LUDII - Le Système Ludémique de General Game Playing

Éric Piette
Dennis J.N.J. Soemers
Matthew Stephenson
Chiara Sironi
Mark Winands
Cameron Browne

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

{eric.piette, dennis.soemers, matthew.stephenson, c.sironi, m.winands, cameron.browne}
@maastrichtuniversity.nl

Résumé

*Bien que les systèmes actuels de General Game Playing (*GGP*) facilitent la recherche en Intelligence Artificielle (*IA*) autour des jeux, ils sont souvent trop spécialisés et fournissent une capacité de calcul trop faible. Cet article décrit une première version du système ludémique de* GGP *dénommé* LUDII *qui apporte un outil efficace à la recherche en* IA *tout autant qu'aux concepteurs de jeux ou aux historiens mais aussi dans bien d'autres domaines.* LUDII *définit les jeux comme des arbres de ludèmes, correspondant à des concepts et mécanismes de jeu de haut niveau et facilement interprétables. Nous établissons les bases de* LUDII *en analysant ses principaux avantages : généralité, extensibilité, compréhensibilité et efficacité. Expérimentalement,* LUDII *surpasse l'un des plus performants raisonneurs décrit avec le General Game Description Language (*GDL*), basé sur un réseau de propositions (propnet), pour tous les jeux disponibles sur le serveur* GGP.

Mots Clef

General Game Playing, *Game AI*, Représentation des Connaissances, Apprentissage Automatique.

Abstract

*While current General Game Playing (*GGP*) systems facilitate useful research in Artificial Intelligence (*AI*) for game-playing, they are often somewhat specialized and computationally inefficient. In this paper, we describe an initial version of a "ludemic" general game system called* LUDII*, which has the potential to provide an efficient tool for* AI *researchers as well as game designers or historians and practitioners in related fields.* LUDII *defines games as structures of ludemes, i.e. high-level, easily understandable game concepts. We establish the foundations of* LUDII *by outlining its main benefits : generality, extensibility, understandability and efficiency. Experimentally,* LUDII *outperforms one of the most efficient Game Description Language (*GDL*) reasoners, based on a propositional network, for all available games in the* GGP *repository.*

Keywords

General Game Playing, *Game AI*, Knowledge Representation, Machine Learning.

1 Introduction

L'objectif du *General Game Playing* (GGP) est de développer des programmes-joueurs capables de jouer à une grande variété de jeux [22]. De nombreux systèmes utilisés pour modéliser des jeux, communément appelés *General Game Systems*, existent actuellement pour différents types de jeux incluant : les jeux déterministes à information complète [11], les jeux combinatoires [2], les puzzles [28], les jeux de stratégies [17], les jeux de cartes [10] et les jeux-vidéos [25].

Depuis 2005, le *General Game Systems* GGP-BASE [1] utilisant le *Game Description Language* (GDL) [16] afin de représenter les jeux est devenu le standard pour la recherche académique autour du GGP. GDL est un ensemble de clauses logiques de premier ordre décrivant les jeux à l'aide d'instructions simples. Bien que ce dernier soit dédié aux jeux déterministes avec information complète, il possède une extension dénommée GDL-II [26] incluant les jeux avec information incomplète et une seconde dénommée GDL-III [31] développée récemment intégrant les jeux épistémiques.

1.1 GDL

La capacité générique induite par GDL propose un défi algorithmique de haut niveau et a conduit à de nombreuses contributions importantes [1], particulièrement autour de *Monte Carlo Tree Search* (MCTS) [8, 9] tout en apportant différents algorithmes originaux comme la combinaison de la programmation par contraintes, MCTS et la détection de symétries [13]. Malheureusement, les aspects clés définissant un jeu (tels que les plateaux, les pièces d'un jeu, les jeux de cartes, etc) ou même les opérateurs arithmétiques doivent être redéfinies explicitement de zéro pour chaque nouvelle description de jeu.

1. https://github.com/ggp-org/ggp-base

De ce fait, modéliser et/ou débugger un jeu peut être chronophage tout en étant difficilement déchiffrable une personne n'étant pas familière avec la programmation logique. L'équipement et les règles sont fortement interconnectés à tel point que changer n'importe quel aspect du jeu nécessiterait une réécriture importante du code. Par exemple, changer la taille du plateau de 3×3 à 4×4 dans la description GDL du *Tic-Tac-Toe* demande la modification et l'ajout de nombreuses lignes de code.

De plus, GDL est actuellement limité en terme d'applicabilité en dehors de la recherche en IA pour les jeux. En effet, ce langage est verbeux et difficile à interpréter tout en n'intégrant pas les concepts clés relatifs aux jeux que tout concepteur utilise typiquement. De même, traiter informatiquement de telles descriptions est coûteux (en temps et en mémoire) du fait qu'il requiert d'utiliser un interpréteur logique rendant le langage difficilement portable pour tout autre application externe.

De nombreux jeux plus "complexes" peuvent être difficiles et demander un temps important pour être modéliser (ex : *Go*), ou sont rendus injouables suite à un important coût en temps de calcul (ex : *Échecs*). Au sein des différents dépots de jeux/instances GDL [27], seuls quelques jeux sont ajoutés chaque année suite à ces défauts.

1.2 Le Digital Ludeme Project

Le *Digital Ludeme Project* (DLP)[2] est un project de recherche sur cinq ans, récemment lancé à l'université de Maastricht, avec pour objectif de modéliser l'ensemble des jeux traditionnels dans une seule base de données numérique et permettant de jouer à ces derniers. Cette base sera utilisée afin de découvrir les liens intrinsèques entre les jeux et leurs composants dans le but de développer un modèle décrivant leur évolution tout au long de l'histoire humaine et de retracer leur propagation à travers les cultures du monde entier. Ce projet établira un nouveau champs de recherche dénommée *l'archéoludogie numérique* [6].

L'une des principales missions du DLP est de modéliser les 1.000 jeux traditionnels les plus influents au cours de l'histoire, où chacun d'entre eux peuvent avoir de multiples interprétations et nécessitant le test de plusieurs centaines de variantes. Ce n'est pas seulement un défi mathématiques et informatique mais aussi un défi logistique requérant un nouveau type de *General Game System*. Le DLP traite l'ensemble des jeux anciens de stratégies incluant principalement des jeux de plateaux, des jeux de cartes, des jeux de dés, des jeux d'empilement, des jeux de tuiles, etc, pouvant impliquer des éléments non déterministes ou de l'information incomplète. Notons que le DLP exclut tout jeu de dextérité, jeu social, sport ou jeu vidéo, etc.

