii Portal (a suite of public-facing web pages that provide external users access to public data and services provided by the System).

Please contact the authors with any comments or corrections at:
{cameron.browne, eric.piette}@maastrichtuniversity.nl

Contents

1
Overview

The Ludii general game system is a software system for playing, evaluating, comparing, designing and optimising a wide range of games. Ludii is being developed as part of the ERC-funded Digital Ludeme Project (DLP) and will be used to model the full range of traditional strategy games throughout recorded human history. This document describes the functional requirements of the Ludii system and its basic architecture. The key components are:

1. Ludii **System** *(public/private)*: The Ludii System is a software system for describing, compiling, playing, evaluating and modifying games in *ludemic* form (described shortly. It consists of a Ludeme Library containing hundreds of Java classes each describing a *ludeme*, and associated programs for compiling and using these. Some of this functionality will be made publicly available via the Ludii Portal while some will remain private to DLP team members.
2. Ludii **Server** *(private)*: The Ludii Server consists of a set of databases hosted on a dedicated machine at Maastricht University's (UM) Internet, Computing and Technical Services (ICTS) centre. This machine and its contents are strictly private and can only be accessed by authorised DLP team members from within the UM network.
3. Ludii **Portal** *(public-facing)*: The Ludii Portal is a suite of public-facing web pages through which external users can access the system, its public content, and request various services. These web pages may be hosted on various machines, including machines owned by UM, the DLP and/or team members.

Figure 1.1 shows the basic arrangement of these components and how they interact. External users request data and services through the Ludii Portal, which in turn accesses the Ludii database to satisfy these requests. The Server will contain potentially sensitive user data and at no point will any outside entity have direct access to the Server.

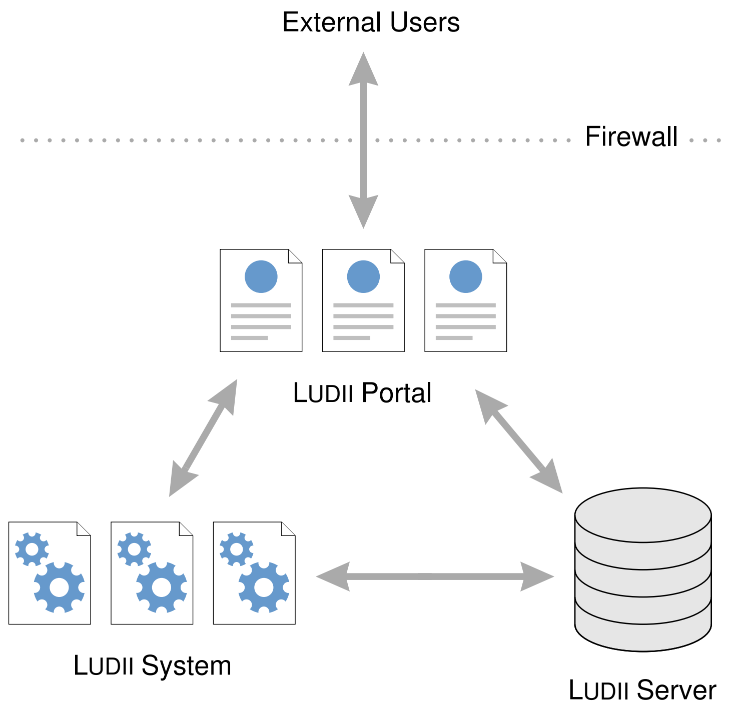

Figure 1.1: The key Ludii components and data flow between them.

1.1 Project

The DLP deals with *traditional games of strategy*, i.e. games with no proprietary designer [1] or publisher [2, p.5] that exist in the public domain and in which players succeed through mental rather than physical acumen. This category includes most board games, some card games, some dice games, some tile games, etc., and may involve non-deterministic elements of chance or hidden information as long as strategic play is rewarded over random play.

This project aims to:

1. *Model*: the world's traditional strategy games in a single, playable database.
2. *Reconstruct*: rule sets for ancient games with improved reliability.
3. *Map*: the spread of games and associated mathematical ideas throughout history.

Further details are provided in [3].

1.1.1 Scale

This project will involve modelling 1,000 representative examples of the world's traditional strategy games. The total number of rule sets to be modelled and evaluated could be in the order of 1 million, based on 1,000 core games, their variants, and alternative rule sets generated during the reconstruction process.

It is hoped that the project will attract an active community of hundreds or even thousands of Ludii users, many of whom might want to simultaneously access the data and services provided by the Ludii Portal.

1.1.2 Ludemes

Games are modelled as structures of *ludemes*, i.e. game memes or conceptual units of game-related information [4]. These constitute a game's underlying building blocks, and are the high-level conceptual terms that human designers use to understand and describe games.

For example, Tic-Tac-Toe might be described in *ludemic form* as follows:

Ludeme 1

```
(game "Tic-Tac-Toe"
   (play {(player "P1") (player "P2")} Alternating)
   (equipment (board "Board" (square 3)))
   (rules
      (moves (to (indexOf Mover) (empty)))
      (end (line Mover Any 3) (result Mover Win))
   )
)
```

An important benefit of the ludemic approach is that it encapsulates key game concepts which can be meaningful labelled. Each ludeme corresponds to a Java class in the associated Ludeme Library, and is described by its associated class name.

1.2 This Document

The Ludii System will provide the functionality to realise the DLP's aims. The Ludii Server will house the required data behind a secure firewall. The Ludii Portal will provide access to the public data and services provided by the Ludii system to external users. The following chapters outline the requirements for the Ludii System, Server and Portal.

2 Ludii System

The Ludii System is a software system for describing, compiling, playing, evaluating and modifying games in ludemic form. Any reference to the term “Ludii” is assumed to the refer to the Ludii System unless stated otherwise.

