Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE Corso di Laurea in Informatica

Realizzazione di un gioco a informazione incompleta sulla piattaforma Ludii

Relatore:

Prof. Paolo Ciancarini

Presentata da:

Andrea Cristiano

Sessione II Anno Accademico: 2022-2023

A Mamma,

forza motrice della mia motivazione.

Alla mia famiglia,

Il cui sostegno è instancabile e fondamentale.

Sommario

In questa tesi presentiamo il progetto e l'implementazione di una versione ad informazione incompleta del gioco da tavola Tic Tac Toe (noto in Italia anche come Tris) nel sistema Ludii.

Il gioco presentato, essendo ad informazione incompleta, prevede una differenza chiave con la sua versione originale: ciascun giocatore è in grado di visualizzare in campo solo i pezzi da lui posti, senza vedere quelli dell'avversario. In ogni turno del gioco interverrà un "arbitro" a stabilire la legalità della mossa (ovvero se il giocatore pone la pedina in una casella già occupata dall'avversario) e ad assegnare la vittoria qualora venissero incasellate tre pedine consecutive.

Ludii, la piattaforma scelta per la realizzazione del programma, è un General Game Player (GGP) sviluppato dall'università di Maastricht nell'ambito del Digital Ludeme Project (DLP). Questo progetto ha l'obiettivo di permettere di ricostruire ed analizzare oltre 1000 giochi di strategia tradizionali usando tecniche moderne, tra cui l'Intelligenza Artificiale.

Il General Game Playing è un ambito di ricerca che consiste nella progettazione e realizzazione di sistemi di intelligenza artificiale in grado di giocare con successo a più di un gioco, a partire da una descrizione formale del gioco stesso, senza che vi sia intervento umano nello sviluppo di un metodo o una strategia di gioco.

Indice

1 Introduzione **5**

2 General Game Players **7**

2.1 Definizione di un General Game Player 7

2.2 Game Description 8

2.2.1 Game Description Languages 9

2.3 General Game Players 11

2.3.1 Monte Carlo Tree Search 12

2.3.2 Proof-Number Search 15

2.3.3 Transfer Learning 17

2.3.4 Game Independent Feature Learning 18

3 Ludii **19**

3.1 Introduzione 19

3.2 Digital Ludeme Project 19

3.2.1 Modellare i giochi 20

3.2.2 Ricostruzione dei giochi 21

3.2.3 Mappare lo sviluppo 22

3.3 Struttura Ludemes 23

3.3.1 Ludeme Clusters 24

3.4 Euristiche 25

4 Implementazione di Tic-Tac-Toe ad informazione incompleta in Ludii **29**

4.1 Giochi ad informazione incompleta 29

4.1.1 Giochi ad informazione incompleta in Ludii 30

4.2 Tic-Tac-Toe 31

4.3 Tic Tac Toe ad informazione incompleta 32

4.4 Dettagli implementativi 33

5 Conclusione **37**

A Codice **43**

Capitolo 1

Introduzione

Il campo delle Intelligenze Artificiali (IA) è sempre stato profondamente connesso con i giochi. I giochi, infatti, offrono un prezioso banco di prova per le IA in quanto le pongono in contesti complessi e articolati in cui, al fine di ottenere risultati migliori, hanno bisogno di imparare e adattarsi a situazioni ed input sempre diversi. Non a caso l'esigenza che ha portato alla nascita dei GGP è esattamente questa: testare l'efficienza delle IA in ambienti nuovi, dinamici e realistici. I GGP infatti sono strumenti molto versatili e utili per una ricerca scientifica poiché offrono dati originali da analizzare in fase di progettazione dell'agente del gioco, pensando a strumenti specifici per far fronte alle particolari sfide che il singolo gioco pone [11], a differenza degli agenti nati per eccellere in giochi specifici le cui analisi vengono fatte a monte.

La tesi si svilupperà seguendo la seguente struttura:

- Nel capitolo 2 verrà introdotto la nozione di General Game Player, spiegando cosa sono, come nascono, quali sono gli algoritmi e i linguaggi che li implementano.