Dans ce papier, nous introduisons formellement une première version de LUDII[3], le premier système Ludémique GGP complet capable de modéliser et de jouer (par un humain ou une IA) à l'ensemble des jeux de stratégies traditionnels. La notion de ludème est introduite en Section 2 avant de décrire l'approche ludémique que nous avons implémentée en Section 3. En Section 4, le système LUDII est détaillé avant de mettre en avant ses différentes aptitudes lui permettant de répondre au DLP en Section 5. L'efficacité sous-jacente du système LUDII en terme de raisonnement est démontré expérimentalement en Section 6 en le comparant à l'un des des meilleurs raisonneurs utilisant le système GGP-BASE [29].

2. *Digital Ludeme Project* : `http://ludeme.eu`
3. LUDII est nommé ainsi suite à son prédécesseur LUDI [2]

2 Ludèmes

La décomposition de jeux en ludèmes [21], i.e. unités conceptuelles d'informations relatives au jeu, nous permet de distinguer sa forme (ses règles et son équipement) de sa fonction (le contexte). Cette séparation fournit une analogie claire au génotype/phénotype qui rend possible l'analyse phylogénétique, où les ludèmes constituent les blocs d'ADN définissant chaque jeu.

Ce modèle ludémique de jeux a été démontré avec succès dans des travaux précédents impliquant de nouveaux jeux de plateau générés à partir de jeux existants [3]. Un important bénéfice de l'approche ludémique est d'encapsuler les concepts clés des jeux et de les décrire par des termes significatifs. Ceci permet la description automatique d'ensemble de règles de jeux, la comparaison entre différents jeux et potentiellement l'explication automatique des stratégies apprises par une IA en terme humainement compréhensible. De récents travaux ont montré comment ce modèle pouvait être amélioré pour une plus grande généralité et extensibilité en permettant à tout ludème modélisable d'être défini par une approche grammaticale de classe, qui dérive le langage de description de jeu directement de la hiérarchie de classes sous-jacente de la bibliothèque du code source correspondant [5].

Cette approche assure à une seule IA d'être capable de modéliser, jouer et analyser tout jeu traditionnel de stratégie comme une architecture de ludèmes. Mais aussi fournissant un mécanisme pour identifier les correspondances mathématiques entre les jeux en établissant des liens probabilistes entre eux en tant que patrimoine génétique concret.

3 L'approche Ludèmique

L'approche ludèmique utilisée pour modéliser les jeux est maintenant décrite.

3.1 Syntaxe

Definition 1 *Un état de jeu* LUDII s *encode le joueur devant jouer dans* s *(formulé par* $mover(s)$*) et cinq vecteurs contenant respectivement une donnée pour chaque localisation :* $what$, who, $count$, $state$, *et* $hidden$. *Une description plus précise de ses localisations et des données spécifiques fournies par ces vecteurs est donnée après la Définition 3.*

Definition 2 *Une fonction successeur* LUDII *est donnée par*

$$\mathcal{T} : (S \setminus S_{ter}, \mathcal{A}) \mapsto S,$$

où S est l'ensemble de tous les états de jeu LUDII*, S_{ter} l'ensemble des état terminaux, et $\mathcal{A}$ l'ensemble de toutes les listes d'actions possibles.*

Étant donné un état courant $s \in S \setminus S_{ter}$ et une liste d'actions $A = [a_i] \in \mathcal{A}$, $\mathcal{T}$ retourne un état successeur $s' \in S$. Intuitivement, une liste complète des actions A peut être interprétée comme un unique mouvement sélectionné par un joueur, pouvant avoir de multiples conséquences sur un état de jeu (chacune implémentée par une action primitive différente).

Definition 3 *Un jeu* LUDII *est représenté par un 3-tuple de ludèmes $\mathcal{G} = \langle Mode, Equipment, Rules \rangle$ où :*

- $Mode = \{p_0, p_1, \ldots, p_k\}$ *est un ensemble fini de $k+1$ joueurs, où $k \geq 1$. Les éléments aléatoires (tels que lancer un dé, lancer une pièce, distribuer des cartes, etc) sont joués par p_0, décrivant la nature/l'environnement. Le premier joueur à réaliser une action dans tout jeu est p_1, et le mover fait référence au joueur.*
- $Equipment = \langle C^t, C^p \rangle$ *où :*
  - C^t *désigne une liste de conteneurs (plateaux, main du joueur, paquets de cartes, etc). Chaque conteneur $c^t = \langle V, E \rangle$, où $c^t \in C^t$, est un graphe possédant des sommets V et des arêtes E. Chaque sommet $v_i \in V$ correspond à un emplacement jouable (ex : une case aux échecs ou encore une intersection au Go), alors que chaque arête $e_i \in E$ représente deux emplacements adjacents.*
  - C^p *dénote une liste de composants (pièces, cartes, tuiles, dés, etc) dont certaines peuvent être placées sur les emplacements des conteneurs dans C^t. Nous convenons que le composant $c_0^p \in C^p$ est placé sur l'ensemble des emplacements vides.*
- $Rules$ *définit les différentes règles d'un jeu, incluant :*
  - $Start = [a_0, a_1, \ldots, a_k]$ *représentant une liste d'actions de départ. Les actions de début de jeu sont appliquées séquentiellement sur un état "vide" (état où c_0 est présent sur chaque emplacement de tout conteneur) pour modéliser l'état initial s_0.*
  - $Play : S \mapsto \mathcal{P}(\mathcal{A})$*, où $\mathcal{P}(\mathcal{A})$ décrit l'ensemble des parties de l'ensemble $\mathcal{A}$ de toutes les listes possibles d'actions légales. C'est une fonction qui selon un état $s \in S$ retourne un ensemble de listes d'actions.*
  - $End = (Cond_0(s), \vec{\mathbb{S}}_0) \cup \ldots \cup (Cond_e(s), \vec{\mathbb{S}}_e)$ *désigne un ensemble de conditions $Cond_i(s)$ sous lesquelles un état donné s est considéré terminal. Chaque condition de terminaison $Cond_i(s)$ mène à un ensemble de scores $\vec{\mathbb{S}}_i$.*

LUDII propose quelques vecteurs pré-définis afin de simplifier leurs écritures : `Win`, `Loss`, `Draw`, `Tie`, et `Abort`. De plus, si le *mover* n'a pas d'actions légales alors il se trouve (temporairement) dans une impasse et il doit réaliser l'action spéciale `pass` sauf si les règles *End* dictent ce cas autrement. Les états dans lesquels tous les joueurs sont forcés de réaliser l'action `pass` lors du dernier tour complet de jeu sont abandonnés en tant que match nul (*Draw*).

