

École polytechnique de Louvain

Porting Ludii to Android

Auteur : **Alexandre de Salle**
Promoteur : **Eric Piette**
Lecteurs : **Julien Liénard, Achille Morenville**
Année académique 2025–2026
Master [60] in Computer Science

CONTENTS

1	Introduction	1
2	State of the Art	3
2.1	General Game Playing	3
2.1.1	Difference between GGP Systems and GGP Agents.	3
2.1.2	The Three Fundamental Pillars of GGP.	3
2.1.3	Historical Evolution and Notable Systems (GGS)	4
2.2	Game Description Language	4
2.2.1	Stanford GDL (S-GDL)	4
2.2.2	Regular Boardgames (RBG).	4
2.3	Digital Ludeme Project.	5
2.4	GameTable	5
2.5	Ludii and the Ludemic Approach.	6
2.5.1	Ludemic Approach.	6
2.5.2	The Ludii System.	8
3	Methodology	11
3.1	Strategic Approach.	11
3.2	Tools and Technical Environment	12
3.3	Development and Adaptation Process	14
4	Implementation	17
4.1	Overall Architecture of the Android Port.	17
4.2	Emulation of Critical Java Libraries	17
4.3	Adaptation in original code	24
5	Result	29
6	Further Work	33
6.1	Functional Games	33
6.2	User Interface	34
6.3	Audio and Motion Feedback	34
7	Conclusion	37
	Bibliography	41
	Appendix	43
A	: GDL and RBG implémentation of Tic Tac Toe.	43
B	: Ludii Android survey	45

1

INTRODUCTION

Ludii[4] is an open-source software developed by a team of researchers as part of the Digital Ludeme Project[10], supported by the European Research Council(ERC). It allows users to play traditional or modern two-dimensional board games, as well as code them more efficiently through a system of ludemes, a concept that will be explored in greater detail later. Ludii is used in an interdisciplinary context to test games, pit artificial intelligences against each other, or observe the behavior of these AIs, thereby offering insight into their decision-making patterns. It is also widely utilized in academic and educational settings, enabling students to understand different AI models such as alpha-beta pruning or Monte-Carlo Tree Search (MCTS). Furthermore, the software contributes to the rediscovery and democratization of traditional games from various cultures, some of which are little-known or forgotten.

Currently, the project is maintained under the COST Action GameTable[8], which aims to establish an international and interdisciplinary network to develop methodologies and applications that leverage artificial intelligence to study, reconstruct, and preserve the intangible cultural heritage of games. Among its objectives is the design of AI agents with more human-like behavior, using them to analyze the mathematical properties of games. This work aspires to define methods for preserving games as cultural heritage, both ancient and modern, and to design educational programs centered around games.

This pursuit of "human-like" AI finds resonance in ongoing work such as the Cogni-Play[25] model, currently under development. CogniPlay aims to create general game-playing (GGP) agents whose behavior mimics human decision-making processes. Unlike superhuman AIs that rely on massive computational search, this model draws inspiration from cognitive science, particularly the Dual-Process Theory (intuitive System 1 and analytical System 2) and the mechanism of perceptual grouping ("chunking"). It combines an intuitive component, using spatial patterns to filter actions, and an analytical component performing a limited and selective tree search. Such a model, trained through expert iteration to develop cognitive biases similar to humans, however, requires substantial amounts of data for learning.

To meet this need, the preferred approach is often inverse reinforcement learning, which demands a significant volume of training data. Although Ludii, in its current form as a desktop and web application, allows for the collection of a considerable amount of data, it remains insufficient. The rise and democratization of smartphones, particularly those running Android [2], represent a strategic opportunity. Developing an Android version of Ludii would significantly broaden its audience and, consequently, the volume of available gameplay data, drastically increasing the resources needed to train human-behaving AI agents.

However, the current version of Ludii presents major limitations for mobile use. Its interface, designed for high-resolution desktop screens, does not automatically adapt to touchscreens or the various display sizes of mobile devices, which impairs the user experience. Achieving such a port therefore requires rigorous adaptation.

The objectives of this work are thus threefold. First, to analyze the requirements by identifying the key functionalities of Ludii to preserve and the specific technical constraints related to porting to Android, in order to guarantee optimal mobile use. Second, to develop a functional prototype of the application using Android tools such as Java, using the Dalvik[16] virtual machine, and Android Studio, drawing on mobile development best practices while preserving the integrity of the game models. Finally, to evaluate the results by testing the performance, compatibility, and ergonomics of the mobile application, comparing the results with the desktop version to measure the impact of the port on the user experience.

2

STATE OF THE ART

2.1 GENERAL GAME PLAYING

General Game Playing (GGP)[14, 15] is a subfield of artificial intelligence dedicated to creating autonomous agents capable of learning to play previously unknown games without requiring specific, pre-programmed knowledge for each one. These agents are provided solely with a formal description of the game's rules and must, within a limited timeframe, interpret these rules and objectives, analyze the current game state, and develop effective strategies to compete. This domain is widely regarded as a critical challenge for advancing intelligent systems that can adapt to novel and unpredictable environments, and is often viewed as a significant step towards the development of Artificial General Intelligence (AGI)[6].

2.1.1 DIFFERENCE BETWEEN GGP SYSTEMS AND GGP AGENTS

A fundamental distinction exists between General Game Systems (GGS) and General Game Playing agents. GGP Systems, or General Game Systems, constitute the foundational software environments. They provide the necessary infrastructure to parse formal game descriptions, simulate the game environment, manage interactions between entities, and execute complete matches. In contrast, GGP Agents are the artificial intelligence programs that operate within these systems. Their core function is to consume the provided rule descriptions, engage in reasoning about the game state, and ultimately make strategic decisions to play effectively.

2.1.2 THE THREE FUNDAMENTAL PILLARS OF GGP

The field of General Game Playing rests upon three core principles. The first is generality, which refers to an agent's capacity to play a wide and diverse array of games. This includes games with varying numbers of players, games based on perfect or imperfect information, and those that are turn-based or simultaneous. This stands in opposition to specialized AI designed for a single specific game. The second pillar is the formal game structure. This involves the use of a standardized Game Description Language (GDL) to represent a game's rules in a precise, unambiguous, and machine-readable format, ensuring consistent

interpretation. The final pillar is the time constraint. Agents must perform all reasoning, analysis, and decision-making under strict computational time limits, which necessitates the design and implementation of highly efficient algorithms.

2

2.1.3 HISTORICAL EVOLUTION AND NOTABLE SYSTEMS (GGS)

The conceptual foundations for General Game Playing were laid in the 1960s, with more concrete systems beginning to emerge in the 1990s, such as SAL, Metagamer, and Hoyle. The modern era of GGP was largely initiated by research at Stanford University in the early 2000s, which produced foundational work like GGP-Base and the Game Description Language (GDL). Among the notable systems that have since been developed are GGP-Base[26], often considered the first modern framework and a cornerstone for academic competitions; Zillions of Games[27], which utilizes its own proprietary rule description language (ZRF); Regular Boardgames[19], a system emphasizing expressiveness for deterministic, perfect-information board games; Openspiel[20], a research framework focused on imperfect-information games and accessible via Python APIs; and Polygames[7], a tool from Facebook AI Research optimized for deep learning and self-play training on specific games. The Ludii system, which will be discussed in greater detail subsequently, also occupies a significant place within this landscape.

2.2 GAME DESCRIPTION LANGUAGE

Game Description Languages (GDLs) [15] are formal languages designed to describe the rules of a game in an unambiguous manner, serving as a bridge between the game and the AI agent. They can be broadly categorized into two main types: low-level and high-level languages. Low-level languages employ generic constructs, such as logical formalisms. They tend to be verbose, unintuitive, difficult to write, and challenging to debug. In contrast, high-level languages use terminology dedicated to games, making them concise, readable, and accessible to non-experts.

2.2.1 STANFORD GDL (S-GDL)

Stanford's GDL (S-GDL) is a low-level language based on first-order logic. It offers several advantages, including high generality and the existence of extensions for handling complex games: GDL-II for games with hidden information and GDL-III for epistemic games. However, it also suffers from significant limitations. Its runtime inefficiency stems from costly logical evaluations, which constrain the complexity of games that can be practically modeled (e.g., Go, Chess). The language is verbose, resulting in lengthy descriptions. It lacks modularity, as even minor modifications often necessitate extensive rewrites. Furthermore, it is unintuitive, requiring the redefinition of basic concepts—such as boards and arithmetic—for every new game.

2.2.2 REGULAR BOARDGAMES (RBG)

The Regular Boardgames (RBG) approach employs a dual-language system featuring convertible high-level and low-level syntaxes. Its theoretical foundation lies in regular language theory. Its advantages include greater efficiency, as it is engineered for more performant execution compared to S-GDL. It also offers improved readability through a concise and

comprehensible high-level syntax. Finally, it provides considerable flexibility, facilitating code modification and reuse across different games. Its primary limitation is its restriction to deterministic, turn-based games with perfect information.

2.3 DIGITAL LUDEME PROJECT

The Digital Ludeme Project [10] is a research initiative funded by an ERC (European Research Council) grant, with the objective of understanding traditional strategy games from around the world and throughout history, using artificial intelligence and computer science. The methodology involves modeling games in the form of ludemes and subsequently creating a software system capable of modeling, playing, and evaluating them. The ambitions of this project are threefold: first, to build an exhaustive database of traditional strategy games; second, to reconstruct the rules and lost knowledge of ancient games; and finally, to map the historical and cultural diffusion of games and their concepts.

2.4 GAME TABLE

The objective of COST Action GameTable [8, 24] is to establish an international and interdisciplinary network that leverages artificial intelligence to study, reconstruct, and preserve the cultural heritage – both material and intangible – of board games. These games represent a global heritage, a portion of which has been lost due to colonization, imperialism, and commercialization. Furthermore, research on games is fragmented across multiple disciplines – including computer science, history, archaeology, mathematics, and pedagogy – which typically collaborate very little. GameTable proposes a methodology centered on developing holistic approaches that combine human-like AI techniques, the analysis of mathematical game properties, models from game theory, historical knowledge, and an intercultural understanding of play. However, several significant challenges must be addressed to realize these goals: finding new ways to study and preserve ludic heritage; reconstructing missing rules for incomplete games and simulating their matches; using AI to identify new historical connections and trace the evolution and diffusion of games across time and space; creating new pedagogical tools that use games to teach concepts (such as mathematics and AI); and developing general, culturally diverse AI approaches that can compete at a human level. Officially titled COST Action CA22145 "Computational Techniques for Tabletop Games Heritage," the initiative began in 2023 and will run until October 2027. It currently involves over one hundred participants from approximately 48 countries worldwide. The action is organized into five Working Groups (WGs). WG1 focuses on research, planning, and learning; WG2 addresses the cultural heritage aspects of games, which is the project's core; WG3 is dedicated to the automated design of games; WG4 handles the mathematics in games; and finally, WG5 manages implementation, dissemination, and education. The expected outcomes of this COST Action include advancements in interdisciplinary research, sophisticated methods for preserving ludic heritage, and innovative educational programs centered on games.

2.5 LUDII AND THE LUDEMIC APPROACH

Ludii [9] is a General Game Playing (GGP) system [5] developed within the framework of the Digital Ludeme Project (DLP) with the goal of creating a comprehensive database of historical and modern board games and supporting research into their history. Its foundational concept is the representation of games via ludemes—modular game units such as rules, mechanics, actions, or pieces—using a dedicated language known as L-GDL.

Its main applications include historical and digital archaeoludology research, enabling the reconstruction and analysis of ancient games from archaeological fragments. It also serves as a platform for general artificial intelligence (GGP) research, progressively replacing systems such as Stanford's GDL and acting as a new benchmark for competitions and research—for example, in improving Monte Carlo Tree Search (MCTS) or transfer learning. Finally, it functions as a tool for education and design, providing an accessible interface for teaching and the procedural generation of new games, such as through the use of Large Language Models (LLMs).

However, Ludii has certain limitations, including difficulty in modeling boardless games where the playing surface is constructed dynamically.