2.1 Data

The core data required for the Ludii System are the collection of ludemes used to define the equipment and rules of each game, and the grammar used to describe games in ludemic form.

2.1.1 Ludeme Library

The Ludeme Library is a collection of Java classes, each describing a ludeme, in a structured class hierarchy. The library starts with a root `game` package, with further code structured logically beneath this to define:

- *Control*: Game type, players, turn order, etc.
- *Equipment*: Containers (boards, hands, decks, piles, etc.) and components (pieces, cards, tiles, dice, etc.). These implement the `Drawable` interface to provide native graphics.
- *Rules*: Start, play, post-conditions and end rules.
- *Support*: Lower level support code (game state, action definitions, etc.) required for implementation but not wanted in the grammar.

Ludeme names need not be unique within the library, provided that they can be disambiguated by context (by package or argument list).

It is expected that the library will contain in the order of 1,000 ludemes by the end of the project. These will provide the required functionality for the full range of traditional strategy games, and will also support many modern games.

The Ludeme Library will not initially be made publicly visible. In no event will external users be allowed to modify or add to the Ludeme Library, for reasons of security and to ensure the correctness and integrity of the code base. Ludeme classes must be added carefully according to strict guidelines to ensure their compatibility with the grammar generation process and the various services from other parts of the project that must interact with the Ludeme Library to perform their functions. See the LUDII *Programming Guide* [6] for details.

2.1.2 Class Grammar

A grammar for describing games in ludemic form is generated automatically from the class hierarchy of the Ludeme Library using a *class grammar* approach [5].

The class grammar is set of *production rules* 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. It is intrinsically bound to the underlying code library, but is a *context-free grammar* that is self-contained and can be used without knowledge of the underlying code.

The basic syntax of the class grammar is as follows:

Grammar 1

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

where:

`<class>`	denotes a LHS symbol that maps to a class in the code library (i.e. ludeme).
`(class [{<arg>}])`	denotes a `class` constructor and its arguments.
`<subClass>`	denotes a subclass derived from `class`.
`terminal`	denotes a terminal symbol (fundamental data type or `enum`).
`[...]`	denotes an optional item.
`{...}`	denotes a collection of one or more items.
|	denotes a choice between options in the RHS sequence.

The resulting grammar is a summary of the ludeme class hierarchy, based on constructors and parameters, that offers full functionality while hiding the implementation details. The programming language (Java) effectively becomes the game description language; it should theoretically be possible to add almost any functionality that can be defined in Java.

Further details are provided in the LUDII *Programming Guide* [6].

2.2 Services

The LUDII System must provide the following core services. The System provides a *graphical user interface* (GUI) by default, but also also offers a *command line interface* (CLI) option that provides full functionality with the GUI. The CLI option is triggered by passing one or more arguments to the app's `main()` method.

2.2.1 Compilation

The System must be able to load and read games described in ludemic form and compile them to executable Java byte code. Games are described as structured ludeme trees, i.e. symbolic expressions compatible with the Ludii class grammar (described above), in plain text files with the extension ".lud".

The System *instantiates* games by decomposing descriptions into their component ludemes, locating the relevant classes using Java reflection, loading them and compiling to Java byte code.

2.2.1.1 API

Compiled games must provide the functionality defined by the following API:

Java 1

```
public void create();
public void start(Context context);
public List<Action> actions(Context context);
public void apply(Context context, Action action);
public Status playout(Context context, List<AI> ais);
public Controller controller(int resolution);
public View view(int resolution);
```

Where:

- Each `Context` object contains a reference to the relevant `Game` object and its static final data members (equipment, players, rules, etc.) and a reference to the current `Trial`.
- Each `Trial` object is a record of a complete game played from start to end, including the moves made and hash values of intermediate states if needed.
- Each `Action` object describes one or more atomic actions to be applied to the game state to effect a move. Actions typically include removing components from containers, adding components to containers, changing component counts or states within containers, deciding which player moves next, etc.
- Each `AI` object describes the AI implementation chosen for each player, including computational budget/time limits, hints such as features for biasing playouts, etc.
- The `Controller` object provides the mechanism for updating the game state based on user input such as mouse clicks.
- The `View` object provides the mechanism for showing the current game state on the screen.

The `API` class is a support class and not a ludeme. Note that **additional domain types beyond games may be added to the system in future.** The user defines games in the Ludii class grammar but executes them through the API. This API decouples the grammar from its implementation from the user's perspective.

2.2.2 Play

The Ludii System must allow human and AI players to play the compiled games. Figure 2.1 shows how the Ludeme Library might be used by a simple `Player` app for playing Ludii games using a `Referee` object to coordinate play via the API.

Default AI agents provided with the System will be based on *Monte Carlo tree search* (MCTS) [7] approaches, with playouts biased by learnt domain-specific features.

Games may involve from one to eight players.