Nous spécifions les *localisations* $loc = \langle c, v_i, l_i \rangle$ par leur conteneur $c = \langle V, E \rangle$, un sommet $v_i \in V$ et un niveau $l_i \geq 0$. Chaque localisation spécifie un emplacement particulier dans un conteneur spécifique à un certain niveau, bien que dans la plus part des jeux $l_i = 0$ excepté pour les jeux d'empilement (*Stacking games*) où possiblement plusieurs niveaux peuvent être utilisés. Pour chacune de ses localisations, un état de jeu s encode plusieurs données tel que décrit dans la Définition 1. L'indice d'un composant localisé en loc dans s est donné par $what(s, loc)$, le propriétaire (indice du joueur) par $who(s, loc)$, le nombre de composants par $count(s, loc)$, l'état interne d'un composant (coté, direction, promotion actuelle, etc) par $state(s, loc)$. Si l'état d'une localisation loc est une information cachée pour un joueur en particulier p_i alors cela est indiqué par $hidden(s, loc, p_i)$.

3.2 Exemple de jeu LUDII

Suite à la Définition 3, LUDII fournit divers ludèmes correspondant à de simples opérations pouvant être utilisées pour définir les joueurs, l'équipement et les règles. Par exemple, `(line 3)` est un ludème booléen retournant `Vrai` si dans l'état courant une ligne de trois pièces se trouve dans un conteneur et `(empty)` est un ludème retournant une liste contenant l'ensemble des emplacements libre d'un conteneur.

Dans la Figure 1, une description complète du jeu *Tic-Tac-Toe* est donnée en utilisant une grammaire de type EBNF générée par LUDII. Le ludème `mode` décrit le mode de jeu ; dans l'exemple il s'agit d'un jeu dont les tours alternent entre deux joueurs nommés `P1` et `P2`. Le premier sous-ensemble de ludème `equipment` décrit le plateau principale comme un carré de taille 3×3 et le second sous-ensemble liste les composants du jeu : un disque nommé "O" pour le joueur 1 et une croix nommée "X" pour le second. Chaque tour, le *mover* place sa pièce sur un emplacement libre, ceci est indiqué par `(to Mover (empty))`. La condition pour gagner pour le *mover* est de créer une ligne de trois de ses pièces. A noter qu'ici *Tic-Tac-Toe* ne requiert pas de règles $Start$.

```
(game "Tic-Tac-Toe"
   (mode {(player "P1")(player "P2")})
   (equipment
      {(board "Board" (square 3))}
      {(disc "O" 1) (cross "X" 2))}
   )
   (rules
      (play (to Mover (empty)))
      (end (line 3)(result Mover win))
   )
)
```

FIGURE 1: Le jeu Tic-Tac-Toe décrit avec LUDII.

Si le plateau est plein sans qu'aucun joueur ne remporte la partie, alors le jeu finira par un match nul (*Draw*) suite à l'action obligatoire *Pass* appliquée par les deux joueurs. Notons que l'utilisation judicieuse de ce paramètre par défaut permet une description plus succincte des jeux.

4 Le Système LUDII

La prochaine section introduit le système LUDII lui-même, décrivant l'approche grammaticale de classe et le noyau du système.

4.1 L'approche Grammaticale de Classe

LUDII est un système GGP complet [4] utilisant une approche grammaticale de classe dans laquelle le langage de description de jeu est automatiquement généré par les constructeurs de chaque classe du code source LUDII [5]. Les descriptions de jeux exprimées dans la grammaire sont automatiquement instanciées dans la bibliothèque de code correspondant afin d'être compilées, donnant ainsi une garantie 1:1 de correspondance entre le source code et la grammaire.

Schaul *et al.* [24] souligne que *"any programming language constitutes a game description language, as would a universal Turing machine"*. LUDII utilise ainsi efficacement son langage de programmation (Java) comme son langage de description. De ce fait, théoriquement, il peut supporter toute règle, équipement ou comportement pouvant être implémentés en Java, mais l'ensemble des détails de programmation sont cachés à l'utilisateur, qui ne voit uniquement qu'une grammaire simplifiée lui indiquant le code devant être appelé.

4.2 Le Noyau de LUDII

Le coeur de LUDII est une bibliothèque de ludèmes implémentée en Java 8, composée de nombreuses classes, chacune implémentant un ludème spécifique. Un jeu LUDII $\mathcal{G}$ définissant tous les ludèmes adéquates pour le modéliser (joueurs, équipement règles) est stocké dans un objet `Game`. Dans le contexte du GGP, permettre l'affichage de tout jeu automatiquement est crucial afin de faciliter la compréhension des stratégies appliquées par les IA. À cette fin, tous les équipements dans le système LUDII implémentent l'interface `Drawable`, permettant à chacun d'entre eux d'être illustrer par une image *bitmap* adaptable à toute résolution, afin de représenter tout état du plateau et de ses composants. Les conteneurs C^t peuvent ainsi dessiner leurs composants actuels aux positions, orientations, états appropriés. Un object `View` fournit le mécanisme permettant d'afficher l'état courant du jeu à l'écran et un object `Controller` permet de mettre à jour l'état du jeu suite aux entrées de l'utilisateur tel que les clics de la souris. De ce fait, tous les jeux disponibles dans le système peuvent être joués à la fois par tout humain et/ou par une IA.

En tant qu'exemple, la Figure 2 représente un jeu à deux joueurs $\mathcal{G}$ avec $C^t = \{c_0^t\}$, où c_0^t est un conteneur hexagonal composé de cellules hexagonales. $C^p = \langle c_0^p, c_1^p, c_2^p \rangle$, où c_0^p et le composant "vide", c_1^p est le disque blanc pour le joueur p_1 et c_2^p illustre le disque noir du joueur p_2. Le

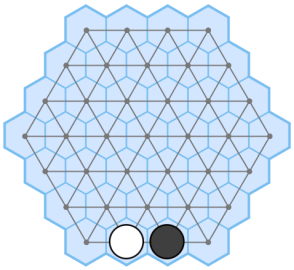

FIGURE 2: Un conteneur hexagonal fragmenté par des hexagones et le dual du graphe.

système possède une modélisation par un graphe pour visualiser le plateau, les sommets, les arêtes et les côtés de ce dernier sont dépeints en bleu. Le *dual* de ce graphe, qui est le graphe donné par c_0^t est dépeint en gris.

Le graphe du jeu peut lui-même être modifié dans certains "jeux de graphes" (ex : *Dots & Boxes*) dans lequel les mouvements d'un joueur implique des modifications du graphe lui-même (ex : Ajouter/Supprimer des arêtes ou des sommets). Afin d'optimiser le raisonnement/calcul, plusieurs données sont pré-générées comme les coins du plateau, les sommets extérieurs, les sommets le long du haut du plateau, etc. dans la classe `Graph` et les sommets voisins pour chaque direction, etc dans la classe `Vertex`.