2.5.1 LUDEMIC APPROACH

The ludemic approach is founded on the concept of ludemes. A ludeme constitutes a fundamental unit of ludic information, serving as an elementary building block for game construction. The term was popularized by game historian David Parlett, though the underlying concept was employed earlier by Alain Borvo in 1970 and later refined by Cameron Browne. An illustrative analogy positions ludemes as the foundational components of games, akin to words forming a language or genes in biology.

A ludeme is characterized by four essential properties: discreteness, transferability, composability, and contrastiveness. Its discrete nature means it represents a distinct and identifiable unit of information, such as a "move," "capture," or an "8x8 board." Transferability allows a ludeme to be reused across different games or combined with others; for example, an "L-shaped move" ludeme can define a knight in chess or a similar piece in another context. Composability indicates that a ludeme can be atomic or composite, forming hierarchies where complex rules are built from simpler ones. Finally, a ludeme is contrastive, meaning that altering it fundamentally changes the game's behavior or structure, as substituting "victory by alignment" with "victory by encirclement" creates a distinct game.

The expressive scope of a ludeme is broad, enabling it to encapsulate any game aspect: physical equipment like boards and pieces, structural elements such as player count and turn order, core rules governing actions and termination conditions, and fundamental mechanics.

This approach grants Ludii significant advantages over traditional game description languages like Stanford's GDL or RBG. Descriptions achieve high readability by employ-

ing self-explanatory, domain-specific keywords. Maximal modularity allows games to be assembled from reusable components, where rule modifications correspond to changes in specific ludemes. The abstraction level remains high, focusing on declarative game concepts rather than implementation details. Consequently, game descriptions are concise, with complex games like Havannah being definable in approximately twenty readable lines.

As illustrated in the Figure 2.1, the structure contains a primary game ludeme, which encompasses three subordinate ludemes: players, equipment, and rules.

```

1 (game "Tic-Tac-Toe"
2   (players 2)
3   (equipment {
4     (board (square 3))
5     (piece "Disc" P1)
6     (piece "Cross" P2)
7   })
8   (rules
9     (play (move Add (to (sites Empty))))
10    (end ("Line3Win"))
11  )
12 )

```

Figure 2.1: Ludii Implementation of Tic Tac Toe. Source: [github.ludeme.ludii](https://github.com/ludeme/ludii)

The players ludeme sets the number of participants; in this instance, there are two, though this parameter can be adjusted to accommodate more players depending on the game's design.

The equipment ludeme serves to configure the game's physical components and player assets. Here, it establishes a square 3x3 board and assigns a distinct piece type to each player: player 1 receives a disc, while player 2 receives a cross, as depicted in Figure 2.1.

Finally, the rules ludeme defines the game's mechanics. It begins by specifying the core play action, which in this case is adding a piece to an empty cell, as declared with the play ludeme. The termination condition(s) are then defined under the end ludeme, which employs the composite ludeme Line3Win for determining a win, visualized in Fig 2.2. This structure is sufficient for a fully functional game.

```

1 // If the mover has 3 pieces aligned, the mover is winning. This ludemplex can be used only in an ending condition.
2 // #1 = Directions of the line (by default Adjacent) or role type of the owner of the pieces making the line.
3 // @example ("Line3Win")
4
5 (define "Line3Win"
6   (if (is Line 3 #1) (result Mover Win))
7 )

```

Figure 2.2: Ludii Implementation of Line3Win. Source: [github.ludeme.ludii](https://github.com/ludeme/ludii)

Beyond this core game ludeme, an optional metadata block can be included, as shown in Figure 2.3. This block utilizes the info ludeme to provide a literary rule explanation, a game description, source citations, and origin information. It also allows for specifying

a preferred AI agent—in this case, a custom Tic-Tac-Toe_ai designed specifically for this game.

2

```

1 (metadata
2
3   (info
4     {
5       (description "Tic-Tac-Toe is a game of alignment popular among children. It is known from the nineteenth
6         century in England and the United States, but may be older.")
7       (aliases {"Noughts and Crosses" "Oughts and Crosses" "Tik Tak Tol" "Tripp Trapp Trull" "Tick-Tack-Toe"})
8       (rules "Play occurs on a 3x3 grid. One player places an X, the other places an O and players take turns placing
9         their marks in the grid, attempting to get three in a row of their colour.")
10      (source "Murray 1951: 40.")
11      (id "100")
12      (version "1.3.13")
13      (classification "board/space/line")
14      (credit "Eric Piette")
15      (origin "This game was played in Unknown, from around 1850 to 1883.")
16    }
17  )
18  (ai
19    "Tic-Tac-Toe_ai"
20  )
21 )

```

Figure 2.3: Ludii Implementation of Metadata for Tic-Tac-Toe. Source: [github.ludeme.ludii](https://github.com/ludeme/ludii)

This structure, based on ludemes and their modular approach, enables a clear game description in a human-readable format. Modifying the board size requires altering only a single parameter within the corresponding ludeme. This highlights the simplicity and adaptability inherent to this methodology. The advantages become particularly evident when comparing the Tic-Tac-Toe implementation depicted in Figure 2.1 with its counterparts in GDL and RBG, presented in Figures A.1 and A.2, respectively.

2.5.2 THE LUDII SYSTEM

Ludii [13] employs a methodology centered on a core Grammar class. Its Game Description Language (L-GDL) is generated automatically from the underlying Java source code. This process establishes a guaranteed one-to-one mapping between a language keyword, such as board or move, and its implementing Java class. Consequently, the programming language Java effectively functions as the game description language itself. This architecture enables Ludii to theoretically support any rule programmable in Java while abstracting its complexity, presenting users with a simplified grammatical interface. During execution, game descriptions structured as ludeme trees are dynamically instantiated into Java objects through reflection, invoking the constructors of their corresponding classes.

The foundation of the system is a comprehensive library where each ludeme is defined as a Java class implementing the Ludeme interface, with a structural representation as depicted in Figure 2.4. A game, denoted as G, is represented by an immutable Game object that encapsulates all constituent ludemes—players, equipment, and rules. Internally, this object is structured as a tree of ludemes. Gameplay logic operates by recursively traversing this tree to evaluate the game state, compute legal moves, and determine outcomes. The logical kernel is built upon several pivotal classes: the Context, which serves as the primary container for a game session; the State, representing the current configuration including piece positions and scores; the Trial, recording the complete historical sequence of states

and actions for a match; and the Action, describing an atomic operation such as adding or removing a piece to be applied to the state.

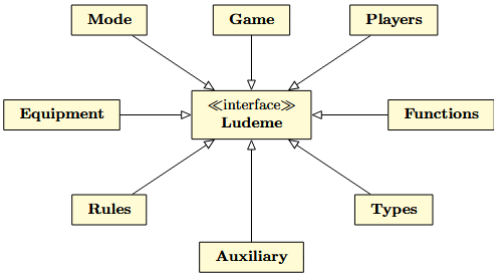


Figure 2.4: Ludemic categorisation in Ludii. Source: [12]

The system implements a Model-View-Controller (MVC) architecture [11]. This facilitates the automatic rendering of games through dedicated View classes like BoardView and PlayerView, while a controller manages user input to propagate state changes. A notable feature is the system’s capacity to support gameplay involving humans, artificial intelligence agents, or any combination thereof concurrently. The general user interface, illustrated in Figure 2.5, is organized into five primary sections: the game board (Board-View), player information (PlayerView), a textual information panel (TabView), a set of control buttons (ToolView), and the primary application menu (MainMenu).

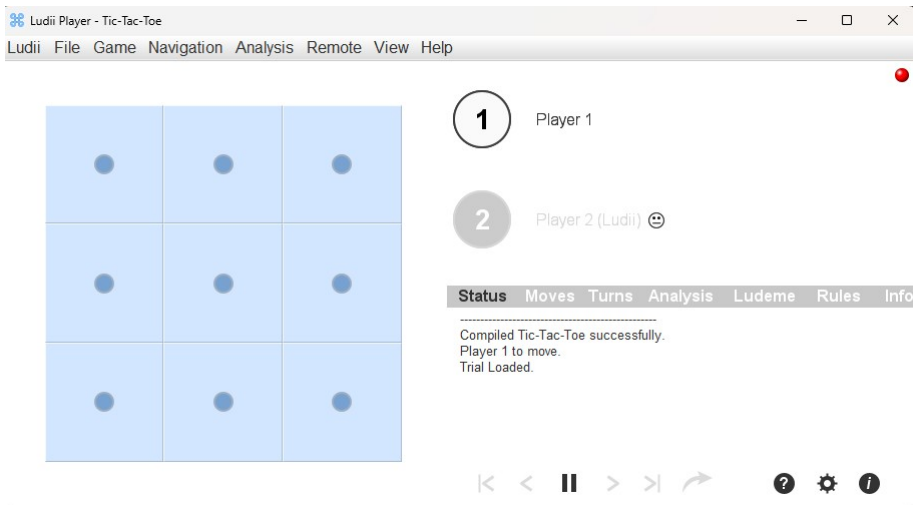


Figure 2.5: Ludii interface.

This design confers several distinct advantages. The system demonstrates pronounced simplicity and clarity, as game descriptions are concise, employ intuitive keywords, and are straightforward to author and modify, thereby enhancing accessibility for non-specialists.

Its generality is evidenced by its capacity to model an extensive variety of games without necessitating alterations to its core architecture. The platform is highly extensible due to its modular design; introducing a novel game concept merely requires the addition of a new Java class to the ludeme library, eliminating the need for extensive system overhauls. From a performance perspective, Ludii is efficient. This efficiency stems from compact linguistic descriptions, optimized reasoning via memory-conscious state representations and pre-computation strategies, and the capacity for fine-grained optimization through direct control over ludeme implementations. The tree-based representation of ludemes renders the system particularly amenable to evolution, as it seamlessly integrates with evolutionary algorithms for procedural game design, where combinations and mutations frequently yield viable and engaging games. Furthermore, the system holds significant cultural application value, acting as an instrumental tool for Digital Archaeoludology by enabling the reconstruction, classification, and historical analysis of traditional games through integration with cultural databases. Finally, it possesses proven universality for the domain of finite deterministic games with perfect information

3

3

METHODOLOGY

3.1 STRATEGIC APPROACH

The initial phase of the project involved a comprehensive analysis of the source code to distinguish between components that could remain unchanged and those reliant on dependencies incompatible with the Android platform [16]. The primary non-functional dependencies identified were the Java awt [21] and swing [22] graphical libraries, which consequently required the most significant porting effort. Additional incompatible dependencies included file input/output management and, critically, user input handling, which differs substantially between desktop and mobile touch-based interfaces. The core functionalities impacted by these dependencies were graphical rendering, asset management (such as image loading), parsing of Ludii game description files, and user input processing.

Regarding the porting strategy, three distinct methodological avenues were considered: a complete from-scratch reimplementing of Ludii, a selective reimplementing limited to files containing calls to the critical libraries, or the behavioral emulation of these libraries through custom classes mimicking the original names, constructors, and method signatures. The third option—emulating the non-functional libraries for Android—was ultimately selected. This approach was supplemented by direct modifications to certain core Ludii classes in cases where inherent source-level incompatibilities between standard Java and the Android environment necessitated adjustments.

This strategic decision was predicated on two key factors: it promised a viable development timeline and, more importantly, ensured that future updates to the desktop application could, in most instances, be synchronized directly with the Android version. Exceptions to this synchronization are confined to code sections that were entirely rewritten or that invoke functionalities not yet ported. The emulating classes, which replicate their awt, swing, or other Java library counterparts, are organized within a dedicated package named `androidUtils`, as illustrated in Figure 3.1.

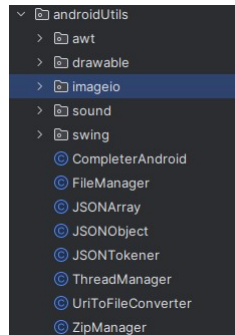


Figure 3.1: Structure of the androidUtils emulation package. Source: [1]

3.2 TOOLS AND TECHNICAL ENVIRONMENT

The Android ecosystem offers multiple pathways for application development, including the use of native languages such as Java or Kotlin, cross-platform frameworks like Flutter (Dart), React Native (JavaScript), or .NET MAUI (C#), as well as game engines such as Unity or Unreal that support deployment to both desktop and mobile platforms. For the present porting effort, Java was selected as the primary implementation language. This choice was motivated by the fact that Ludii’s core codebase is already written in Java, which greatly simplified code reuse and avoided the need for a complete rewrite. Consequently, a substantial portion of the existing code—including AI agents and the game controller—could be retained with only minimal modifications.