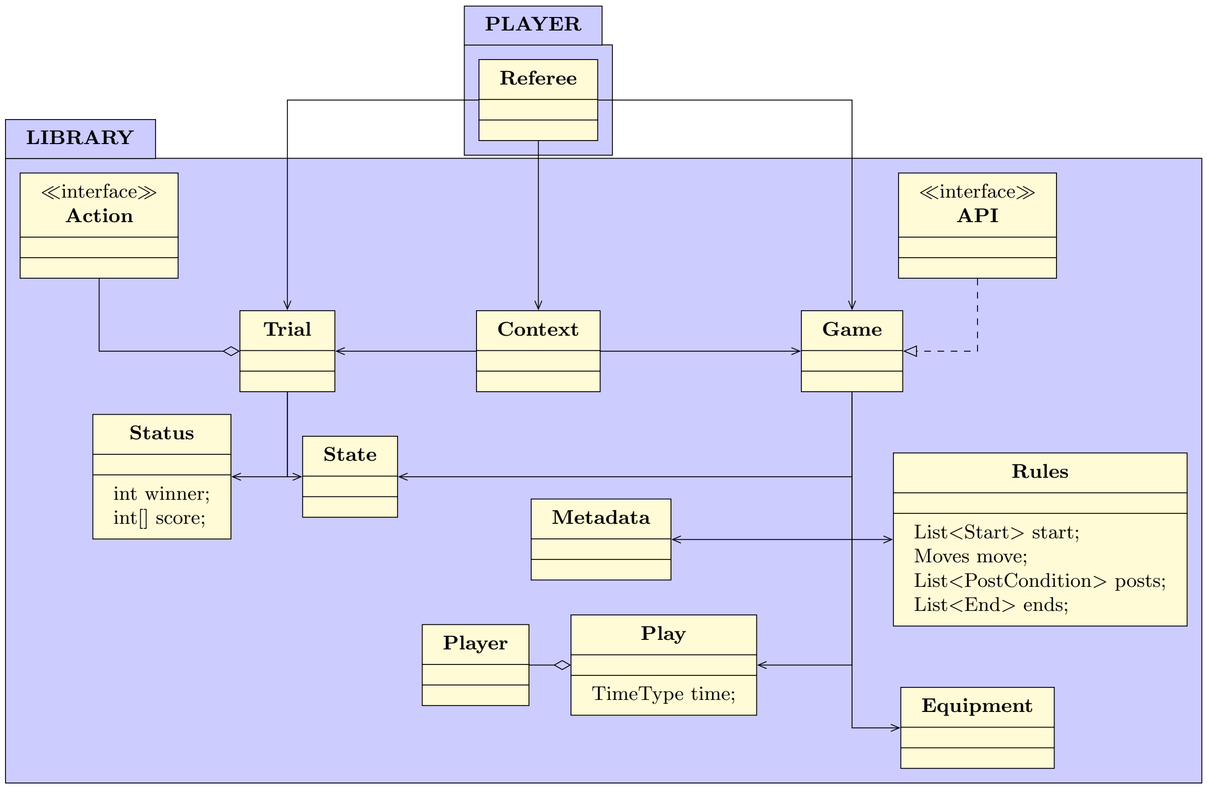

Figure 2.1: Design of a simple `Player` app for playing games via the API.

2.2.2.1 Remote Client

The System should support play between any number of human users and any number of AI agents. The standard mode of operation will be for remote users to download a public version of the Ludii app and run it remotely on their machines. Games can then be played locally on their machines through the GUI or CLI.

The default play mode should be a single (human) user against any number of default AI opponents. However, the Ludii app should allow other users to connect remotely, so that games can be played remotely between users.

The Ludii app should allow relevant user data that might be useful for the project (game choices, play logs, move timings, player preferences, etc.) to be automatically sent to the Ludii Portal, if users agree.

2.2.2.2 Local Server

The System should provide a local centralised server to coordinate play between remote clients. These may be used to create a central repository of games played to maintain statistics about each game, to run user tournaments, and to coordinate AI tournaments.

Game statistics could feed into the project's analyses related to the evaluation of games and the self-regulation of default AI playing strength. Player ratings, such as Elo or AGON ratings, should be maintained for each player for each game.

It may be desirable to provide a public playing area for each game, such as BoardSpace[1], provided that clients access this through remote app that they download to their machines, to avoid heavy work on the local server. Typical functionality includes:

- Creating an account/logging on.
- Browsing available available game descriptions.
- Joining "rooms" for each game or category of games where users can chat or arrange games.
- Playing games with other remote players.

It is envisaged that each remote user will create an account on the local server in order to use it. Users must be at least 18 years of age, or have permission from a responsible guardian, in order to register. User data will be anonymised as much as possible, but may contain potentially sensitive information volunteered by users (age, sex, cultural background, etc.).

This local server should be run on a local machine but probably not the actual Ludii Server (see Chapter 3) to avoid slowdown in case of heavy usage.

2.2.2.3 Tournaments

Tournaments will be an important service provided by the System. These may include:

1. *User Tournaments*: User tournaments are a good way to encourage usage and get users playing new games they might otherwise not play. In-kind cashless prizes – such as access to bonus game definitions – might be offered as incentives. It may be interesting to run:

    - *Decathlons*: Players compete in 10 games.
    - *Centathlons*: Players compete in 100 games.

[1] http://boardspace.net/english/index.shtml

    - *Killathlons*: Players compete in 1,000 games.

2. *AI Tournaments*: The System should coordinate AI tournaments for researchers to compare different AI techniques on any given game. AI agents may be given player ratings, such as Elo or AGON ratings, as per human users. Such AI tournaments may include:

    - *General*: Annual competitions equivalent to the AAAI General Game Playing (GGP) competition, in which researchers submit general AI agents that compete with others over a range of games that are not specified until the day of each event.
    - *Specialised*: Annual competitions equivalent to the ICGA Computer Olympiad, in which researchers submit specialised AI agents that compete with others over a range of pre-known games.
    - *Ongoing*: Regular (monthly?) competitions for a range of popular games, similar to those run by the Computer Go Server (CGOS) for Go.

2.2.2.4 AI Playing Strength

The System will aim to provide default AI agents that play each game at a competent but not overly strong level. This is called *plausible AI* [3] as the idea is to make moves that the average-to-strong human player would plausibly make.

The reason for this is to capture the "human" experience of each game as much as possible; overly weak AI agents will play poor games that do not reflect actual gameplay between (intelligent) human opponents, while similarly overly strong AI agents can play according to unknown superhuman strategies that similarly do not reflect actual gameplay between human opponents.