Les vecteurs de données $what$, who, etc. d'un état s sont implémentées par une collection d'objets `ContainerState`. Différentes représentations de l'état sont implémentées dans le but de minimiser l'empreinte mémoire et d'optimiser le temps nécessaire pour accéder aux données nécessaires au raisonnement dans tout jeu :

- Pièces identiques pour chaque joueur (ex : *Hex*).
- Pièces distinctes par joueur (ex : *Échecs*).
- Pièces avec état local (ex : *Reversi*).
- Compteur de pièces par localisation (ex : *Mancala*).
- Jeux d'empilement (ex : *Laska*).
- Sans plateau fixe (ex : *Dominoes*).
- Information cachée (ex : *Stratego*, *jeux de cartes*).

LUDII sélectionne automatiquement la représentation appropriée de l'état à partir des règles afin de créer l'objet `Game`, assurant la représentation qui convient le mieux.

Les états des conteneurs sont définis dans une classe `BitSet` modifiée sur mesuse, dénommée `ChunkSet`, qui condense les informations induites par l'état de jeu dans un espace mémoire minimum en fonction de la description de chaque jeu. Par exemple, si un jeu n'inclue pas plus d'équipement qu'un plateau et des pièces identiques de N couleurs, alors l'état de jeu sera décrit par un `ChunkSet` subdivisé en bloc (*chunk*) de B *bits* par cellule du plateau, où B est la plus faible puissance de 2 fournissant assez de *bits* pour représenter chaque état possible par cellule (incluant l'état 0 pour les cellules vides) [4]

4. La taille des blocs est établie par la puissance de 2 la plus basse afin d'éviter les problèmes de chevauchement issues de valeurs `long` consécutives.

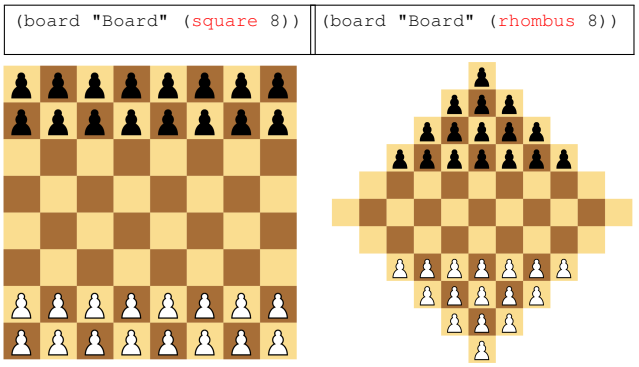

FIGURE 3: Le jeu *Breakthrough* sur un plateau carré (gauche) et sur un plateau losange (droite).

En utilisant l'état de jeu et les différents ludèmes décrivant les règles du jeu, le système calcule les mouvements légaux pour tout état. L'arbre de ludèmes est évalué en retournant la liste d'objets `Action` correspondant au *mover*. Chaque objet `Action` décrit une ou plusieurs actions atomiques qui doivent être appliquées à l'état du jeu pour exécuter le mouvement. Les actions incluent typiquement d'ajouter ou supprimer des composants des conteneurs ou changer le compte des composants par exemple.

Definition 4 *Un trial τ est une suite d'états s_i et une liste d'actions A_i :*

$$\langle s_0, A_1, s_1, \ldots, s_{f-1}, A_f, s_f \rangle$$

tel que $f \geq 0$, et pour tout $i \in \{1, \ldots f\}$,

- *la liste d'actions réalisée A_i est légale pour $mover(s_{i-1})$*
- *les états sont mis à jour : $s_i = \mathcal{T}(s_{i-1}, A_i)$*
- *seul s_f peut être terminal : $\{s_0, \ldots, s_{f-1}\} \cap S_{ter} = \emptyset$*

τ est donné par l'objet `Trial`, fournissant un enregistrement complet du jeu joué du début à la fin incluant les mouvements réalisés.

Dans tout jeu, le raisonnement d'une IA peut être parallélisé en utilisant des *trials* séparés par *thread*. Toutes les données d'un objet `Game` sont constantes et de ce fait peuvent être partagées entre plusieurs *threads*. Un *thread* se référera à un objet `Trial` pour réaliser tout *playout* à partir de n'importe quel état. Dans le système, chaque objet IA décrit l'implémentation d'une IA choisie par le joueur, incluant les limites de temps/budgets de calcul, les indices caractéristiques (*features*) pour biaiser un *playout* [7], etc.

5 Les aptitudes et propriétés clés

LUDII est conçu et implémenté essentiellement pour fournir des réponses aux questions soulevées par le DLP mais sera également une plate-forme pour la recherche autour des jeux dans différents domaines incluant IA, conception, histoire et éducation. LUDII fournit de nombreux avantages par rapport aux système GGP existants, comme suit :

Simplicité : La simplicité fait référence à la facilité avec laquelle les descriptions de jeux peuvent être créées et modifiées. Elle peut être estimée par le nombre de symboles (*tokens*) requis pour définir un jeu. Décrire un jeu avec l'approche ludémique est typiquement beaucoup plus simple que tout approche basée sur la logique (ex : LUDII demande seulement 29 symboles pour *Tic-Tac-Toe* et 298 pour les *Échecs* alors que GDL demande 381 et 4.932 symboles respectivement). De plus, la modélisation ludèmique permet aussi de modifier un jeu facilement pour expérimenter différentes tailles, géométries ou règles. Par exemple, modifier la taille et la forme d'un plateau (ex : Figure 3) peut être accompli en modifiant un seul paramètre, tandis que la même modification avec GDL requiert de nombreuses lignes de code à ajouter ou à modifier.

Clarté : La clarté mesure la capacité de chaque description de jeu d'être comprise pour les non spécialistes. La description de jeu par l'approche logique que propose GDL est souvent difficile à interpréter pour un humain. Dans LUDII, les classes Java définissant chaque ludème sont qualifiées par des noms significatifs décrivant clairement les rôles et concepts qu'ils impliquent. Ceci est particulièrement utile pour les jeux dans lesquels des concepts mathématiques (géométries, algèbres, arithmétique, etc.) sont encapsulés au sein des ludèmes les composants.

Généralité : La généralité se reporte à la quantité de jeux pouvant être couverte par le système. Comme LUDII utilise l'approche grammaticale de classes pour décrire les ludèmes, il peut théoriquement supporter tout jeu pouvant être programmé en Java, la version initiale de LUDII décrit dans ce papier inclut déjà de nombreux jeux différents en dehors de ceux pouvant être implémentés par GDL. La version finale de LUDII sera encore plus générale et facilitera l'ajout de nombreux types de jeux additionnels.