The original Ludii project is developed in the Eclipse Integrated Development Environment (IDE). Because Eclipse is no longer a straightforward or commonly supported tool for contemporary Android development, the project was migrated to Android Studio . This transition was a natural consequence of the decision to use Java for the Android port, as Android Studio [18] is the official and most comprehensive IDE for Android projects based on Java or Kotlin. The choice of development environment is thus intrinsically linked to the selected programming language; a different technology stack (e.g., Dart with Flutter or a game engine) would have required a different toolchain.

Moving from Eclipse to Android Studio necessitated a significant structural adaptation of the project. Eclipse supports workspace configurations composed of multiple interdependent sub-projects, an architecture originally employed by Ludii. Android Studio, however, does not natively support an identical multi-project structure within a single application module without additional complex configuration. To resolve this incompatibility, all necessary sub-projects from the original Ludii solution were consolidated into a single, monolithic project structure within the PlayerAndroid application module. This consolidation is visually contrasted with the original PlayerDesktop structure in Figures 3.2 and 3.3. A critical objective during this restructuring was to preserve the original Java package hierarchies and names in their entirety, ensuring that all internal references, imports, and reflection-based mechanisms within the Ludii code remained valid and avoiding extensive, error-prone modifications to the core source files.

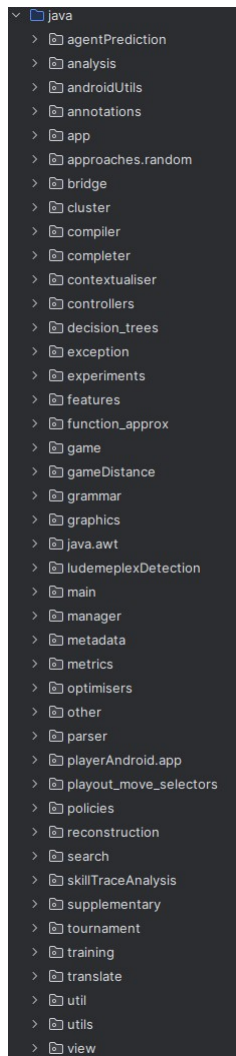


Figure 3.2: Structure of the AndroidPlayer project. Source: [1]

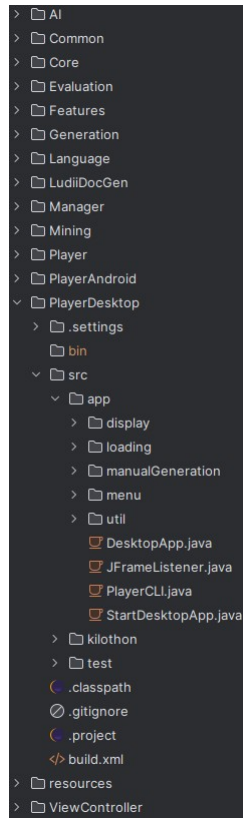


Figure 3.3: Structure of the Ludii project. Source: [11]

After addressing the project structure, the next step was to select Java libraries compatible with Android for the graphical components. The chosen solution was to adopt the core Android APIs provided by AndroidX [17]. AndroidX encompasses the Android Jetpack libraries and represents the modern, recommended approach for Android development in Java, having superseded the deprecated Android Support Library. AndroidX is a repackaged and unified version of the previously fragmented Support Library, offering a predictable and consistent hierarchy. A key advantage is that AndroidX operates at a level of abstraction comparable to awt and swing. Although significant adaptations were still required, this conceptual proximity enabled a more direct mapping of graphical elements from awt to AndroidX within the custom porting classes.

3.3 DEVELOPMENT AND ADAPTATION PROCESS

The development and adaptation process followed an iterative cycle comprising several key phases. The initial phase involved a targeted code analysis, focusing on modules suspected of containing dependencies on awt and swing. This included the view components and, more broadly, the app and DesktopApp packages—which were fundamentally restructured

into a new `AndroidApp` package—as well as the graphics package, which was expected to house critical rendering calls. This audit also extended to the examination of external libraries used by the system.

From this analysis, a comprehensive list was generated, detailing the specific awt and swing classes and methods requiring porting, alongside external libraries that needed replacement with Android-compatible equivalents. Fortunately, direct Android equivalents were available for most dependencies. A notable exception was the Batik library, which required a dedicated porting effort of approximately 200 lines of code. The subsequent phase focused on implementing these required classes and methods using the Android framework. Development began with the foundational awt classes, which would later be utilized by the swing layer, before proceeding to the swing implementations and other dependencies such as file I/O handlers.

The first two phases—analysis and implementation—were repeated iteratively to ensure all instances of non-functional dependencies were identified and addressed, culminating in a preliminary set of custom Android classes. Following this, the adaptation of the user interface commenced. The process started from the existing desktop codebase within the `DesktopApp` package, which was incrementally adapted while striving to maintain visual and functional coherence. During this UI adaptation, several of the previously implemented Android utility classes were refined and corrected. The interface was deliberately streamlined for mobile use, retaining only core elements such as the game board, player panel, game rules display, and a simplified menu. Features deemed non-essential for mobile gameplay, such as tools for visualizing AI decision paths, were omitted.

The final phase involved manual testing of the application, verifying successful launches, correct UI element placement, and proper responsiveness to user interactions. This iterative four-step methodology—analysis, library implementation, UI adaptation, and testing—was instrumental in achieving a functional application. Throughout these stages, the custom `androidUtils` package evolved significantly, as illustrated by the structure of the `androidUtils.awt` and `androidUtils.swing` sub-packages shown in Figures 3.4 and 3.5, respectively.

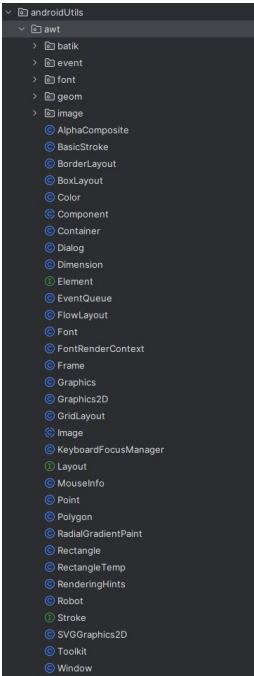


Figure 3.4: Structure of the androidUtils.awt emulation package. Source: [1]



Figure 3.5: Structure of the androidUtils.swing emulation package. Source: [1]

4

IMPLEMENTATION

4

This chapter details the concrete technical execution of the Ludii port to Android. It describes the resulting software architecture, the specific solutions developed to overcome technical hurdles, and the key components of the implemented system. The focus is on the **how**—the practical realization of the strategic approach outlined in the previous chapter.

4.1 OVERALL ARCHITECTURE OF THE ANDROID PORT

The port is built upon a consolidated monolithic structure within Android Studio, as necessitated by the transition from Eclipse. All core Ludii sub-projects were integrated into a single ‘PlayerAndroid’ module, meticulously preserving the original Java package hierarchies (e.g., ‘game’, ‘util’, ‘ai’) to maintain code integrity. The architecture hinges on a custom compatibility layer that intercepts and redirects calls from the original codebase to Android-specific implementations.

A central design principle was the creation of the ‘androidUtils’ package, which hosts emulated versions of incompatible Java Standard Edition libraries. The application’s entry point and main control flow were re-housed within a re-engineered ‘AndroidApp’ package, replacing the original ‘DesktopApp’. This new package orchestrates the Android activity lifecycle, manages the mobile-specific user interface, and serves as the bridge between the native Ludii engine and the Android operating system.

The complete source code is available in the project repository: LudiiAndroid.

4.2 EMULATION OF CRITICAL JAVA LIBRARIES

The core technical challenge was providing functional equivalents for the ‘java.awt’ and ‘javax.swing’ packages, which are unavailable on Android. Our solution was not a full reimplementation but a targeted mimicry via a custom compatibility layer housed in the ‘androidUtils’ package. This layer intercepts calls from the original Ludii codebase and redirects them to Android-specific implementations, preserving the original API signatures to ensure compile-time and runtime compatibility.

The foundation of the graphical emulation is the Android ‘android.graphics’ API. The emulation began with the ‘awt’ package, specifically the foundational ‘Graphics2D’ and ‘Graphics’ classes, whose structures are outlined in Figures 4.1 and 4.2. These were implemented by leveraging Android’s ‘Canvas’, ‘Paint’, ‘Bitmap’, ‘RectF’, and related classes.

4

```

public class Graphics {
    Canvas canvas;
    Paint paint = new Paint();
    Font font;
    int fontSize;
    Bitmap bmp;
    AffineTransform at = null;
    private Color backgroundColor = Color.WHITE;
    protected Shape clip;
    private DashEffect dashEffect;
    private float[] dashEffectArray = new float[0];
    private float dashPhase = 0;

    public Graphics(Canvas canvas, Bitmap bmp) {
        public Graphics() {
        public Graphics(Canvas canvas) {
        public void setCanvas(Canvas canvas) {
        public Canvas getCanvas() { return canvas; }
        public Bitmap getBitmap() { return bmp; }
        public Paint getPaint() { return paint; }
        public void setBitmap(Bitmap bmp) {
        public Graphics create() {
        public Graphics create(int x, int y, int width, int height) {
        public static Graphics createGraphics(Bitmap bmp) {
        public void setFont(Typeface typeface, int fontSize) {
        public void setFont(Font font) {
        public void setPaint(RadialGradientPaint rpg) {
        public void setPaint(Color color) { paint.setColor(color.toArgb()); }
        public Font getFont() { return font; }
        public void setColor(int color) { paint.setColor(color); }
        public void setColor(Color color) { paint.setColor(color.toArgb()); }
        public void setStroke(Stroke stroke) {
        public void setDashArray(float[] dashArray, float phase) {
        public Stroke getStroke() {
        public void setRenderingHints(RenderingHints.Key key, Object value) {
        public void resetPaint() {
        public void setBackground(Color color) {
        public void clearCanvas() {
        public RectAngle2D getStringBounds(String text) {
        public void draw(Shape shape) {
        public void draw(GenericPath path) {
        public void draw(Ellipse2D.Double ellipse) {
        public void draw(Line2D.Double line) {
        public void drawString(String text, int x, int y) {
        public void drawRoundRect(int x, int y, int w, int h, int a, int z) {
        public void drawRoundRect(float x, float y, float w, float h, float a, float z) {
        public void drawImage(Image image, int x, int y, Paint paint) {
        public void drawImage(BufferedImage img, int srcx, int srcy, int srcwidth, int srcheight,
            int destx, int desty, int destwidth, int destheight, Object observer) {
        public void drawImage(BufferedImage image, Paint op, int x, int y) {
        public GeneralPath drawPolygons(int[] x, int[] y, float w, float h) {
        public void drawLine(float x, float y, float w, float h) {
        public void drawOval(int x, int y, int w, int h) {
        public void drawArc(int x, int y, int r, int r1, int startAngle, int endAngle) {
        public void drawArc(Arc2D.Double ellipse) {
        public void fill(Shape shape) {
        public void fill(GenericPath path) {
        public void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle) {
        public void fillRect(float x, float y, float w, float h) {
        public void fillRect(int x, int y, int w, int h) {
        public void fillRoundRect(int x, int y, int w, int h, int a, int z) {
        public void fillOval(int x, int y, int w, int h) {
        public GeneralPath fillPolygons(int[] x, int[] y, float w, float h) {
        public void dispose() { if (bmp != null) bmp.recycle(); }
        public void rotate(double radians, double v, double v1) {
        public void rotate(int rotate) { canvas.rotate((float) Math.toDegrees(rotate)); }
        public void translate(int x, int y) { canvas.translate(x, y); }
        public void setTranslucent() { paint.setAlpha(128); }
        public void setComposite(AlphaComposite composite) {
        public void transform(AffineTransform at) {
        public void setTransform(AffineTransform at) {
        public FontRenderContext getFontRenderContext() { return new FontRenderContext(paint); }
        public FontRenderContext getFontMetrics() { return new FontRenderContext(paint); }
        public Shape getClip() { return clip; }
        public void setClip(Shape clip) {
        public Color getClip() { return new Color(paint.getClip()); }
        public int getClip() {
        public int getClip() {
        public void save() { canvas.save(); }
        public void restore() { canvas.restore(); }
        public void drawPaint(Paint paint) { canvas.drawPaint(paint); }
        public void renderToCanvas(SVG svg, int width, int height, RenderOptions options) {
        public AffineTransform getTransform() {
    }

```

Figure 4.1: Structure of Graphics emulation class. Source: [1]

```

public class Graphics2D extends Graphics {
    public Graphics2D(Canvas canvas, Bitmap bmp) { super(canvas, bmp); }
    public Graphics2D() { super(); }
    public Graphics2D(Canvas canvas) { ... }
    public Graphics2D(Bitmap bt) { super(bt); }

    public static Graphics createGraphics(Bitmap bmp) { ... }

    @Override
    public Graphics2D create(int x, int y, int width, int height) { ... }
}

```

Figure 4.2: Structure of Graphics2D emulation class. Source: [1]

A ‘Bitmap’ serves as the pixel buffer, while a ‘Canvas’ object provides the drawing context, executing all rendering commands. The visual properties of these operations are

controlled by a ‘Paint’ object, which encapsulates color, stroke, text attributes, and style. A key adaptation involved managing the ‘Paint.Style’ attribute: ‘Paint.Style.STROKE’ is used for drawing outlines (e.g., ‘drawRect’), and ‘Paint.Style.FILL’ is used for filling shapes (e.g., ‘fillRect’). This necessitates correctly setting the ‘Paint’ object before each corresponding drawing call in the emulated class.