One way to achieve might be to self-regulate the playing strength of the default AI agents provided with the System, to achieve an average win rate of 50% against the top 50% of users. Overachieving AIs might be "dumbed down" by forgetting some features, while underachieving AIs might be "smartened up" by additional feature learning.

2.2.2.5 Feature Learning

The System should automatically learn relevant features for each to bias MCTS playouts. These may include beneficial (positively weighted) features indicating high priority moves to make, and detrimental (negatively weighted) features indicating low priority moves to avoid.

Features are based on local geometric piece patterns. We will investigate the hypothesis that learnt features encode simple local strategies relevant to the game.

Initial feature sets may be generated directly from the ludeme descriptions of each game, then optimised through self-play. Further, the feature format is deliberately kept simple and general to facilitate transfer of learnt features between similar games, to accelerate learning.

It is assumed that AIs may regulate playing strength by turning learnt features on or off.

The feature format, implementation and generation process is described elsewhere [8, 9].

2.2.3 Evaluation

The automated evaluation of rule sets is central to the project. Games are evaluated based on a combination of game quality and historical authenticity.

2.2.3.1 Game Quality

Game quality will be measured using self-play trials between AI agents trained sufficiently to play each game at a competent human level. A first level of quality testing will check for obvious flaws in rule sets, containing:

- *Bias*: Whether the rule set favours one player or colour.
- *Drawishness*: Whether too many games end in draws (threshold to be decided).
- *Length*: Whether games are too short or too long (relative to state space complexity).

More subtle quality metrics such as depth, clarity, drama, decisiveness, etc. can be difficult to measure [10] and are highly variable anyway, being subject to cultural, chronological and personal preferences. Instead, we will look for indicators of strategic potential based on the number and complexity of high-valued features learnt for each game, on the assumption that features encode basic strategies.

Due to the large number of rule sets to be evaluated over the course of the project – in the order of 1 million – quality testing must be performed quickly, ideally within a few minutes per game. This may be achieved by keeping AI thinking time short (1-2s per move) and running self-play trials in parallel.

Basing quality tests on random playouts would be much faster, but would be highly inaccurate as random playouts do not capture the essence of a game. There is no obvious way to perform quality testing without intelligent AI playouts, hence there is a lower limit on the quality measurement process of the time it takes to play out a complete game with MCTS move planning.

Automated quality measurement through self-play testing could be a valuable service to both game companies and hobbyist designers.

2.2.3.2 Historical Authenticity

Historical authenticity refers to the likelihood of a reconstructed rule set existing at a given time, place and cultural context. For example, if a rule set contains rules not found in any other known game from a similar time, place or within its culture, then that rule set is unlikely to be historically authentic.

Historical authenticity may be used to direct the reconstruction process, by biasing the search towards rules known to have been used in similar a time, place and cultural context to the target game being reconstructed, theoretically encouraging an immediate focus on more plausible reconstructions.

Overall game quality will therefore be a linear combination of:

1. Absence of flaws.
2. Strategic potential.
3. Historical authenticity.

2.2.4 Generation

Automated game generation will be a potentially powerful service provided by the Ludii System. However, while it is easy to generate thousands or even millions of new games quickly, evaluating them for quality is a much harder and more time consuming task [10]. It is therefore recommended to keep a tight rein on most generation services initially.

Modes of generation include:

- *Forensic Reconstruction*: Given some partial knowledge about a game's equipment and its historical context, e.g. a board and pieces from an archaeological dig, find plausible rule sets that maximise game quality and historical authenticity.
- *Historical Optimisation*: Given a rule set and its historical context, find similar variant rule sets that maximise game quality and historical authenticity.
- *General Optimisation*: Given a rule set, find similar rule sets that remove flaws or deepen strategic potential, or address some other degenerate cases or strategies.
- *Targetted*: Generate rule sets constrained to some subset of the search space, e.g. number of players, family of game, preferred mechanisms, etc., that produce high quality games.
- *Bespoke*: Generate high quality games with bespoke rule sets tailored to individual users, based upon player preferences deduced from their user data.
- *Unconstrained*: Explore the game design space for new high quality ludeme combinations.

We are currently implementing the LUDII Player app for loading, compiling and playing games described in the LUDII class grammar. It would make sense to implement separate Evaluator and Generator apps for evaluating games and generating games, respectively, to maintain separation between these distinct services.

3
Ludii Server

The Ludii Server is a dedicated high-end machine housed Maastricht University (UM)'s Information and Communication Technical Services (ICTS) centre. The Server contains a number of databases relevant to the DLP including potentially sensitive information, so is strictly private and can only be accessed by authorised individuals – typically DLP team members – from within the UM network.

The Server belongs to the DLP and can be accessed as `ludii.unimaas.nl`. It will contain the following databases.

3.1 Game Database

A core resource of the DLP is the playable game database. Each entry consists of a game description in ludemic form, stored as a single plain text file with extension ".lud". Each game description may contain:

- Any amount of embedded plain text metadata in a "metadata" field.
- Instructions for generating a range of related variants, e.g. different board sizes, target scores, number of pieces, etc. (exact format to be decided).
- Genealogy details including known source games.
- Quality and authenticity scores (for quick lookup).
- Possibly a list of relevant features for biasing MCTS playouts and their respective weights.

3.1.1 Ludemeplexes

A structured set of ludemes is called a *ludemeplax.* For example, the following *atomic* ludemes:

Ludeme 2
```
(tiling square)
```

Ludeme 3
```
(shape square)
```

Ludeme 4
```
(size 3)
```

may be built into a named higher-level ludemeplex as follows:

Ludeme 5
```
(board "Square 3x3"
   (tiling square)
   (shape square)
   (size 3)
)
```

Ludemeplexes are useful for describing commonly used equipment combinations and rule mechanisms. The grammar will support import of ludemeplexes by name, to simplify game descriptions. Ludemeplexes may be imported and component ludemes overridden in game descriptions, for maximum flexibility.