Extensibilité. L'extensibilité décrit la capacité d'ajouter de nouvelles fonctionnalités au système. La version initiale de LUDII propose 68 jeux différents (et 179 variantes) utilisant 259 différents ludèmes, qui testent les différentes options et capacités du système. Étendre LUDII revient simplement à ajouter de nouvelles classes à la bibliothèque de ludèmes, qui seront ensuite automatiquement incorporées à la grammaire, rendant ainsi l'extensibilité très libre. Étendre GDL implique des modifications significatives du langage et du coeur du système correspondant.

Évolutivité : Ceci représente la capacité d'utiliser des descriptions de jeux afin de produire des "enfants" qui ressemblent à leurs "parents". Les descriptions de jeux GDL tendent à impliquer des chaînes d'opérations logiques qui doivent être élaborées avec le plus grand soin. L'application de mutations et croisements aléatoires entre descriptions de jeux GDL est très peu susceptible de produire des résultats corrects (e.i. jouables) et encore moins d'améliorer les parents. Réciproquement, l'approche ludèmique est idéale aux approches évolutives telles que la programmation génétique [15] et a déjà fait ses preuves dans l'évolution/génération de nouveaux jeux de haute qualité [2].

Application Culturelle : Outre ses avantages pour le GGP, LUDII a également plusieurs applications en tant qu'outil pour le nouveau domaine de l'archéoludologie [6]. Le système LUDII sera relié à un serveur et à une base de données contenant les informations culturelles et historiques pertinentes sur les jeux. Ces informations fourniront non seulement un contexte réel supplémentaire, mais permettra également de reconstituer des ensembles de règles viables et historiquement authentiques pour tout jeu contenant certaines informations manquantes, pour développer un "arbre généalogique" des jeux traditionnels et de cartographier la propagation des jeux, à travers l'histoire.

Universalité : Bien que LUDII supporte un large éventail de jeux, y compris des jeux non déterministes à information cachée, nous ne pouvons pas prouver l'universalité de l'ensemble de la grammaire dans le cadre de cet article. Toutefois, nous démontrons que LUDII est universel pour les jeux finis et déterministes à information complète :

Théorème 1 LUDII *est universel pour la classe de jeux finis et déterministes à information complète.*

Dans LUDII, cette classe de jeux ne requiert pas de nature (p_0) et d'informations cachées ($hidden(s, loc, p_i)$ retourne faux pour tout état s, dans toute localisation loc, et pour tout joueur p_i). La preuve est structurée similairement à la preuve d'universalité pour GDL et GDL-II [30]. Basée sur la définition d'un jeu en forme extensive [23], nous formalisons les jeux déterministes et prouvons que LUDII peut définir un arbre de jeux fini arbitrairement. Par conséquent, tous les jeux qui peuvent être modélisés en GDL peuvent aussi être décrit dans LUDII.

5.1 Preuve du Théorème 1

Similairement à Kowalski *et al.* [2019], nous formalisons un jeu fini et déterminisite à k joueurs avec information complète par un tuple (k, T, ι, υ), où :

- $k \in \mathbb{N}$ indique le nombre de joueurs.
- T est un arbre fini avec :
  - Les noeuds S (faisant références aux états du jeu).
  - Un état initial $s_0 \in S$ (la racine de T).
  - Les états terminaux $S_{ter} \subseteq S$ (les feuilles de T).
  - Une fonction prédécesseur $f : (S \setminus \{s_0\}) \mapsto S$, tel que $f(s)$ désigne le parent de s dans T.
- $\iota : (S \setminus S_{ter}) \mapsto \{0, \ldots, k\}$ indiquant quel joueur à le contrôle dans un état donné.
- $\upsilon : S_{ter} \mapsto \mathbb{R}^k$, tel que $\upsilon(s)$ désigne le vecteur de gains pour k joueurs pour état terminal $s \in S_{ter}$.

Cela équivaut à la formalisation des jeux en forme extensives à k joueurs par [23], excluant tous les éléments non déterministes ou à information incomplète.

Nous prouvons que, étant donné tout jeu déterministe fini avec information complète tel que défini ci-dessus, un jeu LUDII peut être construit de telle sorte qu'il existe un mappage un à un entre les états et les transitions d'états entre le jeu original et le jeu LUDII. L'intuition de la preuve est de construire un jeu LUDII où le plateau de jeu est représenté par un graphe avec une structure identique à l'arbre complet de jeu T. Le jeu LUDII se joue en déplaçant un seul jeton, placé sur la racine à l'état initial, le long du graphe jusqu'à atteindre l'une des feuilles. Pour tout état z du jeu d'origine, il existe un état correspondant s dans le jeu LUDII tel que le jeton se trouve sur le sommet correspondant à la position z dans T. Notez qu'énumérer explicitement l'arborescence complète du jeu en tant que graphe n'est probablement pas la représentation la plus optimale pour la plupart des jeux, mais cela montre que LUDII est capable de représenter tous ces jeux.

Definition 5 *Soit $\mathcal{D} = (k, T, \iota, \upsilon)$ le jeu fini déterministe à information complète à k joueurs tel que formalisé ci-dessus. Nous définissons un jeu* LUDII *$\mathcal{G}(\mathcal{D}) = \langle Mode, Equipment, Rules \rangle$, où $Rules = \langle Start, Play, End \rangle$, tel que :*

- *$Mode = \langle p_0, p_1, \ldots, p_k \rangle$, où p_i pour $i \geq 1$ correspond aux différents k joueurs. La joueur environnement p_0 restera inutilisé dans les jeux déterministes.*
- *$Equipment = \langle \{c_0^t\}, \{c_0^p, c_1^p\} \rangle$. Le conteneur $c_0^t = \langle V, E \rangle$ est un graphe possédant une structure identique à l'arbre T du jeu original $\mathcal{D}$. Étant donné que la structure c_0^t étant identique à celle de T, nous pouvons uniquement identifier un sommet $v(z)$ pour chaque état $z \in T$ du jeu original. Pour tout sommet de ce type – à l'exception de $v(s_0)$ – nous pouvons également identifier uniquement un sommet "parent" adjacent $p(v(z))$, tel que $p(v(z)) = v(f(z))$; le parent d'un sommet correspond au prédécesseur de l'état correspondant dans l'arbre d'origine T.*
- *Les règles $Start$ sont données par une liste contenant uniquement une action. Cette action crée l'état initial du jeu en plaçant le jeton c_1^p sur le site $v(s_0)$ de c_0^t qui correspond à la racine de T.*
- *Soit s tout état non terminal du jeu* LUDII *tel qu'il est exactement un site $v(z)$ pour chaque $what(s, \langle c_0^t, v(z), 0 \rangle) = c_1^p$. Soit z l'état dans le jeu original qui correspond au site $v(z)$. Soit $g(z)$ les enfants de z dans T. Étant donné s, nous définissons $Play(s)$ pour retourner un ensemble $\{A_i\}$ de listes d'actions A_i, avec une liste d'actions pour chaque noeud enfant $z' \in g(z)$. Chacune de ces listes contient deux actions primitives; une qui prend le jeton c_1^p du site $v(z)$ (le replaçant par le jeton "vide" c_0^p), et une seconde action qui place ce jeton c_1^p sur le site $v(z')$ de c_0^t qui correspond à l'enfant $z' \in T$.*
- *Les règles End sont données par $End = \{(Cond_i(\cdot), \vec{\mathbb{S}}_i)\}$. Pour tout état terminal $z_i \in S_{ter}$, soit $v(z_i)$ le site dans le graphe c_0^t qui correspond à la position de z_i dans T. Nous ajoutons un tuple $(Cond_i(\cdot), \vec{\mathbb{S}}_i)$ à End tel que $Cond_i(s)$ retourne vrai si et seulement si $what(s, loc) = c_1^p$ pour $loc = \langle c_0^t, v, 0 \rangle$, et $\vec{\mathbb{S}}_i = v(z_i)$. Intuitivement, nous utilisons une condition de fin séparée pour chaque état terminal*