Standard drawing operations required parameter adaptation. For example, as shown in Figure 4.3, the ‘drawRoundRect’ function must first convert the AWT method’s parameters—‘x’, ‘y’, ‘width’, ‘height’, ‘arcWidth’, ‘arcHeight’—into an Android ‘RectF’ object. The third and fourth parameters (‘width’ and ‘height’) are used to calculate the right and bottom coordinates (‘x + width’, ‘y + height’) for the ‘RectF’ constructor. Similar adaptations were implemented for ‘drawOval’, ‘drawPolygon’, and ‘drawString’.

```
public void drawRoundRect(int x, int y, int w, int h, int a, int z) { 3 usages
    paint.setStyle(Paint.Style.STROKE);
    RectF rectF = new RectF((float)x, (float)y, (float)(w + x), (float)(h + y));
    canvas.drawRoundRect(rectF, a, z, paint);
    paint = new Paint(paint);
}
```

Figure 4.3: Example of a drawing function adaptation (‘drawRoundRect’). Source: [1]

The ‘Typeface’ class handles typography and is utilized within a custom ‘Font’ class that mimics its AWT counterpart. ‘Rect’ and ‘RectF’ are Android’s rectangle representations, serving as equivalents to AWT’s ‘Rectangle’ and ‘Rectangle2D.Float’. Corresponding emulation classes were created for the various AWT rectangle types (‘Rectangle’, ‘Rectangle.Double’, etc.). In our ‘Rectangle’ class, we store the four attributes: ‘x’, ‘y’, ‘width’, and ‘height’. As shown in Figure 4.1, various other geometric structures such as ‘Arc2D’, ‘Ellipse2D’, ‘Path2D’, and ‘Line2D’ were also emulated.

Ludii’s extensive use of SVG (Scalable Vector Graphics) for game pieces necessitated the emulation of the SVGGraphics2D class. An initial attempt to implement a full SVG rendering pipeline proved complex and unstable. The final solution combines two techniques. For reading and parsing SVG files, the AndroidSVG library is employed. For image data interchange, Base64 encoding is utilized: the original code’s SVG string data is replaced with a Base64-encoded representation, which can be decoded into a bitmap for display. Specifically, the Base64 library is used to transform a bitmap into a string and to decode such a string back into a bitmap. This approach leverages established Android libraries for vector graphics, resulting in a more robust and concise implementation. The SVGGraphics2D emulation class is shown in Figure 4.4.

```

public class SVGGraphics2D extends Graphics2D {

    public SVGGraphics2D(int xSize, int ySize) {...}
    private SVGGraphics2D() { super(); }
    public SVGGraphics2D(Bitmap bmp) { super(bmp); }

    @Override
    public Graphics2D create() {
        SVGGraphics2D g2dnew = new SVGGraphics2D();
        Bitmap bmpnew = Bitmap.createBitmap(bmp);
        g2dnew.canvas = new Canvas(bmpnew);
        g2dnew.bmp = bmpnew;
        g2dnew.paint = new Paint(Paint);
        g2dnew.font = new Font(Font);
        if (at != null) g2dnew.at = new AffineTransform(at);
        g2dnew.fontSize = fontSize;
        g2dnew.clip = clip.copy();
        return g2dnew;
    }

    public AffineTransform getTransform() { return at; }
    public static String bitmapToBase64(Bitmap bitmap) { 1 usage
        if (bitmap == null || bitmap.isRecycled()) return null;

        try (ByteArrayOutputStream baos = new ByteArrayOutputStream()) {
            bitmap.compress(Bitmap.CompressFormat.PNG, quality: 100, baos);
            return Base64.encodeToString(baos.toByteArray(), Base64.NO_WRAP);
        } catch (Exception e) {
            Log.e(tag: "BitmapConverter", msg: "Encoding error", e);
            return null;
        }
    }

    public static Bitmap base64ToBitmap(String base64) { 1 usage
        if (base64 == null || base64.isEmpty()) return null;

        try {
            byte[] decoded = Base64.decode(base64, Base64.DEFAULT);
            return BitmapFactory.decodeByteArray(decoded, offset: 0, decoded.length);
        } catch (Exception e) {
            Log.e(tag: "BitmapConverter", msg: "Decoding error", e);
            return null;
        }
    }

    public String getSVGDocument() {
        return SVGGraphics2D.bitmapToBase64(bmp);
    }

    public static BufferedImage getBufferedImageFrom64(String string64) { 4 usages
        Bitmap bt = SVGGraphics2D.base64ToBitmap(string64);
        return new BufferedImage(bt);
    }

    public String getSVGElement() { return getSVGDocument(); }
}

```

Figure 4.4: SVGGraphics2D emulation class. Source: [1]

The AWT ‘Image’ and ‘BufferedImage’ classes were also mimicked. The emulated ‘BufferedImage’ internally wraps an Android ‘Bitmap’. Common operations like ‘getGraphics()’ return an emulated ‘Graphics2D’ object configured to draw onto this internal ‘Bitmap’, maintaining the expected workflow from the original code. The ‘Image’ class wrapper uses an Android ‘Bitmap’ and is relatively simple. The majority of functionality resides in its subclass ‘BufferedImage’, which is more heavily utilized as the fundamental image representation in AWT and is employed by ‘Graphics’ and its derivatives.

The Batik library, directly tied to AWT for SVG processing, required the implementation

of five distinct classes. The primary class is ‘ImageTranscoder’, which modifies images based on rendering hints using a helper ‘ImageRenderer’ class (which in turn utilizes a ‘Canvas’ for image manipulation).

User input mechanisms were fundamentally different. The AWT ‘MouseInfo’ class, which provides cursor coordinates, was emulated using Android’s ‘MotionEvent’ system. The emulated class tracks the latest touch coordinates, simulating a pointer position. Similarly, AWT events (‘ActionEvent’, ‘FocusEvent’) and their corresponding listeners (‘ActionListener’, ‘FocusListener’) were mapped to Android’s event system. For instance, an ‘ActionListener’ is implemented as a runnable callback triggered by a view’s ‘OnClickListener’. ‘FocusEvent’/‘FocusListener’ was extended from Android’s ‘View.OnFocusChangeListener’.

Another significant class is the ‘Dialog’ emulation, which mimics the AWT ‘Dialog’ class. It is implemented using an Android ‘DialogFragment’, enabling the display of modal dialogs such as the AI configuration menu and providing a foundation for other Ludii dialogs not yet integrated into the Android version. The differing lifecycle and API of ‘DialogFragment’ necessitated adjustments in the calling code during dialog creation and invocation.

For the ‘swing’ package, over 50 classes were ported. We highlight the most critical ones. ‘JPanel’ was implemented by extending Android’s ‘LinearLayout’ (‘android.widget.LinearLayout’). It serves as the primary container for on-screen elements, structuring them via a layout manager. Its structure is shown in Figure 4.5. ‘JFrame’ also extends ‘LinearLayout’ and provides the basic window structure, dividing the interface into three conceptual areas: a menu bar (‘JMenuBar’), a content panel (‘JPanel’), and an optional default button (‘JButton’)—the latter not being used in the main screen. Many features were mimicked or stubbed with empty implementations where necessary (e.g., ‘Icon’ support). The structure of ‘JFrame’ is depicted in Figure 4.6.

```

public class JPanel extends LinearLayout implements ViewComponent { 10 usages

    public static final int LEFT_ALIGNMENT = 0; 3 usages
    public static final int CENTER_ALIGNMENT = 1; 7 usages
    public static final int RIGHT_ALIGNMENT = 2; 4 usages
    public static final int TOP_ALIGNMENT = 4; 4 usages
    public static final int BOTTOM_ALIGNMENT = 8; 4 usages

    public static final int TOP_LEFT = TOP_ALIGNMENT | LEFT_ALIGNMENT; 1 usage
    public static final int TOP_CENTER = TOP_ALIGNMENT | CENTER_ALIGNMENT; no usages
    public static final int TOP_RIGHT = TOP_ALIGNMENT | RIGHT_ALIGNMENT; no usages
    public static final int CENTER_LEFT = CENTER_ALIGNMENT | LEFT_ALIGNMENT; no usages
    public static final int CENTER_CENTER = CENTER_ALIGNMENT | CENTER_ALIGNMENT; no usages
    public static final int CENTER_RIGHT = CENTER_ALIGNMENT | RIGHT_ALIGNMENT; no usages
    public static final int BOTTOM_LEFT = BOTTOM_ALIGNMENT | LEFT_ALIGNMENT; no usages
    public static final int BOTTOM_CENTER = BOTTOM_ALIGNMENT | CENTER_ALIGNMENT; no usages
    public static final int BOTTOM_RIGHT = BOTTOM_ALIGNMENT | RIGHT_ALIGNMENT; no usages

    private Paint paint; 3 usages
    protected Graphics graphics;
    private Border border; 2 usages
    private Layout layoutManager; 8 usages
    private int alignment = TOP_LEFT; 7 usages
    private final List<FocusListener> focusListeners = new ArrayList<>(); 4 usages
    private final List<HierarchyListener> hierarchyListeners = new ArrayList<>(); 4 usages
    private final List<MouseListener> mouseListeners = new ArrayList<>(); 3 usages
    private final List<MouseMotionListener> mouseMotionListeners = new ArrayList<>(); 3 usages
    private final List<KeyListener> keyListeners = new ArrayList<>(); 4 usages
    private final JPanel panel; 7 usages

    public JPanel() {}
    public JPanel(Attributeset set) {}
    public JPanel(Layout layout) {}
    public JPanel(JTree tree) {}
    private void init() {}
    public void add(View comp) { add(comp, constraints, null); // appelle la méthode avec contraintes; }
    public void add(JSeparator comp) {}
    public void add(View comp, Object constraints) {}
    public void remove(View child) { removeView(child); }
    public void removeAll() { removeAllViews(); }
    public void setOnClickListener(String text) { setContentDescription(text); }
    public void paint(Graphics g) { dispatchDraw(g.getCanvas()); }
    public void setLayout(Layout layout) {}
    public void setLayout() {}
    public void repaint() { invalidate(); }
    public Graphics getGraphics() {}
    public void setPreferredSize(int w, int h) {}
    public void setGraphics(Graphics graphics) { this.graphics = graphics; }
    public void repaint(Rectangle rectangle) { invalidate(); }
    public void revalidate() { invalidate(); }
    public void setPreferredSize(Dimension dimension) {}
    public void setBorder(Border emptyborder) {}
    public void setBorder(Border border) {}
    protected void paintComponent(Graphics g) {}
    public void setVisible(boolean visible) {}

    public void setLocation(int x, int y) {}
    public void setLocation(Point point) { setLocation(point.x, point.y); }
    public Point getLocation() { return new Point(getX(), getY()); }
    public Dimension getPreferredSize() {}
    public Dimension getMinimumSize() {}
    public void setMinimumSize(Dimension dimension) {}
    public void setMaximumSize(int w, int h) {}
    public void setSize(Dimension dimension) { setSize(dimension.width, dimension.height); }
    public void setSize(int width, int height) {}
    public void setBackground(Color color) {}
    public void setForeground(Color color) {}
    public boolean isVisible() { return getVisibility() == View.VISIBLE; }
    public void setBounds(int x, int y, int width, int height) {}
    public Rectangle getBounds() {}
    public void addHierarchyListener(HierarchyListener listener) {}
    public void addFocusListener(FocusListener listener) {}
    public void addMouseListener(MouseListener listener) {}
    public void addMouseMotionListener(MouseMotionListener listener) {}
    public void addKeyListener(KeyListener listener) {}
    public void scrollToRectToVisible(Rectangle rect) {}
    @Override
    public void setFont(Font font) {}
    @Override
    public Font getFont() { return null; }
    @Override
    public Dimension getSize() { return new Dimension(getWidth(), getHeight()); }
    public void setDoubleBuffered(boolean doubleBuffered) {}
    public void setAlignment(int align) {}
    public void setAlignment(int horizontal) {}
    public void setAlignment(int vertical) {}
    private void updateLayoutGravity() {}
    @Override
    protected void onLayout(boolean changes, int l, int t, int r, int b) {}
    @Override
    protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {}
}