Ludemes and ludemeplexes will also be cross-referenced with other ludemes and ludemeplexes to find combinations that work well together and those which don't. These relationships may be used to direct the search towards promising combinations when generating rule sets.

3.1.2 Game Properties

Additional properties to be stored with each game description include:

- Whether the game is public domain. If not, then it can not be included in public versions of Ludii without the owner's permission.
- Classification by family, type, mechanism, period, culture, country, geographical location, etc.

3.1.3 Game Records

It would be useful to store for each game complete play records including game logs from all user and AI games. These may be used for fine-tuning AI playing strength, etc. This information may be anonymised and independent of the user data that creates it.

3.1.4 User Ratings

User ratings for each game. This may involve an explicit scoring system (e.g. out of 10) or be deduced from game logs e.g. by number of times the game is played, number of players who play it, time spent playing it, etc.

3.1.5 Executable Bytecode

It may be desirable to store a compiled version of each game in executable bytecode for quick access. To be decided.

3.2 Historical/Cultural Profile

In order to achieve the cultural mapping objective of the DLP, it is necessary to maintain an amount of historical and cultural information for each game. The exact format of this data is yet to be decided in consultation with the project Advisory Panel and "cultural" postdoc (currently being hired).

Likely information for each game will include:

- Where it was played.
- When it was played.
- Who played it: age, sex, class, ethnicity, etc. (where this information is available).
- Why it was played: social, cultural or religious context beyond the game itself.
- How it was played: rituals or rites associated with playing the game.
- Known precedents and antecedents.

This information will provide a *cultural profile* for each game. The components ludemes within each game will share the combined cultural profiles of all games in which they are used, by association.

Each entry of historical and cultural information should be accompanied by:

1. *Source*: Complete details of the source from which the information was obtained.
2. *Permissions*: Any necessary permissions to use the information.
3. *Evidence*: Any digital evidence, such as documents, scans, photos, videos, sound recordings, etc.
4. *Confidence*: Percentage estimate of how reliable the information is, where:
    - 100% = Certain.
    - 50% = Probable.
    - 0% = Pure conjecture.
    - $< 0\%$ = Probably false(?)

3.2.1 Shapefile Format

Game occurrences may be described in the Shapefile format [11]. This is a standard data format for used for *geographic information system* (GIS) software, that spatially defines vector features using points, lines and polygons. For example, Figure 3.1 shows geographical map in Shapefile format consisting of wells (points), rivers (polylines) and a lake (polygon). Each feature may have a name and other attributes associated with it.

[1]

[1]Source: https://en.wikipedia.org/wiki/Shapefile

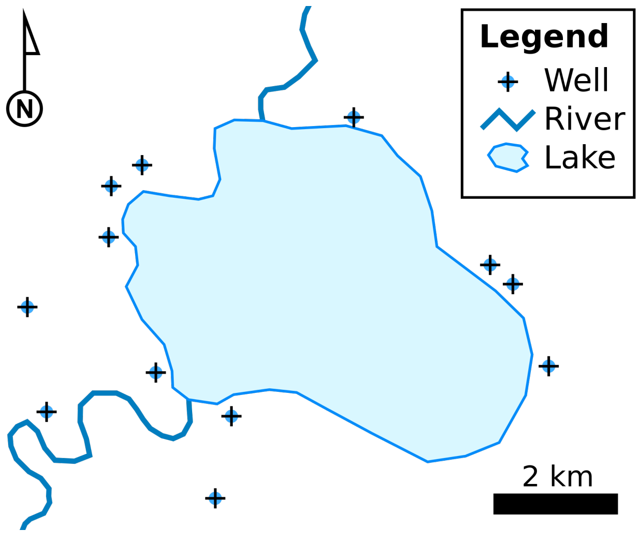

Figure 3.1: A simple vector map with wells (points), rivers (polylines) and a lake (polygon).

Game and ludeme occurrences may be described in Shapefile format using points to indicate probable occurrences (with associated dates), polylines to indicate probable dispersal routes, and polygons to indicate regions of probable usage (with associated date ranges). Each feature may have a weight associated with it to indicate the confidence of its accuracy.

The DLP will most likely engage the services of third party software provider GeaCron. GeaCron maintains a database of geo-political world maps for every year from 3000BC to the present day, which can be queried to specify which empire, nation, civilisation or culture was dominant at any given geographical location in recorded history. GeaCron also provides details of known trade routes, expeditions, and other key historical events. This provides a mechanism for situating known occurrences of games and ludemes within a cultural context.

The exact data format used by GeaCron is not known, but they describe their data as being in vector form based on points, lines and polygons, so if not the Shapefile format then is presumably sufficiently close to allow porting to Shapefile. For example, Figure 3.2 shows GeaCrons record of the Viking route from Denmark to Paris in 845AD, superimposed on a current political map of the time showing key regions, towns and landmarks.

3.3 Mathematical Profiles

Each ludeme and ludemeplex will be tagged with keywords indicated the basic mathematical principles that it embodies. A *mathematical profile* can then be automatically derived for each game based on its component ludemes.

This information will be stored separately in the LUDII Server rather than as part of the Ludeme Library, to avoid cluttering the code base with large amounts of peripheral information. Individual ludemes will be identified by package and class name.