possible $z_i \in S_{ter}$ dans le jeu original $\mathcal{D}$, qui vérifie sa véracité spécifiquement pour cet état en s'assurant que le jeton c_1^p est placé sur le sommet $v(z_i)$.

— *Soit s l'état* LUDII *non terminal, tel qu'il existe un site $v(z)$ pour chaque $what(s, \langle c_0^t, v(z), 0\rangle) = c_1^p$. Soit z l'état original du jeu qui correspond au site $v(z)$. Alors nous définissons $mover(s) = \iota(z)$.*

Lemme 1 *Soit $\mathcal{G}(\mathcal{D})$ le jeu* LUDII *construit par la Définition 5. Chaque état de jeu s qui peut être atteint par une suite d'actions légales dans un tel jeu a exactement un sommet $v \in c_0^t$ tel que $what(s, \langle c_0^t, v, 0\rangle) = c_1^p$, et $what(s, \langle c_0^t, u, 0\rangle) = c_0^p$ pour tous les autres sommets $u \neq v$.*

Intuitivement, ce lemme dit que chaque état de jeu accessible par une suite d'actions légales a le jeton c_1^p situé sur exactement un sommet et que tous les autres sommets sont toujours vides (indiqué par c_0^p).

Preuve 1 *Soit s_0 l'état initial du jeu. Les règles $Start$ sont définies en plaçant un jeton c_1^p sur $v(z_0)$, où z_0 est l'état initial du jeu d'origine $\mathcal{D}$, signifiant que le lemme est valide pour s_0.*
Soit s l'état non terminal pour lequel le lemme est validé. Alors les hypothèses de la Définition 5 pour une définition adéquate de $Play(s)$ sont satisfaites, ce qui signifie que $\{A_i\} = Play(s)$ est un ensemble non vide de listes d'actions, dont l'une doit être sélectionnée par le $mover(s)$. Chaque A_i est définie pour déplacer le jeton c_1^p d'un sommet où il est actuellement, pour le placer sur un nouveau sommet. Cela signifie que le lemme est également valable pour tout état successeur $\mathcal{T}(s, A_i)$, ce qui prouve que le lemme est valide par induction pour tout état.

Nous sommes maintenant prêt à prouver le Théorème 1 :

Preuve 2 *Soit $\mathcal{D}$ un jeu en forme extensive, avec un arbre T. Soit $\mathcal{G}(\mathcal{D})$ un jeu* LUDII *construit par la Définition 5. Nous démontrons que pour toute traversée arbitraire à travers T, de s_0 à tout état terminal $z_{ter} \in S_{ter}$, il existe un trial τ équivalent, tel que définit dans la Définition 4, dans $\mathcal{G}(\mathcal{D})$. Par trial "équivalent", nous voulons dire que la séquence d'états traversés est de même taille, que l'ordre dans lequel les joueurs sont en contrôle est égale et que les vecteurs de gains sont les mêmes.*

Soit $z_0, z_1, \ldots, z_f$ un playout arbitraire de $\mathcal{D}$, tel que z_0 est l'état initial, et $z_f \in S_{ter}$. Par construction, l'état initial s_0 de $\mathcal{G}(\mathcal{D})$ a le jeton c_1^p placé sur le sommet $v(z_0)$ correspondant à la racine de T. Cela signifie que nous avons une correspondance un à un de z_0 à s_0, où $what(s_0, \langle c_0^t, v(z_0), 0\rangle) = c_1^p$.

Soient z_i les états non terminaux dans la séquence $z_0, z_1, \ldots, z_{f-1}$, tels que nous ayons une correspondance unique z_i sur un état LUDII *s_i où $what(s_i, \langle c_0^t, v(z_i), 0\rangle) = c_1^p$. Le Lemme 1 garantit que les hypothèses requises pour la Définition 5 pour bien définir $Play(s_i)$ sont satisfaites. De plus, nous savons que $\iota(z_i) = mover(s_i)$, ce qui signifie que le même joueur a le contrôle. La définition de $Play(s_i)$ assure qu'il y a exactement une liste d'actions légales A_i tel que $\mathcal{T}(s_i, A_i) = s_{i+1}$, où $what(s_{i+1}, \langle c_0^t, v(z_{i+1}), 0\rangle) = c_1^p$ (Notons que z_{i+1} doit être le successeur de z_i dans T). Nous choisissons s_{i+1} pour correspondre uniquement à z_{i+1}.*

Par induction, ceci termine la correspondance unique entre les séquences d'états $z_0, z_1, \ldots, z_f$ et $s_0, s_1, \ldots, s_f$, spécifiant uniquement la liste d'actions A_i qui doit être sélectionnée en cours de route, et assure que z_i correspond toujours à l'état s_i tel que le jeton c_1^p est placé sur $v(z_i)$. Cette dernière observation assure que l'une des conditions End dans $\mathcal{G}(\mathcal{D})$ est déclenchée pour s_f, et que le bon vecteur de gain $\vec{\mathbb{S}} = v(z_f)$ est sélectionné.

6 Expériences

Le système LUDII, comme la plupart des autre systèmes GGP, utilise MCTS comme méthode de base pour les IA, car cette méthode s'est avérée être une approche supérieure pour les jeux en général en l'absence de connaissances spécifiques à un domaine [9]. Les simulations MCTS (e.i. *playouts*) nécessitent des moteurs de raisonnement rapides pour obtenir le nombre souhaité de simulations. Par conséquent, nous utilisons des simulations *flat Monte Carlo* (i.e. *trials* τ où $s_f \in S_{ter}$) comme métrique pour comparer l'efficacité de LUDII sur les autres systèmes GGP.