```

Figure 4.5: Structure of JPanel emulation class. Source: [1]

```

public class JFrame extends LinearLayout implements RootPanel { 4 usages

    protected String title;
    private int closeOperation = WindowConstants.DO_NOTHING_ON_CLOSE; 1 usage
    private WindowAdapter windowListener; 3 usages
    private JPanel contentPane; 10 usages
    private JMenuBar menuBar; 3 usages
    protected JButton defaultButton; 4 usages
    private boolean focusTraversalKeysEnabled = true; 1 usage
    private Layout layout; 3 usages
    protected JFrame frame = this;

    public JFrame(String title) {}
    public JFrame() { this(""); }
    private void init() {}
    public void setTitle(String title) { this.title = title; }
    public String getTitle() { return title; }
    public void setView(Activity activity) {}
    public void setContentView(Object constraints) { add(view); }
    public void add(View view) {}
    public void remove(View view) { if(contentPane != null) contentPane.removeView(view); }
    public void setContentPane(JPanel panel) {}
    public void setMenuBar(JMenuBar mainMenu) {}
    private void updateContentPaneLayout() {}
    public void setDefaultButton(Button button) {}
    public void setVisible(boolean visible) {}
    public void setLocationRelativeTo(Object object) {}
    public void setIconImage(BufferedImage bufferedImage) {}
    public void setIcon(Bitmap bitmap) {}
    public void setIcon(int width, int height) {}
    public void setSize(Dimension dimension) {}
    public void setDefaultCloseOperation(int operation) { this.closeOperation = operation; }
    public void setFocusTraversalKeysEnabled(boolean enabled) {}
    public void addWindowListener(WindowAdapter windowAdapter) {}
    public void triggerWindowClosingEvent() {}
    public void setOrientation(int orientation) { super.setOrientation(orientation); }
    public void setGravity(int gravity) { super.setGravity(gravity); }
    public void setBackgroundColor(Color color) { setBackgroundColor(color.toArgb()); }
    public void setPadding(int left, int top, int right, int bottom) {}
    public void close() {}
    public void dispose() {}
    private void cleanupResources() {}

    public void setMinimumSize(Dimension dimension) {}
    public JMenuBar getMenuBar() { return menuBar; }
    public JPanel getContentPane() { return contentPane; }
    public JButton getDefaultButton() { return defaultButton; }
    public void setResizable(boolean b) {}
    public void setLocation(int l, int t) {}
    public RootPanel getContentPane() { return this; }
    public void dispatchEvent(WindowEvent windowEvent) {}
    public void revalidate() { requestLayout(); }
    public void repaint() {}
    public void setLayout(Layout BorderLayout) {}
    @Override
    protected void onLayout(boolean changes, int l, int t, int r, int b) {}
    @Override
    protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {}
}

```

Figure 4.6: Structure of JFrame emulation class. Source: [1]

Other important elements include classes from the 'swing.tree' package, such as 'JTree'—the main class for representing tree structures. This was implemented using an external library, 'com.github.bmelnychuk:atv' [3], which provides an Android 'TreeView'. Although this library is deprecated, it facilitated a more straightforward emulation. An alternative would be to use Android's 'ExpandableListView', but this would require significant modifications and is limited to two levels. Key menu-related classes such as 'JMenuBar', 'JMenu', 'JMenuItem', 'JMenuButton', and 'JPopupMenu' were also adapted to function analogously to their Swing counterparts within the Android environment. 'JMenuBar' extends 'LinearLayout' and contains a horizontal list of 'JMenu' items, each housed in a 'JMenuButton'. As with AWT, certain Swing-specific events and listeners, such as 'ListSelectionEvent' and 'ListSelectionListener', required emulation.

4.3 ADAPTATION IN ORIGINAL CODE

Despite the use of emulation classes, a portion of the original Ludii source code still required adaptation to function within the Android environment.

4

The most significant adaptation involved the application's entry point. The original StartDesktopApp class was fundamentally redesigned into StartAndroidApp. While the desktop version simply instantiates a DesktopApp and stores it in a static variable, the Android version is more complex. It extends AppCompatActivity, and the initialization logic is moved from a main method into the onCreate lifecycle callback. Crucially, this class initializes and provides global access to the Android Context, which is essential for all Android UI components and system services to function correctly. A comparison of the two structures is shown in Figure 4.8 and 4.7. In these, you can see that the android one is more complex.

```
package app;

/**
 * Main point of Entry for running the Ludii application.
 *
 * @author Matthew.Stephenson and Dennis Soemers
 */
public class StartDesktopApp
{
    private static DesktopApp desktopApp = null;

    public static void main(final String[] args)
    {
        // The actual launching
        if (args.length == 0)
        {
            desktopApp = new DesktopApp();
            desktopApp.createDesktopApp();
        }
        else
        {
            PlayerCLI.runCommand(args);
        }
    }

    // Used in case any agents need DesktopApp functions.
    public static DesktopApp desktopApp() { return desktopApp; }
```

Figure 4.7: Ludii StartAndroidApp class. Source: [11]

```

package playerAndroid.app;

import android.content.pm.ActivityInfo;
import android.os.Bundle;
import android.content.Context;
import androidx.appcompat.app.AppCompatActivity;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.Files;
import androidUtils.ZipManager;
import androidUtils.awt.MouseInfo;
import androidUtils.swing.JFrame;
import manager.Manager;
import manager.ai.AIDetails;

public class StartAndroidApp extends AppCompatActivity {

    private static StartAndroidApp startAndroidApp; 2 usages
    private static AndroidApp androidApp; 4 usages
    private static Context appContext; 3 usages
    public static String AppName = "Ludii"; 1 usage
    protected static JFrame frame;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
        setupInitialVariable();

        androidApp.createAndroidApp();
        frame = AndroidApp.frame();
        frame.setView(StartAndroidApp.startAndroidApp());
        frame.setTitle(AppName);

        frame.setVisible(true);
        MouseInfo.setupTouchTracking(this);
    }

    public void setupInitialVariable() 1 usage
    {
        startAndroidApp = this;
        androidApp = new AndroidApp();
        appContext = this;
        ZipManager.extractSvgs( type: ".svg");
    }
}

```

Figure 4.8: Ludii Android StartAndroidApp class. Source: [1]

To optimize the application package (APK) size, all SVG assets (such as game pieces) are stored within a compressed archive that is extracted upon the application's first launch, making them accessible to the rendering system. A similar mechanism could be implemented for .lud game description files. For rendering, the drawBoard and drawGraph methods in the AndroidApp class (adapted from DesktopApp) required modification. Specifically, the code that converts an SVG string into a displayable image could no longer use the original SVGUtils class. Instead, it utilizes the emulated SVGGraphics2D.getBufferedImageFrom64() static method, which processes a Base64-encoded representation. The touch input system,

emulating `MouseInfo`, was also integrated here to track screen touches and manage click positions required for game interactions.

Instances where the original code used `java.util.Timer.schedule()` with `TimerTask` were replaced with Android's `Handler` and `Looper` mechanisms. This change ensures better compatibility with the Android UI thread model and provides more reliable timing control.

Significant simplifications were applied to the UI views to suit the mobile form factor. In `TabView`, most desktop-oriented panels (e.g., `Status`, `Moves`, `Turns`) were removed, leaving only the `Rules` panel. This panel was further adapted to render hyperlinks correctly, and its title was dynamically set to "Rules of [Game Name]" to provide context, as the standard window title bar is absent on Android. Minor adjustments to coordinates and scaling ratios were made in `BoardView`, `PlayerView`, and `TabView` to ensure all interface elements fit within the screen bounds and to enable proper scrolling for text content.

The different file system architecture on Android necessitated changes to file handling classes (e.g., `FileHandling`). Path references were updated to use Android's `AssetManager` for accessing files bundled in the assets directory. Furthermore, the modal dialog system used on the desktop (which blocks execution until user input) is not standard on Android. This required refactoring asynchronous operations like `loadGameFromFile` and `loadGameFromMemory`. The logic following a user's file selection was moved into callback methods invoked by the dialog. The modifications to `GameLoading` and `GameLoaderDialog` are highlighted in Figures 4.9, 4.10, 4.11 and 4.12.

```
public static void loadGameFromMemory(final PlayerApp app, final boolean debug) 3 usages
{
    GameLoading.debug = debug;
    GameLoading.app = app;
    final String[] choices = FileHandling.listGames();

    String initialChoice = choices[0];
    for (final String choice : choices)
    {
        if (app.manager().savedLudName() != null &&
            app.manager().savedLudName().endsWith(choice.replaceAll(Pattern.quote("\\\\"), replacement: "/")))
        {
            initialChoice = choice;
            break;
        }
    }
    final String choice = GameLoaderDialog.showDialog(AndroidApp.frame(), choices, initialChoice);
}
```

Figure 4.9: Ludii Android `LoadGameFromMemory` class. Source: [1]

```

tree.addMouseListener(new MouseAdapter()
{
    @Override
    public void mouseClicked(final MouseEvent e)
    {
        final TreePath path = tree.getPathForLocation(e.getComponent());
        if (path != null)
        {
            final GameLoaderNode node = (GameLoaderNode) path.getLastPathComponent();
            System.out.println(node.fullName);
            if (node.fullName.endsWith(".lud"))
            {
                pane.setValue(Integer.valueOf(JOptionPane.OK_OPTION));
                dialog.dispose();

                // TODO if we want to preserve per-game last-selected-options in preferences, load them here
                GameLoading.app.manager().settingsManager().userSelections().setRuleset(Constants.UNDEFINED);
                GameLoading.app.manager().settingsManager().userSelections().setSelectOptionStrings(new ArrayList<String>());
                GameLoading.loadGameFromMemory(GameLoading.app, node.fullName, GameLoading.debug);
            }
        }
    }
});

```

4

Figure 4.10: Ludii Android excerpt from the GameLoaderDialog class. Source: [1]

```

public static void loadGameFromMemory(final PlayerApp app, final boolean debug)
{
    final String[] choices = FileHandling.listGames();

    String initialChoice = choices[0];
    for (final String choice : choices)
    {
        if (app.manager().savedLudName() != null && app.manager().savedLudName().endsWith(choice.replaceAll(Pattern.quote("\\\\"), "/")))
        {
            initialChoice = choice;
            break;
        }
    }
    final String choice = GameLoaderDialog.showDialog(DesktopApp.frame(), choices, initialChoice);

    if (choice != null)
    {
        // TODO if we want to preserve per-game last-selected-options in preferences, load them here
        app.manager().settingsManager().userSelections().setRuleset(Constants.UNDEFINED);
        app.manager().settingsManager().userSelections().setSelectOptionStrings(new ArrayList<String>());
        loadGameFromMemory(app, choice, debug);
    }
}

```

Figure 4.11: Ludii LoadGameFromMemory class. Source: [11]

```
tree.addMouseListener(mouseClicked(e) -> {  
    if (e.getClickCount() == 2)  
    {  
        final TreePath path = tree.getPathForLocation(e.getX(), e.getY());  
        if (path != null)  
        {  
            final GameLoaderNode node = (GameLoaderNode) path.getLastPathComponent();  
            if (node.fullName.endsWith(".lud"))  
            {  
                pane.setValue(Integer.valueOf(JOptionPane.OK_OPTION));  
                dialog.dispose();  
            }  
        }  
    }  
});
```

Figure 4.12: Ludii excerpt from the GameLoaderDialog class. Source: [11]

4

The main menu was reduced to four core functionalities: AI (toggle), Load Game, Restart, and Play/Pause. The "Preferences" dialog was renamed to "AI" and stripped of most configuration options, currently retaining only the ability to enable or disable AI for each player. The underlying SettingsDialog class was heavily simplified, while a legacy SettingsDialogOld class preserves the adapted code for the full set of desktop preferences (e.g., thinking time, AI selection) for potential future use.