- Nel capitolo 3 parleremo di uno dei General Game Player introdotti nel capitolo precedente: Ludii. Parleremo del progetto e degli obiettivi

che hanno portato alla sua nascita, di come viene implementato e di quali strategie utilizza per essere efficace.

- Nel capitolo 4 verrà illustrata l'implementazione di Tic-Tac-Toe ad informazione incompleta in Ludii. Oltre alle specifiche dell'implementazione nel linguaggio, si parlerà di cosa sono i giochi ad informazione incompleta, come sono implementati in Ludii e quali giochi sono già ora presenti nel database della piattaforma.

- Nel capitolo 5 riassumiamo i risultati ottenuti.

Capitolo 2

General Game Players

In questo capitolo faremo una panoramica su cosa sono i GGP e quale è il loro ruolo nella ricerca nel campo delle IA.

2.1 Definizione di un General Game Player

Un General Game Player (GGP) è un sistema o agente artificiale in grado di giocare e competere in diversi tipi di giochi, senza bisogno di essere specificamente programmato per un gioco in particolare. In altre parole, è un approccio di intelligenza artificiale che mira a creare un sistema adattabile e flessibile, capace di affrontare una vasta gamma di giochi diversi, inclusi giochi da tavolo, giochi di carte, giochi di strategia, e molti altri.

A differenza dei tradizionali programmi di gioco che sono sviluppati per giocare al meglio ad un singolo gioco (come accade per giochi come il go o gli scacchi), un GGP sfrutta metodi di ragionamento e strategie generali che gli consentono di apprendere e adattarsi a nuovi giochi in modo automatico. Questo approccio spesso si basa su nozioni di intelligenza artificiale quali l'apprendimento automatico, la logica, la teoria dei giochi e l'elaborazione del linguaggio naturale.

Il General Game Player viene utilizzato in competizioni e sfide di giochi, come International General Game Playing Competition (IGGPC) [11], dove i sistemi GGP vengono messi alla prova contro altri agenti per valutare le loro capacità e prestazioni in un ambiente competitivo e dinamico. L'obiettivo è quello di sviluppare sistemi di gioco altamente adattabili e in grado di affrontare una vasta gamma di giochi con prestazioni competitive.

2.2 Game Description

Ogni GGP, al fine di avere la versatilità che lo porta a poter giocare ad una molteplicità di giochi, ha bisogno di una descrizione formale e coerente dei giochi stessi.

La classe di giochi considerata dai GGP è una classe finita, tale che, qualsiasi sia la situazione del gioco nella quale uno dei giocatori si trova, l'insieme delle mosse possibili è sempre finito [11].

Da queste premesse è possibile vedere il gioco come un grafo, in cui ad ogni nodo corrisponde uno stato del gioco, e ad ogni arco corrisponde una transizione da uno stato all'altro, cioè una mossa.

Nel costruire un grafo di tal fatta per descrivere un gioco, avremo un sottoinsieme dei nodi che corrisponde agli stati iniziali del gioco e un altro sottoinsieme che corrisponde agli stati finali.

Più formalmente possiamo descrivere il grafo corrispondente ad un gioco come una quintupla in cui [11]:

- S è l'insieme degli stati del gioco.

- $r_1...r_n$ sono gli n ruoli in una partita a n giocatori.

- $I_1...I_n$ n insiemi di azioni, un insieme per ogni ruolo.

- $l_1...l_n$ tali che ogni $l_i \subseteq I_i \times S$ rappresentano l'insieme di azioni legali da un certo stato.

- n è una funzione di aggiornamento t.c.:

$I_i \times ... \times I_n \times S \to S$

- $s_1 \in S$ è lo stato iniziale - $g_1, ..., g_n$ in cui ogni $g_i \subseteq S \times [0...100]$ - $t \subseteq S$ l'insieme degli stati terminali del gioco.

Il gioco inizia in uno stato s_1, uno degli stati iniziali. Ogni giocatore r fa una mossa $m_i \in I_i$ tale che $l_i(m_i, s_1)$ sia vero. Il gioco quindi arriva allo stato $n(m_1, ..., m_n, s_1)$. Questo continua fino a che il gioco entra in uno stato terminale $s \in S$ tale che $t(s)$ è vero, portando a termine la partita. Il punteggio della partita del giocatore r_i è dato da $g_i(s, value)$.