6.1 Schéma expérimental

Dans la comparaison suivante, nous comparons GGP-BASE et LUDII en nous basant sur le nombre de *playouts* obtenu par seconde. Pour GGP-BASE, nous utilisons les descriptions des jeux les plus rapides du dépôt [27] et testons l'un des raisonneurs GGP-BASE les plus efficaces basé sur les réseaux de propositions (*propnets*) [29].

Les *Propnets* accélère le processus de raisonnement par rapport aux raisonneurs personnalisés ou basés sur le *Prolog* en traduisant les règles GDL en un graphe dirigé ressemblant à un circuit logique, dont les noeuds correspondent à des portes logiques ou à des propositions GDL qui représentent l'état, les mouvements des joueurs et d'autres aspects du jeu. Les informations sur l'état actuel peuvent être calculées en définissant la valeur de vérité des propositions qui correspondent à l'état et en propageant ces valeurs à travers le graphique. Définir et propager les valeurs de vérité des propositions qui correspondent aux actions des joueurs permet de calculer l'état suivant.

Toutes les expérimentations sont réalisées sur un seul coeur d'un Intel(R) XEON(R) CPU E5-2680 v2 à 2.80 GHz avec 2GB RAM, passant 10 minutes par test.

6.2 Résultats

Les résultats de nos expérimentations sur une sélection de jeux GDL disponibles sont présentés dans le Tableau 1. La partie haute du tableau est dédiée aux jeux à un joueur et la partie basse aux jeux multi-joueurs. La colonne la plus à droite illustre le facteur d'accélération obtenu par LUDII sur GGP-BASE.

Jeu	LUDII	GDL	Taux
Jeux à un joueur			
8 Puzzle	26.958	4.113	6,55
8 Queens	963.443	1.946	495,10
16 Queens	785.973	522	575,85
Futoshiki (4×4)	294.772	23.797	12,39
Futoshiki (5×5)	273.348	11.494	23,78
Futoshiki (6×6)	182.245	5.838	31,56
Knight's Tour (8×8)	91.156	75.000	1,21
Lights Out (5×5)	32.889	11.799	2,79
Nonogram (5×5)	279.242	59.044	4,73
Nonogram (10×10)	45.817	6.883	6,65
Peg Solitaire	7.713	3,.72	2,43
Sudoku	88.565	635	139,47
Towers of Hanoi 3	264.345	35.847	7,37
Jeux Multi-joueurs			
Amazons (10×10)	2.933	185	15,85
Breakthrough (6×6)	19.022	3.923	4,85
Breakthrough (8×8)	6.632	1.123	5,91
Chess	1.075	0,06	17,916,67
Connect 4 (6×7)	124.349	13.664	9,10
Connect 4 (12×9)	99.697	9.856	10,12
English Draughts	1.914	872	2,19
Gomoku (15×15)	2.514	927	2,71
Hex (9×9)	21.244	195	108,94
Knightthrough (8×8)	5.085	1.869	2,72
Reversi (8×8)	2.085	203	10,27
Skirmish (8×8)	2.278	124	18,37
Tic-Tac-Toe (3×3)	641.445	85.319	7,52
Tic-Tac-Toe (5×5)	197.972	11.453	17,29
Tron (10×10)	494.999	121.989	4,06
Wolf and Sheep (8×8)	22.305	5.532	4,03

TABLE 1: Le nombre moyen de *playouts* par seconde pour des jeux disponibles dans le dépôt GGP Tiltyard.

Jeu	LUDII	Jeu	LUDII
Alquerque	796	Mu-Torere	7.719
Ashtapada	12.755	Nine men's morris	5.121
Connect 6 (19×19)	21.873	Oware	3.275
Dara	5.408	Ploy	819
Fanorona	1.906	Tant Fant	5.549
Hnefatafl (11×11)	326	Three men's morris	100.174
MineSweeper (8×8)	68.233	Yavalath	189.335

TABLE 2: Le nombre moyen de *playouts* par seconde pour des jeux indisponibles en GDL.

Le Tableau 2 met en évidente nos résultats sur une sélection de jeux, incluant plusieurs jeux historiques qui n'ont pas d'équivalence en GDL. Le fait qu'aucun système GGP ou langage de description existant ne prenne en charge l'ensemble des jeux requis pour le DLP a été une motivation majeure pour le développement de LUDII.

6.3 Discussion

LUDII surpasse GGP-BASE en terme d'efficacité pour l'ensemble des jeux testés. L'amélioration de performance pour les jeux à un joueur (puzzles) s'accroit avec la taille des puzzles (ex : *Futoshiki* de 12.39 à 31.56). Pour quelques uns proposant des descriptions GDL optimisées (ex : *Knight's Tour*) LUDII accomplit des performances similaires, mais pour d'autres (ex : *N Queens* ou *Sudoku*) LUDII obtient un nombre de *playouts* significativement plus important, de 100 à pratiquement 600 fois plus rapide.

LUDII est au moins 2 fois plus rapide pour tous les jeux multi-joueurs testés. Pour des jeux simples, la taille du plateau est étroitement liée à l'amélioration de la vitesse ; LUDII est près de 7 fois plus rapide que le *Tic-Tac-Toe* 3×3 mais plus de 17 fois plus rapide pour une version plus grande de 5×5. Pour des jeux plus complexes, tels que *Amazons* et *Skirmish*, LUDII est une fois encore plus efficace que GDL (plus de 15 fois rapide dans ces cas).

La plus grande différence de vitesse concerne les *Échecs*, avec un taux d'amélioration de près de 18.000. La description GDL pour les *Échecs* ne peut être traduite en un réseau de proposition suite à une taille très importante dépassant la mémoire disponible, c'est pourquoi nous utilisons un prouveur GDL disponible dans le GGP-BASE pour comparaison. Le GGP-BASE prouveur est généralement plus lent pour les jeux complexes comparés au *propnet*, expliquant le faible nombre de *playouts* obtenu $(0, 06)$.

Le DLP nécessite la capacité de modéliser un large éventail de jeux de stratégies traditionnels. Dans le Tableau 2, LUDII démontre sa capacité à raisonner sur une variété de différents jeux : jeux de courses (ex : *Ashtapada*), jeux à information incomplète (ex : jeux de cartes ou *MineSweeper*), jeux Mancala (ex : *Oware*), jeux avec un plateau imposant (ex : *Connect-6*), etc. Cependant, pour *Hnefatal*, le plus connu des jeux de *Tafl*, la version actuelle de LUDII obtient un faible nombre de *playouts* par seconde. Ceci est probablement dû au fait que les *playouts* peuvent être long, souvent supérieur à 1.000 mouvements, car il est très peu probable que la victoire de chaque joueur soit atteinte en jouant aléatoirement. Des travaux sont en cours pour résoudre ce problème dans la version suivante de LUDII.