5

5

RESULT

This methodology and implementation resulted in an application that is easily maintainable in parallel with the desktop version. Indeed, the approach required minimal direct modification of the original source code, primarily limited to changes in import statements. Consequently, in most cases, updating the Android application would only require replacing the codebase with the new version from the desktop project. However, in some instances, further adaptation might be necessary, such as extending the `androidUtils` package if an update introduces new uses of `awt`, `swing`, or other Android-incompatible dependencies (like the Batik library discussed earlier).

Regarding the user interface, maximum maintainability was sought by keeping a structure similar to the desktop version. The only modifications were a simplification of the main menu and the removal of certain panels from the rules tab page. This similarity is clearly visible in Figures 2.5 and 5.1. The board occupies the left side of the screen, taking up a square area sized to the phone's height minus the menu bar. The player panel is situated to its right, filling the remaining width, with the rules panel directly below it. The rules panel, which can contain variable amounts of text, is scrollable.



Figure 5.1: Ludii Android interface.

5

Furthermore, the menu located in the top-left corner of the screen is a streamlined version of its desktop counterpart. We retained only the options deemed essential for the Android version, totaling four items: "AI" (replacing the "Preferences" item found under the "Ludii" menu on desktop), "Load Game" (functioning like its namesake under the "File" menu, allowing the user to select a game included in the APK), "Restart" (to reset the current game, equivalent to the item under the "Game" menu), and "Play/Pause" (to start or stop the game, enabling AI players to move if activated). A game begins in a paused state and must be set to "Play" for any enabled AIs to take their turns. This menu is shown in Figure 5.1. You can also see the IA menu in the Figure 5.2 and the Load Game menu in the Figure 5.3.

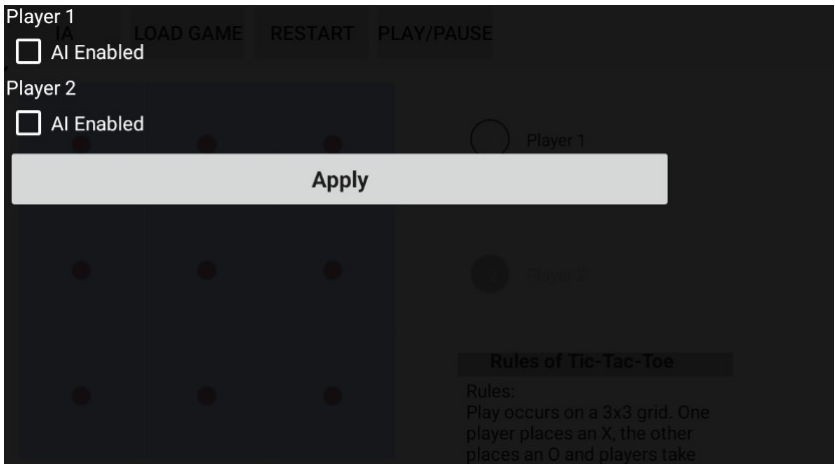


Figure 5.2: Ludii Android IA menu.

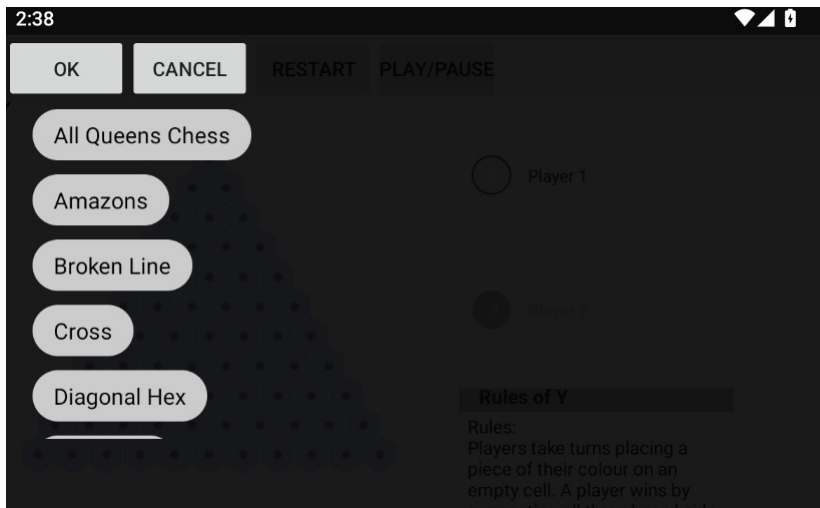


Figure 5.3: Ludii Android Load Game Dialog.

Unfortunately, certain complex ludemes, such as those using vertices or containing the ":" symbol, are not functional in the port. This may be explained by the use of reflection during the compilation of ludii code. Reflection is more limited on Android compared to standard Java, partly due to build tools that remove unused classes. The issue could also stem from other sources, which will be discussed in the next chapter.

Finally, we have succeeded in making only 15 of the games available in the desktop application functional on Android. Others may work, but testing primarily focused on games in the "Board" folder, as these were deemed more likely to function given the problems with complex ludemes (like those using vertices), which are more common in non-board games. The following games were tested and are functional: All Queens Chess, Amazons, Broken Line, Cross, Diagonal Hex, Havannah, Hex, Product, Squava, T'uk T'uk, Tic-Tac-Mo, Tic-Tac-Toe, Tic-Tac-Toe Misere, Y, and Yavalath.

6

FURTHER WORK

6

This chapter outlines potential improvements and challenges aimed at enhancing the application's functionality and user experience to attract a wider audience. The analysis is based on application testing and a user survey, the results of which are available in Appendix B. Some improvements highlighted in the survey, such as the menu redesign and the correction of disappearing pieces in games like All Queens Chess, have already been implemented.

6.1 FUNCTIONAL GAMES

The previous chapter highlighted the limited number of currently functional games (15 games). A more in-depth analysis of the Ludii compiler and its adaptation for Android could drastically increase this number. As mentioned, a key issue may be the use of Java reflection for dynamic ludeme instantiation, which is less performant and more restricted on Android. Refactoring this mechanism using design patterns like the Factory Pattern could provide similar dynamic object creation while being more optimized for the mobile environment. Additionally, a more thorough analysis of the codebase could identify other discrepancies between the desktop and Android execution environments that currently disrupt gameplay.

For such an analysis, it would be necessary to read the code and examine the dependencies used, as well as the way the different parts of the compiler are implemented, in order to compare them with what is possible on Android. A systematic code review appears to be the most feasible approach, given that even though Java is used on Android, some operations function slightly differently, and identifying these sections of code requires careful inspection. However, this approach could be quite time-consuming.

Regarding technical avenues beyond the Factory Pattern, another possibility is the use of code generation techniques, such as Annotation Processing.

6.2 USER INTERFACE

While the current interface maintains the desktop version's simplicity, it may not be fully optimized for mobile. For instance, the victory condition is not visually prominent; it is indicated only by a subtle color change of the winning player's name from black to red. This can be easily missed, especially in games like Amazons where no winning line is highlighted. Enhancements could include a blinking animation or a modal dialog announcing the winner and prompting a restart. Furthermore, the game board cells can become very small in games with large boards like Yavalath, Hex, or Havannah, making precise touch input difficult. Implementing a zoom functionality, as requested by 9 out of 10 survey respondents, would significantly improve usability. The placement of the rules panel in the bottom right is also suboptimal. A potential solution is to replace the static panel with a "Rules" button that opens a centered, scrollable dialog, improving visibility and screen real estate. Additionally, making hyperlinks within the rules (e.g., to sources) interactive would enhance the user experience. The AI configuration menu could be expanded to allow users to select the type of AI algorithm (e.g., Alpha-Beta, Monte-Carlo Tree Search) and adjust thinking time, which directly influences difficulty. Finally, an "undo" feature would be valuable, allowing players to explore different moves from a given game state.

6.3 AUDIO AND MOTION FEEDBACK

Integrating sound effects would greatly enhance immersion and user experience on the Android platform. Ambient background music, sound cues for piece placement/movement, and a distinctive victory fanfare would make the application more engaging. Audio feedback, particularly for signaling victory, would also complement possible visual enhancements, making the outcome of a game unmistakable. Well-implemented, context-appropriate sounds can significantly improve user retention by creating a more polished and enjoyable experience, thereby encouraging longer sessions and generating more gameplay data. Conversely, poorly chosen or timed audio can have a negative impact, underscoring the importance of careful design in this area.

On the implementation side, MediaPlayer could be used to play audio from a file, providing control over starting, stopping, restarting, and pausing playback. Another option, or a complementary one, is SoundPool, which is designed for playing small audio files with low latency, using its play function.

In line with the addition of sound, implementing haptic feedback (vibration) for board interactions could further enhance interactivity. According to a study on haptic design [23], key elements influencing user experience include motion properties and the "weight factor."

Motion properties primarily affect visceral (immediate physical sensation) and reflective (interpretation, elegance) affective qualities, but less so the behavioral level (perceived ease of use). The most influential property is response duration, followed by acceleration and bounce. A short response duration is associated with qualities such as light, hard, clicky, simple, and clear, whereas a long duration is linked to heavy, soft, fluid, complicated, and

profound.

The weight factor, on the other hand, influences a broader range of affective qualities much more strongly than manipulating motion properties alone. For instance, requiring pressure makes an interface feel heavier, harder, tenser, etc. User preference for this functionality depends on the context and interface type. It is generally well-received in simple interfaces like buttons but less so in complex interactions like rotary switches. In the case of Ludii, which typically involves simple tap actions, integrating subtle, context-appropriate haptic feedback should increase interactivity by positively influencing these affective qualities.

7

CONCLUSION

This thesis successfully addressed the significant challenge of porting the complex, desktop-oriented Ludii system to the Android mobile platform. The primary research problem—the fundamental incompatibility between Ludii’s Java AWT/Swing dependencies and Android’s UI framework—was solved through the design and implementation of a bespoke emulation layer. By creating Android-based counterparts for core classes like Graphics2D, JPanel, JFrame, and handling SVG rendering via an adapted Batik library, this project enabled Ludii’s core logic to run with minimal direct source code alteration. The parallel architecture of the StartAndroidApp and strategic code adaptations for file handling, dialogs, and threading further cemented a maintainable codebase that can evolve alongside the main Ludii project.

The result is a functional Android application that preserves the core Ludii experience for a subset of 15 strategic board games, proving the technical viability of the chosen approach. The user interface maintains the desktop version’s logical layout while being pragmatically streamlined for touchscreens, as evidenced by the simplified four-option menu.

However, this work also clearly delineates the project’s current boundaries. The most notable limitation is the restricted library of functional games, stemming from issues with complex ludemes and the constrained use of Java reflection on Android—a key mechanism in Ludii’s game compilation. Furthermore, the user interface, while functional, offers clear avenues for enhancement to meet mobile usability standards.