The exact taxonomy of mathematical terms to be applied is yet to be decided, and is the subject of current work [12]. But it will probably be a combination of the Mathematical Association of America (MAA)'s *Core Subject Taxonomy for Mathematical Sciences Education* and

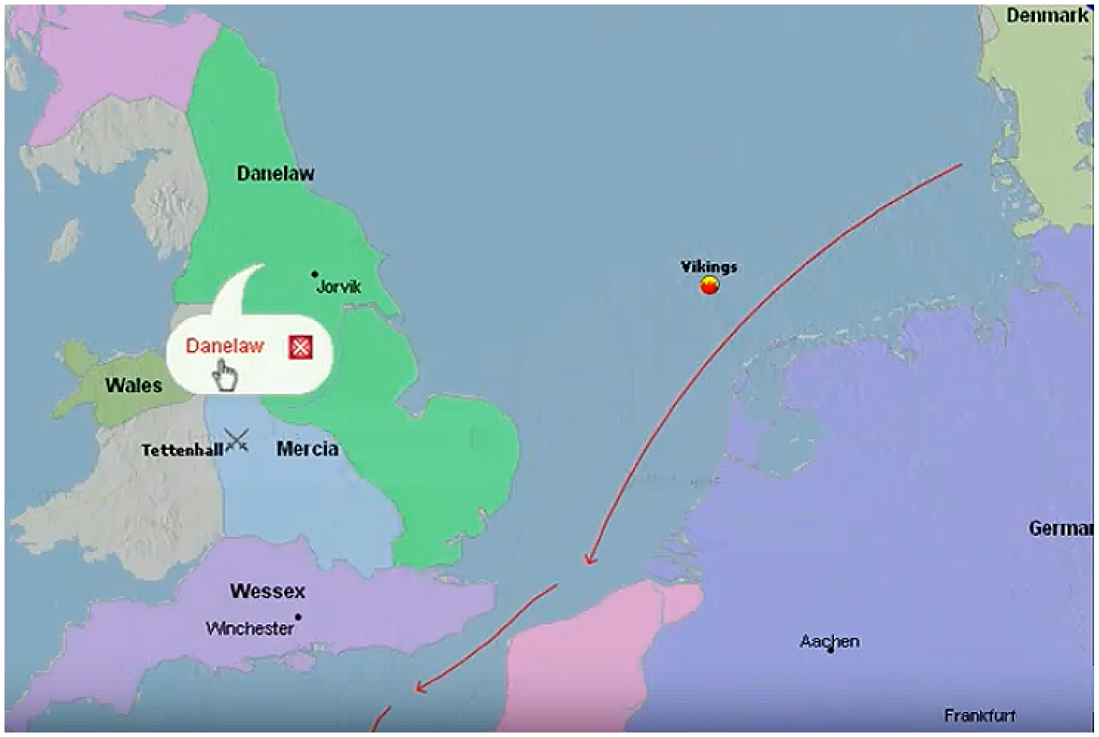

Figure 3.2: Viking route from Denmark to Paris in 845AD (from GeaCron).

the more detailed *Mathematics Subject Classification* (MSC) scheme used by the Mathematical Reviews and Zentralblatt MATH databases.

3.4 User Data

Personal data on users of the Ludii Portal and Ludii System may be stored on the Ludii Server. This has obvious implications for the GDPR and will be kept privately and securely according to UM data management policy at all times.

No user data will be kept without explicit permission from the relevant users. Users must be at least 18 years of age, or have permission from a responsible guardian, before doing so. User data will be anonymised as much as possible, but may contain potentially sensitive information volunteered by users. User data might include:

- Age.
- Gender.
- Current geographical region.
- Ethnic background.
- Cultural identity.
- Game preferences.
- Mechanism preferences.
- Experience with particular games and games in general.

3.5 Features

Features for biasing MCTS playouts learnt during self-play may be stored separately in the Ludii Server as plain text files with extention ".ftr". Related sets of features may be stored in the same file. Each file may be uniquely identified according to a naming system (yet to be devised). Features may be identified by the string description of their rotated and reflected template with lowest alphabetical order. The feature format is described elsewhere [8, 9].

Note that features might instead be stored as metadata inside each game description, if they prove compact enough and only a small number of features prove necessary for each game. Otherwise, features will be stored on the Ludii Server as described above, while game descriptions specify the unique name of relevant features and the weight with which they are to be applied.

4
Ludii Portal

The Ludii Portal is a suite of public-facing web pages through which external users will access the Ludii System and its public content such as the core game descriptions, and request various services. These web pages may be hosted on various machines, including machines owned by UM, the DLP and/or team members.

4.1 Online Catalogue

The complete list of games described in the main game database housed on the Ludii Server should be presented to external users on a dedicated web site. The list should be organised according to a number of different classification schemes, according to the user's needs. Potential classification schemes for organising games include:

- *Historical*: According to previous schemes used in classic books in the area, such as those by Murray [13], Bell [14] or Parlett [2].
- *Practical*: Consistent with the scheme of Type, Category, Mechanisms and Family used by the popular BoardGameGeek online boardgame database[1], which has been fine tuned by thousands of players over the last two decades through everday experience.
- *Functional*: By base state type as implemented in the Ludii System, e.g. colour-per-cell, index-per-cell, count-per-cell, index-state-per-cell, stack-per-cell, etc.
- *Superficial*: By name.
- *Geographical*: By country, continent, region, etc.
- *Cultural*: By culture.
- *Temporal*: By date first played.

[1]https://www.boardgamegeek.com/boardgame/33767/yavalath

A good model for this interactive catalogue might be the Glottolog[2] online catalogue of the world's languages, language families and dialects. This presents users with a catalogue of over 4,500 languages entries with phylogenetic links, which may be organised according to various criteria and manually searched.