2.2.1 Game Description Languages

Descrivere i giochi attraverso i grafi però, nonostante permetta una rappresentazione finita (nel numero degli stati), può non risultare ottimale in quanto le dimensioni di ciascun grafo possono diventare significative. Ad esempio, per descrivere un gioco da tavola classico come gli scacchi, sarebbero richiesti indicativamente 10^{30} stati [11].

Nasce da qui la necessità di creare modelli che possano descrivere i giochi in maniera più compatta ed efficiente, pur mantenendone l'espressività.

Sono stati quindi sviluppati diversi linguaggi per descrivere i giochi in diversi sistemi di GGP, che di seguito illustreremo:

1. **GDL**: è il linguaggio proposto dallo Stanford Logic Group, è uno dei primi linguaggi formulati per descrivere i giochi in un GGP ed è considerato uno degli standard nella ricerca accademica in materia. Il linguaggio usa clausole della logica del primo ordine per definire

formalmente ciascun gioco. La descrizione di un gioco in GDL consiste in un insieme di clausole della logica del primo ordine che descrivono ogni stato del gioco. Vi sono infatti clausole per descrivere gli stati iniziali e finali, la legalità delle mosse, i giocatori e ogni altro aspetto rilevante del gioco.

Per GDL è possibile dimostrare che è in grado di definire giochi ad informazione incompleta [11]. GDL, nonostante l'espressività e la versatilità, non è molto efficiente [17]: il codice dei giochi non può essere facilmente riusato in quanto la descrizione a clausole dei giochi e la verbosità della descrizione di alcune dinamiche non lo rendono di facile uso o lettura. Questi aspetti rendono la scrittura di giochi per GDL complessa e lunga.

2. **RBG**: (Regular BoardGames) è un sistema di GGP in cui i giochi vengono codificati tramite un linguaggio regolare [25]. L'implementazione prevede due linguaggi: uno a basso livello che viene usato come input per i programmi ad alta efficienza; l'altro, più ad alto livello, che rende i programmi leggibili. Queste specificità del linguaggio lo rendono sia più fruibile rispetto a GDL grazie alle descrizioni facilmente comprensibili ad alto livello, sia molto più efficiente, con risultati di efficienza fino a 50 volte superiori. Il sistema, inoltre, supporta tutti i giochi ad informazione completa, fatta eccezione per quelli con mosse simultanee.

3. **Ludii**: è un GGP con un' amplissima espressività, in quanto è in grado di modellare tutti i giochi a informazione incompleta [19], i giochi senza tabellone, oltre a tutti i giochi modellabili da GDL. Il linguag-