Therefore, this project does not represent a final product but a foundational and strategic port. It provides a robust, maintainable codebase that resolves the core architectural clash. The future work outlined—ranging from deep technical refactoring (e.g., replacing reflection with factory patterns) to user-centric features (zoom/pan, victory dialogs, configurable AI)—charts a clear path forward. Ultimately, this thesis makes a tangible contribution to the field of General Game Playing by taking a crucial step toward mobile accessibility. It lays the groundwork for transforming Ludii from a desktop research tool into a platform capable of reaching a vast user base, thereby unlocking new potentials for human-AI interaction, large-scale gameplay data collection, and the democratization of digital board game heritage.

BIBLIOGRAPHY

- [1] Alexandre de Salle. *Ludii Android: The Ludeme General Game System (GitHub repository)*. <https://github.com/adesalle/LudiiAndroid/tree/master/PlayerAndroid>. Dernière consultation: 20 Décembre 2025.
- [2] Mohammad Aliannejadi et al. “The Impact of User Demographics and Task Types on Cross-App Mobile Search”. In: *Flexible Query Answering Systems*. Ed. by Troels Andreasen et al. Cham: Springer International Publishing, 2021, pp. 223–234. ISBN: 978-3-030-86967-0. doi: 10.1007/978-3-030-86967-0_17.
- [3] Bogdan Melnychuk. *AndroidTreeView*. <https://github.com/bmelnychuk/AndroidTreeView>. Dernière consultation: 20 Décembre 2025.
- [4] Cameron Browne et al. *Ludii: General Game System*. <https://ludii.games/>.
- [5] Matthew Stephenson Cameron Browne Éric Piette and Dennis J. N. J. Soemers. “Ludii General Game System for Modeling, Analyzing, and Designing Board Games”. In: *Encyclopedia of Computer Graphics and Games*. Ed. by Newton Lee. Cham: Springer International Publishing, 2023, pp. 1082–1095.
- [6] Robert Canaan et al. “Leveling the Playing Field: Fairness in AI Versus Human Game Benchmarks”. In: *Proceedings of the 14th International Conference on the Foundations of Digital Games (FDG ’19)*. ACM, 2019. doi: 10.1145/3337722.3341841.
- [7] Tristan Cazenave et al. *Polygames: Improved Zero Learning*. <https://github.com/facebookarchive/Polygames>. Zero-learning framework for deep reinforcement learning in games. 2020.
- [8] COST Action CA22145. *Computational Techniques for Tabletop Games Heritage (GameTable)*. <https://gametable.network/>. EU-funded interdisciplinary research network on game AI and cultural heritage. Approved 12/05/2023, started 24/10/2023. 2023–2027.
- [9] Matthew Stephenson Dennis J. N. J. Soemers Eric Piette and Cameron Browne. *Ludii User Guide*. <https://ludii.games>. Accessed from September 2024 to July 2025. 2019.
- [10] Walter Crist Dennis J.N.J. Soemers and Cameron Browne. “Report on the Digital Ludeme Project”. In: *ICGA Journal* 41.3 (2019), pp. 138–142.
- [11] Digital Ludeme Project. *Ludii: The Ludeme General Game System (GitHub repository)*. <https://github.com/Ludeme/Ludii>. Dernière consultation: 20 Décembre 2025.
- [12] Cameron Browne Eric Piette and Dennis J. N. J. Soemers. *Ludii Game Logic Guide*. arXiv:2101.02120. 2021. arXiv: 2101.02120 [cs.AI]. URL: <https://arxiv.org/abs/2101.02120>.

- [13] Dennis J. N. J. Soemers Eric Piette Matthew Stephenson et al. “Ludii – The Ludemic General Game System”. In: *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI)*. 2020.
- [14] Michael Genesereth and Yngvi Björnsson. “The International General Game Playing Competition”. In: *AI Magazine* 34.2 (2013), pp. 107–107.
- [15] Michael Genesereth and Michael Thielscher. *General Game Playing*. Morgan & Claypool Publishers, 2014. ISBN: 978-1-60845-909-0.
- [16] Google. *Runtime Android*. Documentation technique. Google. 2024. URL: <https://source.android.com/docs/core/runtime> (visited on 11/15/2024).
- [17] Google Inc. *Android SDK API Reference*. Accessed from September 2024 to July 2025. 2024–2025. URL: <https://developer.android.com/reference>.
- [18] Google LLC. *Android Asset Packaging*. 2023. URL: <https://developer.android.com/>.
- [19] Jakub Kowalski et al. “Regular Boardgames”. In: *AAAI Conference on Artificial Intelligence (AAAI ’19)*. 2019, pp. 1699–1706. doi: 10.1609/aaai.v33i01.33011699.
- [20] Marc Lanctot et al. *OpenSpiel: A Framework for Reinforcement Learning in Games*. https://github.com/deepmind/open_spiel. Reinforcement learning framework for games. 2020.
- [21] Oracle Corporation. *Java AWT API Documentation*. Accessed from September 2024 to July 2025. 2024–2025. URL: <https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/java/awt/package-summary.html>.
- [22] Oracle Corporation. *Java Swing API Documentation*. Accessed from September 2024 to July 2025. 2024–2025. URL: <https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/javax/swing/package-summary.html>.
- [23] Doyun Park, Ji-Hyun Lee, and Sangtae Kim. “Investigating the affective quality of interactivity by motion feedback in mobile touchscreen user interfaces”. In: *International Journal of Human-Computer Studies* 69.12 (2011), pp. 839–853. ISSN: 1071-5819. doi: <https://doi.org/10.1016/j.ijhcs.2011.06.006>. URL: <https://www.sciencedirect.com/science/article/pii/S1071581911000784>.
- [24] Éric Piette et al. “GameTable COST Action: Kickoff Report”. In: *ICGA Journal* 46.1 (2024). Kickoff held at Leiden University, January 29–30, 2024. doi: 10.3233/ICG-240245.
- [25] Alois Rautureau and Éric Piette. “CogniPlay: A work-in-progress Human-like model for General Game Playing”. In: *arXiv preprint arXiv:2507.05868* (2025). 5 pages, 1 figure. URL: <https://arxiv.org/abs/2507.05868>.
- [26] Sam Schreiber. *Games-base repository*. <http://games.ggp.org/base/>. Accessed: May 2025. Stanford University, 2016.

-
- [27] Zillions Development Corporation. *Zillions of Games*. <https://www.zillionsofgames.com>. Consulted in May 2025. Zillions Development Corporation, 1998.

APPENDIX

A : GDL AND RBG IMPLÉMENTATION OF TIC TAC TOE

```

1 (role xplayer) (role oplayer)
2 (index 1) (index 2) (index 3)
3 (<= (base (cell ?x ?y b)) (index ?x) (index ?y))
4 (<= (base (cell ?x ?y x)) (index ?x) (index ?y))
5 (<= (base (cell ?x ?y o)) (index ?x) (index ?y))
6 (<= (base (control ?p)) (role ?p))
7 (<= (input ?p (mark ?x ?y)) (index ?x) (index ?y) (role ?p))
8 (<= (input ?p noop) (role ?p))
9 (init (cell 1 1 b)) (init (cell 1 2 b)) (init (cell 1 3 b))
10 (init (cell 2 1 b)) (init (cell 2 2 b)) (init (cell 2 3 b)) (init (cell 3 1 b))
11 (init (cell 3 2 b)) (init (cell 3 3 b))
12 (init (control xplayer))
13 (<= (next (cell ?m ?n x)) (does xplayer (mark ?m ?n)) (true (cell ?m ?n b)))
14 (<= (next (cell ?m ?n o)) (does oplayer (mark ?m ?n)) (true (cell ?m ?n b)))
15 (<= (next (cell ?m ?n ?w)) (true (cell ?m ?n ?w)) (distinct ?w b))
16 (<= (next (cell ?m ?n b)) (does ?w (mark ?j ?k)) (true (cell ?m ?n b)) (or (distinct ?m ?j) (distinct ?n ?k)))
17 (<= (next (control xplayer)) (true (control oplayer)))
18 (<= (next (control oplayer)) (true (control xplayer)))
19 (<= (row ?m ?x) (true (cell ?m 1 ?x)) (true (cell ?m 2 ?x)) (true (cell ?m 3 ?x)))
20 (<= (column ?n ?x) (true (cell 1 ?n ?x)) (true (cell 2 ?n ?x)) (true (cell 3 ?n ?x)))
21 (<= (diagonal ?x) (true (cell 1 1 ?x)) (true (cell 2 2 ?x)) (true (cell 3 3 ?x)))
22 (<= (diagonal ?x) (true (cell 1 3 ?x)) (true (cell 2 2 ?x)) (true (cell 3 1 ?x)))
23 (<= (line ?x) (row ?m ?x))
24 (<= (line ?x) (column ?m ?x))
25 (<= (line ?x) (diagonal ?x))
26 (<= open (true (cell ?m ?n b)))
27 (<= (legal ?w (mark ?x ?y)) (true (cell ?x ?y b)) (true (control ?w)))
28 (<= (legal xplayer noop) (true (control oplayer)))
29 (<= (legal oplayer noop) (true (control xplayer)))
30 (<= (goal xplayer 100) (line x))
31 (<= (goal xplayer 50) (not (line x)) (not (line o)) (not open))
32 (<= (goal xplayer 0) (line o))
33 (<= (goal oplayer 100) (line o))
34 (<= (goal oplayer 50) (not (line x)) (not (line o)) (not open))
35 (<= (goal oplayer 0) (line x))
36 (<= terminal (line x))
37 (<= terminal (line o))
38 (<= terminal (not open))

```

Figure A.1: GDL Implementation of Tic Tac Toe. Source: games.ggp.org

```

1 #players = xplayer(100), oplayer(100)
2 #pieces = e, x, o
3 #variables =
4 #board = rectangle(up,down,left,right,
5   [e, e, e]
6   [e, e, e]
7   [e, e, e])
8
9 #anySquare = ((up + down)(left + right))
10
11 #winAtDir(dir; oppDir; piece) = (dir {piece} (dir + (oppDir)^2) {piece} + (oppDir {piece})^2)
12 #winAt(piece) = (
13   winAtDir(up left; down right; piece)
14   + winAtDir(up; down; piece)
15   + winAtDir(up right; down left; piece)
16   + winAtDir(left; right; piece)
17 )
18
19 #turn(me; opp) =
20   ->me-player anySquare {e} ->> [me]
21   (
22     {! winAt(me)}
23     + {? winAt(me)} [$ me-player=100] [$ opp-player=0] ->> {}
24   )
25
26 #rules = [$ xplayer=50] [$ oplayer=50] (
27   turn(x; o)
28   turn(o; x)
29 )

```

Figure A.2: RBG Implementation of Tic Tac Toe. Source: [github.marekesz.rbgGames](https://github.com/marekesz.rbgGames)

B : LUDII ANDROID SURVEY

04/01/2026 16:36

Porting Ludii to Android

Porting Ludii to Android

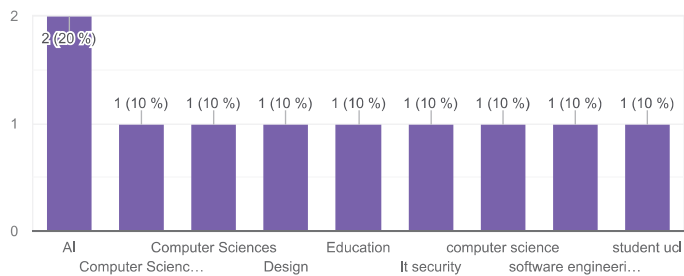
10 réponses

[Publier les données analytiques](#)

In which field do you work?

 Copier

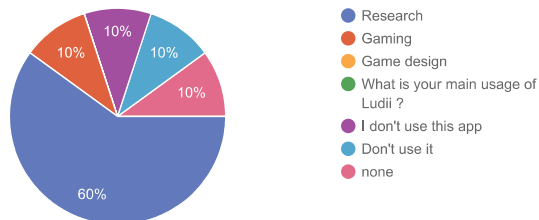
10 réponses



What is your main usage of Ludii ?