4.2 Playing Arena

A public virtual playing arena should be provided for each game, such as that provided BoardSpace[3]. Note, however, that code run on local machines at UM should only coordinate play between users, whereas the heavy computation of move planning should occur remotely on the client side on an app downloaded for actually playing the game and running AI move planning on the user's remote machines.

Typical functionality might include:

- Creating an account/logging on.
- Browsing available available games by their descriptions.
- Joining "rooms" for each game or category of games where users can chat or arrange games.
- Playing games with other remote users.
- Providing feedback and scores for games.

4.2.1 Sandpit

It may be also be desirable to provide a safe "sandpit" in which users can experiment with freeform play with selected virtual game equipment.

4.2.2 Visual Designer

The sandpit mode described above may be supplemented with a visual game designer, consisting of a GUI that presents ludemes in a structured visual form, and allows users to drag & drop ludemes to manually create game descriptions, which may then be compiled, played and evaluated.

4.3 Family Tree

A core aim of the project is to develop a "family tree" of traditional strategy games throughout recorded history. This will probably be a network rather than a tree as such.

Instead of genetic distance indicating genetic distance between individuals, ludemic distance will be used as a measure of functional distance between games. This is based on the number of edits required to change one game description in ludemic form into another.

Users should be able to interactively navigate the resulting family tree/network by:

1. *Game*: The game's location in the tree/network should indicate its context in the history of games, which games led to it and which games follow from it.

[2]https://glottolog.org
[3]http://boardspace.net/english/index.shtml

2. *Ludeme*: Users should be able to track individual ludemes or ludemeplexes throughout the tree/network to see what games and contexts they were used in throughout history.

4.4 Interactive Maps

Another core aim of the project is to provide interactive maps that chart the spread of games, ludemes and associated mathematical ideas throughout history. This may be achieved by animating yearly snapshots of the probable occurrence of each game, ludeme/ludemeplex and tagged mathematical terms. This visualisation of this data will require careful design in order to convey the large amounts of information in a comprehensible manner.

The following study may provide some ideas for visualisation these mappings. The UCL Energy Institute recently developed an interactive map of the world's shipping routes, tracking individual ships across the globe and colour coding them by type.[4] For example, Figures 4.1 and 4.2 show an animation of world shipping routes with individual ships coloured by type, and a time lapse view of world shipping routes coloured by ship type, respectively.

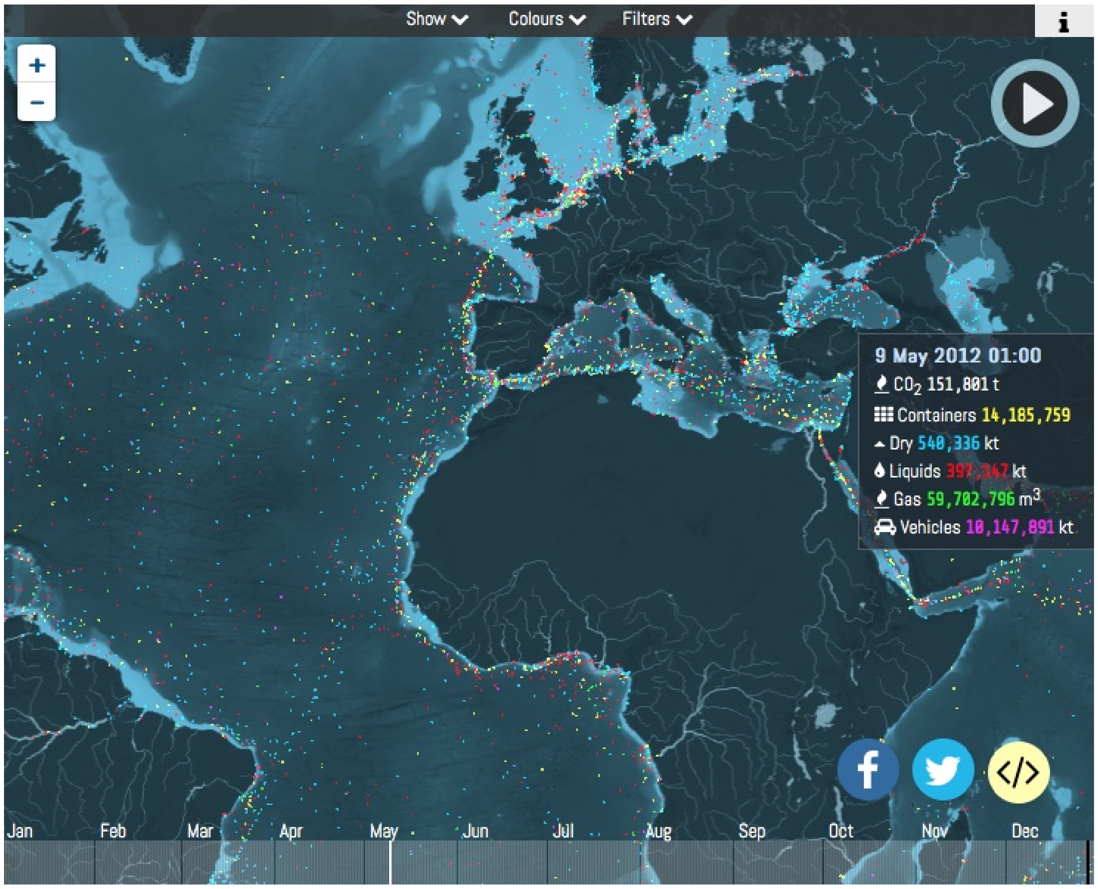

Figure 4.1: Animation of world shipping routes with individual ships coloured by type.

[4]https://www.vox.com/2016/4/25/11503152/shipping-routes-map

Such an approach might be applied to tracking the movement of games, ludemes and mathematical concepts. Key exemplars may be assigned particular colours, then intermediate examples coloured based on their ludemic distance to the most similar exemplars.