Kowalski *et al.*[14] ont récemment proposé un nouveau langage le *Regular BoardGames* (RBG) basé sur les langages réguliers, qui fournit une meilleure expressivité, efficacité et clarté que GDL. Nos premières recherches autour de ce nouveau langage révèlent qu'il est concis mais limité aux jeux de société déterministes dont la géométrie peut être décrite au format ASCII. Il semble également être moins efficace que l'approche grammaticale de classes ; ex : Pour le jeu de *Hex* 9×9, RBG obtient 3.425 *playouts* par seconde mais LUDII en obtient 21.244. De futurs travaux seront conduis afin d'établir une comparaison plus détaillée entre ces deux approches.

7 Conclusion

Le système ludèmique de GGP LUDII surpasse GGP-BASE, l'actuel standard pour la recherche académique en IA autour de GGP, en termes de raisonnement. Il apporte aussi de nombreuses aptitudes inégalées par aucun autre système en terme de simplicité, de clarté, de généralité, extensibilité, évolutivité et il est réalisé de sorte d'être applicable à d'autres champs de recherche que l'IA.

Les bénéfices de cette nouvelle approche GGP présentent plusieurs opportunités pour de futurs travaux. Par exemple, la découverte de nouvelles *features* par l'apprentissage par renforcement pourra être visualisé automatiquement afin de révéler des nouvelles stratégies efficaces ou

fournir des descriptions de stratégies humainement compréhensibles basées sur les ludèmes. Améliorer MCTS en le polarisant par ces *features* est un travail en cours.

Références

[1] Yngvi Björnsson et Stephan Schiffel, General Game Playing, *Handbook of Digital Games and Entertainment Technologies*, pp. 1–23, 2016.

[2] Cameron B. Browne, *Automatic generation and evaluation of recombination games*, Queensland University of Technology, 2009.

[3] Cameron B. Browne, *Evolutionary Game Design*, Springer, 2011.

[4] Cameron Browne, Julian Togelius et Nathan Sturtevant, Guest Editorial : General Games, *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 6, pp. 317–319, 2014.

[5] Cameron B. Browne, A Class Grammar for General Games, *Advances in Computer Games*, Vol. 10068, pp. 167–182, 2016.

[6] Cameron B. Browne, Back to the Past : Ancient Games as a New AI Frontier, *AAAI*, 2017.

[7] Cameron Browne, Dennis J.N.J. Soemers et Eric Piette, Strategic Features for General Games, *Proceedings of the 2nd Workshop on Knowledge Extraction from Games co-located with 33rd AAAI Conference on Artificial Intelligence*, pp. 70–75, 2019.

[8] Hilmar Finnsson et Yngvi Björnsson, Simulation-Based Approach to General Game Playing, *The Twenty-Third AAAI Conference on Artificial Intelligence*, pp. 259–264, 2008.

[9] Hilmar Finnsson and Yngvi Björnsson, Learning Simulation Control in General Game-Playing Agents, *The Twenty-Fourth AAAI Conference on Artificial Intelligence*, pp. 954–959, 2010.

[10] Jose M. Font, Tobias Mahlmann, Daniel Manrique et Julian Togelius, Applications of Evolutionary Computation, *16th European Conference, EvoApplication Proceedings*, Vol. 7835 LNCS, pp. 254–263, 2013.

[11] Michael R. Genesereth, Nathaniel Love et Barney Pell, General Game Playing : Overview of the AAAI Competition, *AI Magazine*, Vol. 26, pp. 62-72, 2005.

[12] Michael R. Genesereth et Yngvi Björnsson, The International General Game Playing Competition, *AI Magazine*, Vol. 34, pp. 107-111, 2013.

[13] Frédéric Koriche, Sylvain Lagrue, Éric Piette et Sébastien Tabary, Constraint-Based Symmetry Detection in General Game Playing, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pp. 280-287, 2017.

[14] Jakub Kowalski, Mika Maksymilian, Jakub Sutowicz et Marek Szykula, Regular Boardgames, *The Thirty-Third AAAI Conference on Artificial Intelligence*, 2019.

[15] John Koza, *Genetic Programming*, MIT Press, 1992.

[16] Nathaniel Love, Timothy Hinrichs, David Haley, Eric Schkufza et Michael Genesereth, *General Game Playing : Game Description Language Specification*, Stanford University, 2008.

[17] Tobias Mahlmann, Julian Togelius et, Georgios N. Yannakakis, Modelling and evaluation of complex scenarios with the Strategy Game Description Language, *IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 174–181, 2011.

[18] Jeff Mallet et Mark Lefter, Zillons of Games : Unlimited Board Games & Puzzles, `hhttps://www.zillions-of-games.com`, 1998.

[19] Joe Marks et Vincent Hom, Automatic Design of Balanced Board Games, *Proceedings of the Third Artificial Intelligence and Interactive Digital Entertainment Conference* pp. 25–30, 2007.

[20] Harold J. R. Murray, *A History of Board-Games Other Than Chess.*, Clarendon Press, 1952.

[21] David Parlett, What's a Ludeme ?, *Game Puzzle Design*, Vol. 2, pp. 83–86, 2016.

[22] Jacques Pitrat, Realization of a general game-playing program, *IFIP Congress*, Vol. 2, pp. 1570–1574, 1968.

[23] Eric Rasmusen, *Games and Information : An Introduction to Game Theory, 4th ed.*, B. Blackwell, 2007.

[24] Tom Schaul, Julian Togelius et Jürgen Schmidhuber, Measuring Intelligence through Games, *CoRR*, Vol. abs/1109.1314, 2011.

[25] Tom Schaul, An Extensible Description Language for Video Games, *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 6, pp. 325–331, 2014.

[26] Stephan Schiffel et Michael Thielscher, Representing and Reasoning About the Rules of General Games With Imperfect Information, *Journal of Artificial Intelligence Research*, Vol. 49, pp. 171–206, 2014.

[27] Sam Schreiber, Games-base repository, `http://games.ggp.org/base/`, 2016.

[28] Mohammad Shaker, Mhd Hasan Sarhan, Ola Al Naameh, Noor Shaker et Julian Togelius, Automatic generation and analysis of physics-based puzzle games, *IEEE Conference on Computational Inteligence in Games (CIG)*, pp. 1–8, 2013.

[29] Chiara F. Sironi et Mark H.M. Winands, Optimizing Propositional Networks, *Computer Games. Springer* pp. 133–151, 2017.

[30] Michael Thielscher, The general game playing description language is universal, *Proceedings of the Twenty-second International Joint Conference on Artificial Intelligence, IJCAI-11*, pp. 1107–1112, 2011.

[31] Michael Thielscher, GDL-III : A Description Language for Epistemic General Game Playing, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pp. 1276–1282, 2017.