 Copier

10 réponses



Ludii Android: Introduction

<https://docs.google.com/forms/d/1qHaXd5u2jibCRgvm4qJOcIsi7FMjmBTSipx838CHkHw/viewanalytics>

1/9

Figure B.1: Survey page 1

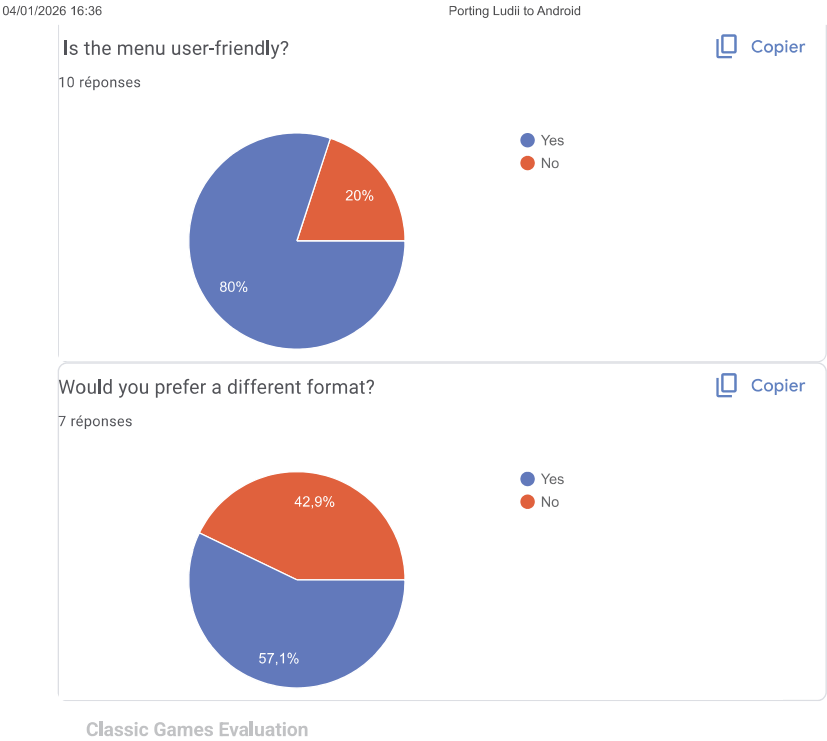


Figure B.2: Survey page 2

04/01/2026 16:36

Porting Ludii to Android



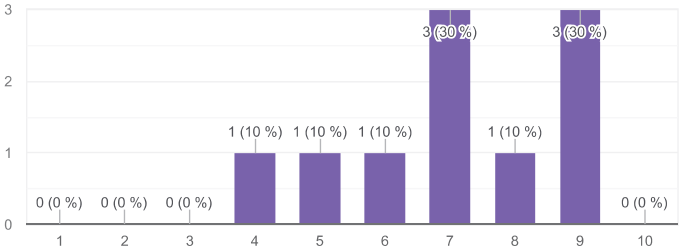
Now, try the following **mandatory** games:

Tic-Tac-Toe, Amazons, All-Queens-Chess

You are **also** welcome to try any other games you are interested in, but the three listed above are required.

How would you rate the enjoyability and usability of the Android version of Ludii?

10 réponses



<https://docs.google.com/forms/d/1qHaXd5u2jybCRgvm4qJOCsi7FMjmBTSipx838CHkHw/viewanalytics>

3/9

Figure B.3: Survey page 3

04/01/2026 16:36

Porting Ludii to Android

Why?

8 réponses

The rules are placed very uncomfortably in the corner of the screen. The "whose turn is it" section takes nearly 2/3 of the screen and on bigger boards (like in amazons) each grid is too small. Overall the system works fine but it makes for a very stale presentation.

All seems to work really well on my phone.

It's mostly ok, but I have a small phone and big fingers. Not being able to zoom in is annoying and make playing the bigger boards hard to play (Especially when half the screen isn't even used when playing)

It works nicely but we can't see who is the winner. For the rest it works well.

UI is simple; game library is already good; just one minor detail -> when clicking on AI, the 'AI Enabled' is black on a grey background, I wasn't seeing the text in the first 5 seconds

use like original

The game is not displayed in full screen (there is empty space on the right side of the screen) and it is always in horizontal mode.

There are several bugs in All-Queens-Chess. When I click on a piece, the possible moves are shown, but all the other pieces disappear, which makes the game difficult to play.

At the end of the game, there is no indication of who wins and no option to restart the game.

The Amazons game has the same issue with the pieces.

The games are playable, the AI is at an acceptable level (it would be interesting to have different AI levels), but I see one major flaw: once the game is over, you get no feedback on whether you won or lost, and the game simply stops responding.



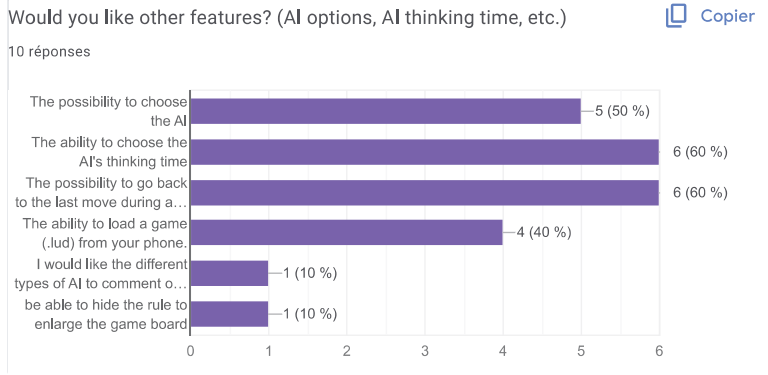
<https://docs.google.com/forms/d/1qHaXd5u2jbCRgvm4qJOdsi7FMjmBTSipx838CHkHw/viewanalytics>

4/9

Figure B.4: Survey page 4

04/01/2026 16:36

Porting Ludii to Android



Big board games evaluation and Feedback

Now try the following games: **Yavalath**, **Hex**, and **Havannah**.

For these games:

How easy is the interface to use?

10 réponses

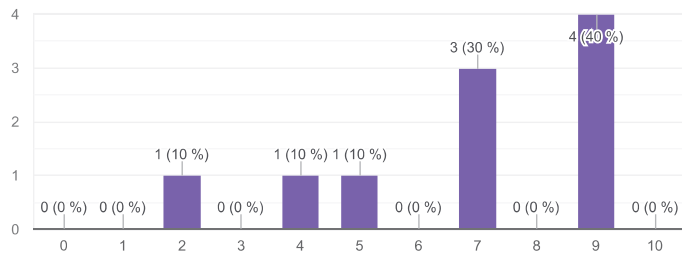


Figure B.5: Survey page 5

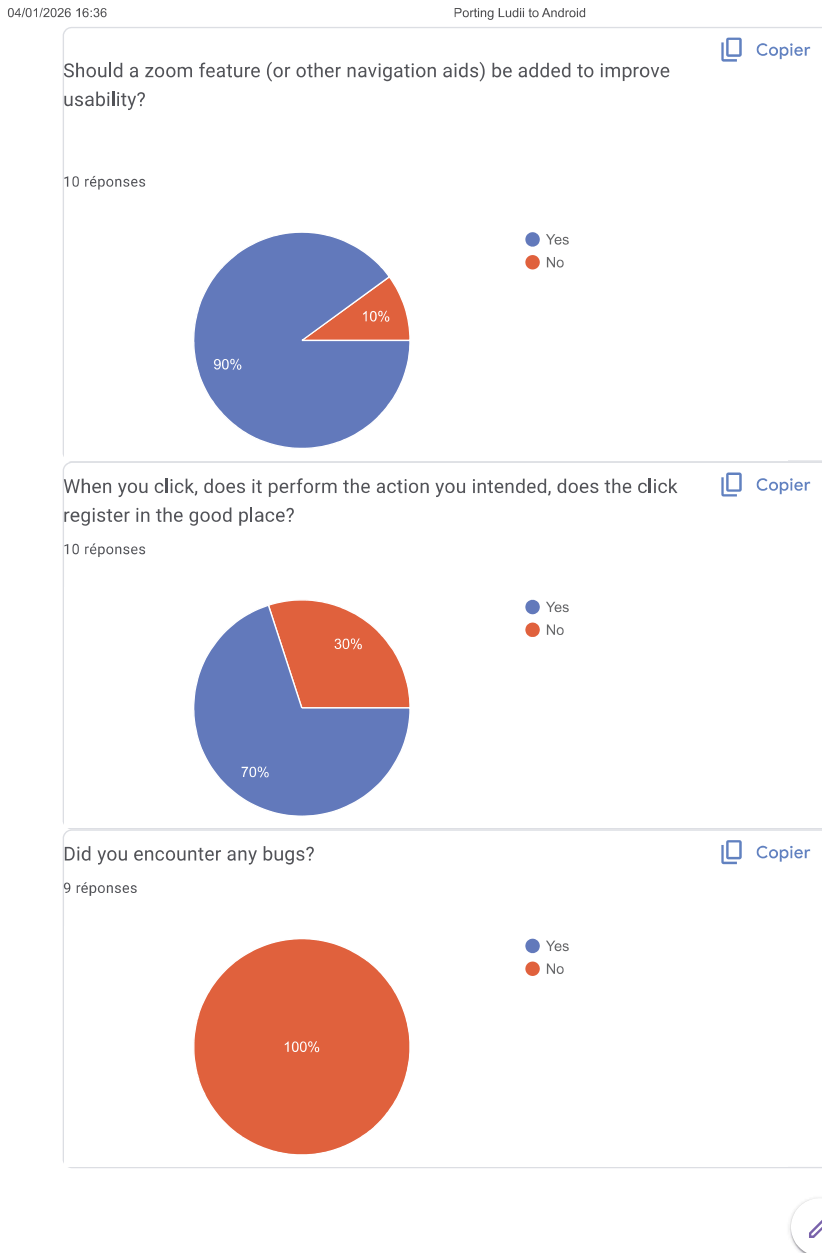


Figure B.6: Survey page 6

04/01/2026 16:36

Porting Ludii to Android

If so, please specify which bugs and, if possible, the steps to reproduce them.

2 réponses

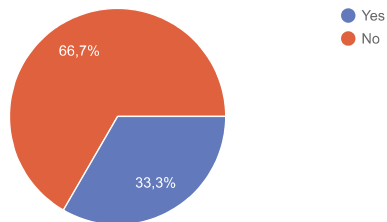
i don't know

This is not a bug, but adding navigation is important. The Hex game is so small that I sometimes click on nearby cells by mistake.

Do you think an Android version will increase your use of Ludii?

 Copier

9 réponses



Why?

8 réponses

Having it on the phone will no doubt make it more likely to fidget with or to show others. Also would probably help a lot in playtesting or larger studies.

I would like to play against other players through my phone.

My daily smartphone is on ios

I mean right now I'm only using ludii in code, I don't need the interface

I don't use Android devices

I have an iPhone so I would say no, and using the app on an emulator on PC is the same as using the Ludii app so...

But if I was on Android, I would definitely say yes, because that would allow portability for users/developpers and could be a more user-friendly/intuitive way to play some games

I don't use it

i never used it

<https://docs.google.com/forms/d/1qHaXd5u2jbCRgvm4qJOdsi7FMjmBTSipx838CHkHw/viewanalytics>

7/9

Figure B.7: Survey page 7

04/01/2026 16:36

Porting Ludii to Android

Do you have any other ideas for improvements?

3 réponses

no

- Improve navigation during games and in the navigation bar (I didn't notice the other games until I scrolled down).
- Improve the game layout: center the game area or enable a full-screen mode.
- Add a visual indicator to show when the AI is "thinking"; at first, I thought the game was bugged.

hide the rules to earn space

Ce contenu n'est ni rédigé, ni cautionné par Google. - [Contacter le propriétaire du formulaire](#) - [Conditions d'utilisation](#) - [Règles de confidentialité](#)

Ce formulaire vous semble suspect ? [Signaler](#)

Google Forms



<https://docs.google.com/forms/d/1qHaXd5u2jibCRgvm4qJ0dsi7FMjmBTSipx838CHkHw/viewanalytics>

8/9

Figure B.8: Survey page 8