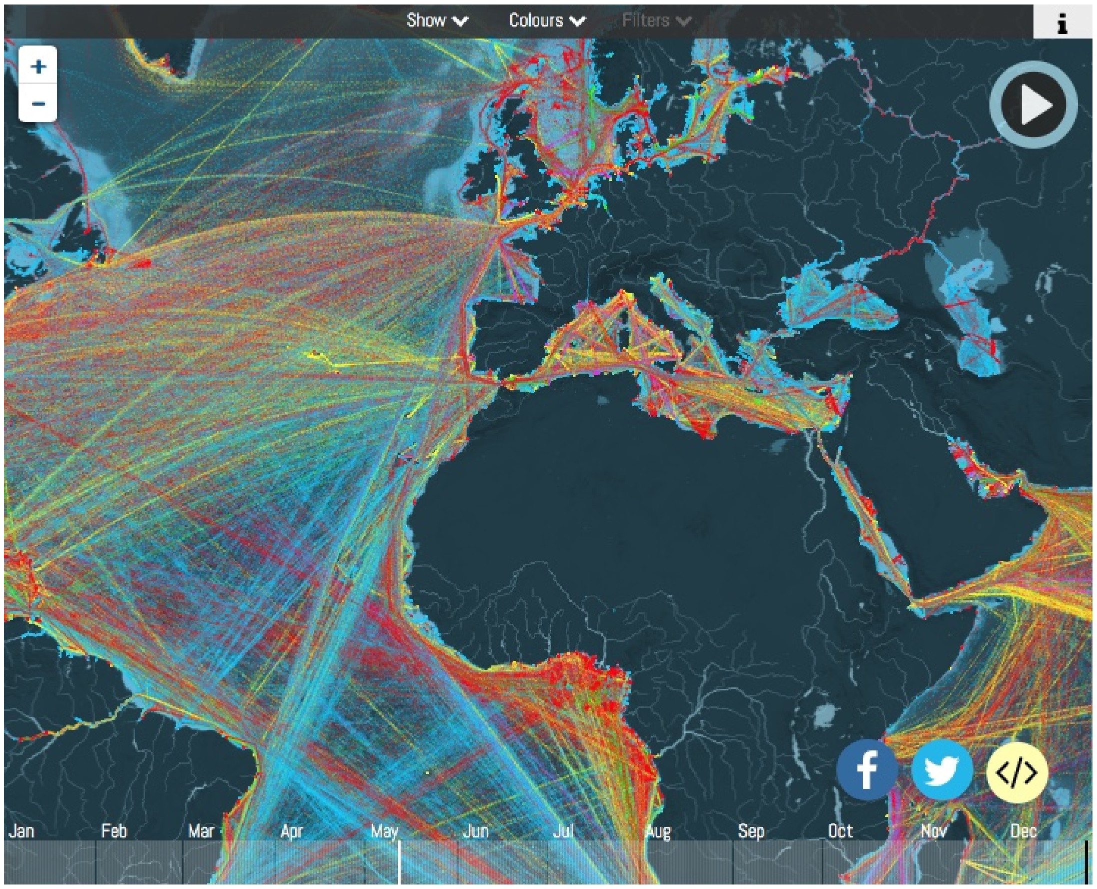

Figure 4.2: Time lapse view of world shipping routes coloured by ship type.

5 Conclusion

The following outlines the basic technical requirements of the Ludii general game system, being developed as part of the DLP. The Ludii System must provide the functionality to realise the project's aims. The Ludii Server must house the required data behind a secure firewall. The Ludii Portal must provide access to the public data and services provided by the Ludii system to external users, as described.

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 over 2018–23.

Bibliography

[1] F. Horn and A. de Voogt, “The development and dispersal of l’Attaque games”, *Board Game Studies*, 2008, pp. 43–52.

[2] D. Parlett, *The Oxford History of Board Games*, Oxford, Oxford University Press, 1999.

[3] C. Browne, “Modern Techniques for Ancient Games”, *IEEE CIG 2018*, Maastricht, IEEE Press, 2018, pp. 490–497.

[4] D. Parlett, “What’s a Ludeme?”, *Game & Puzzle Design*, vol. 2, no. 2, 2016, pp. 83–86.

[5] C. Browne, “A Class Grammar for General Games”, *Computers and Games (CG 2016)*, Leiden, Springer, LNCS 10068, 2016, pp. 169–184.

[6] C. Browne and E. Piette, LUDII *Programming Guide*, technical report, Department of Data Science and Knowledge Engineering, Maastricht University, 2018.

[7] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis and S. Colton, “A Survey of Monte Carlo Tree Search Methods”, *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, 2012, pp. 1–43.

[8] C. Browne, D. J. N. J. Soemers and E. Piette, ”Strategic Features for General Games”, *AAAI 2019*, workshop on *Knowledge Extraction from Games (KEG)*, Hawaii, AAAI Press, 2019.

[9] D. J. N. J. Soemers, C. Browne and E. Piette, ”Biasing MCTS with Features for General Games”, *CEC 2019*, Wellington, IEEE Press, 2019.

[10] C. Browne, *Evolutionary Game Design*, Springer, Berlin, 2011.

[11] ESRI, *ESRI Shapefile Technical Description*, technical report, Environmental Systems Research Institute, 1998.
https://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

[12] C. Browne and J. N. Silva, “Mathematics Through Games”, *Board Game Studies*, Bologna, BGS Press, 2019 (submitted).

[13] H. J. R. Murray, *A History of Board-Games other than Chess*, Oxford University Press, Oxford, 1952.

[14] R.C. Bell, *Board and Table Games from Many Civilizations*, revised vols. I and II, Dover, New York, 1979.