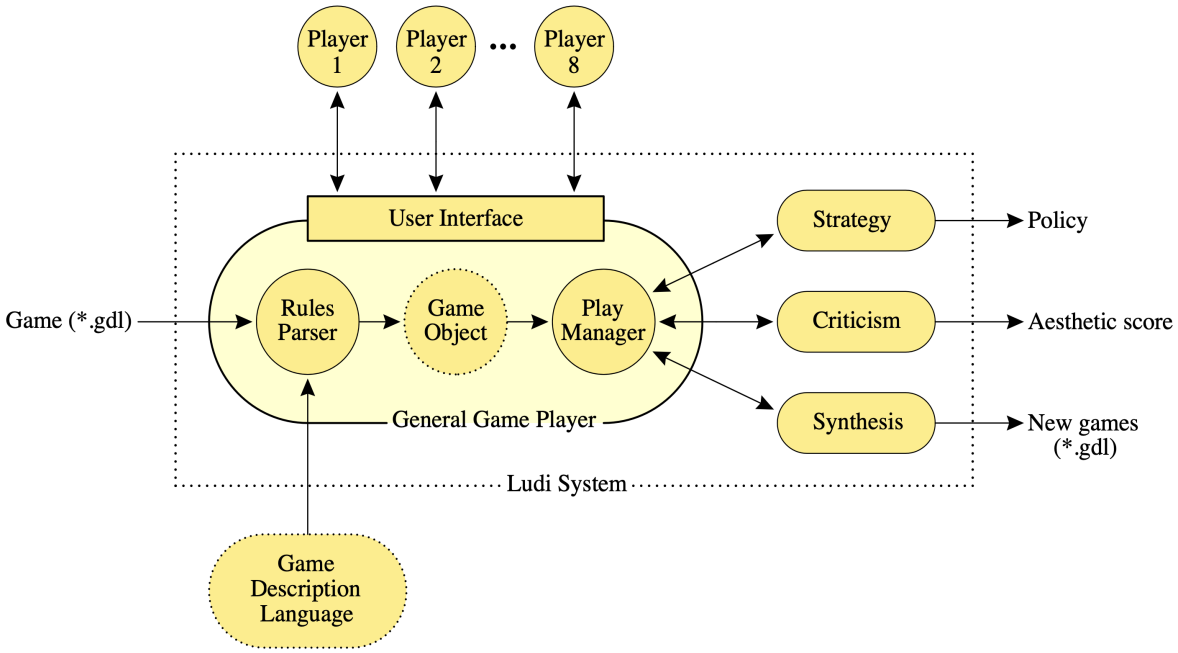

Figura 2.1: **Architettura originale del sistema Ludii [5]**

gio di Ludii si basa sul concetto di Ludeme, "una decomposizione del concetto di gioco in unità concettuali di informazioni relative al gioco" [17], che nel concreto si traduce in una descrizione dei giochi che risulta più compatta e comprensibile rispetto a GDL o altri sistemi più antiquati. Inoltre, anche sotto l'aspetto dell'efficienza, Ludii si è dimostrato un sistema molto efficace in quanto è stato dimostrato che è in grado di elaborare i giochi in maniera significativamente più veloce rispetto a linguaggi come GDL [16] .

2.3 General Game Players

L'obiettivo primario per un GGP consiste nella realizzazione di un programma in grado di comprendere le regole per giocare a giochi sconosciuti fino all'istante prima della compilazione; il tutto senza alcun intervento umano, in contesti in cui la conoscenza di regole, euristiche e strategie è interamente vincolata alla descrizione formale del gioco stesso [17].

La ricerca in quest'ambito è stata arricchita da molte competizioni che mirano allo sviluppo di GGP sempre migliori, come la AAAI Competition [11] o la IGGP Competition [20], che nel tempo hanno portato alla definizione di numerose tecniche per lo sviluppo degli agenti ludici, ognuno con i suoi pro e i suoi contro. Di seguito faremo quindi una rapida e generale rassegna dei metodi più diffusi ed efficaci.

2.3.1 Monte Carlo Tree Search

Tra le numerose strategie emerse dalle competizioni sopra citate, l'approccio che risulta essere più diffuso ed efficiente risulta essere quello che utilizza Monte Carlo Tree Search (MCTS), un algoritmo euristico basato sul metodo Monte Carlo.

MCTS è un metodo di ricerca best-first che non richiede una funzione di valutazione posizionale ed è basato su esplorazioni randomizzate dell'albero di ricerca. Usando i risultati delle esplorazioni precedenti infatti l'algoritmo gradualmente costruisce un albero di gioco che diventa più accurato ad ogni iterazione, stimando i valori delle mosse più promettenti. Il funzionamento di MCTS si sviluppa in 4 fasi, ripetute iterativamente fino a che si ha tempo a disposizione [8]:

1. **Selezione**: come primo passo viene selezionato, in base alle informazioni raccolte dalle precedenti iterazioni, un ramo dalla radice. Il passo di selezione controlla l'equilibrio tra sfruttamento ed esplorazione del ramo. Da una parte quindi si occupa di selezionare ed analizzare la mossa che in quel momento è ritenuta la più promettente (sfruttamento), dall'altra deve necessariamente esplorare le mosse meno promettenti (o non ancora provate) a causa dell'incertezza del-

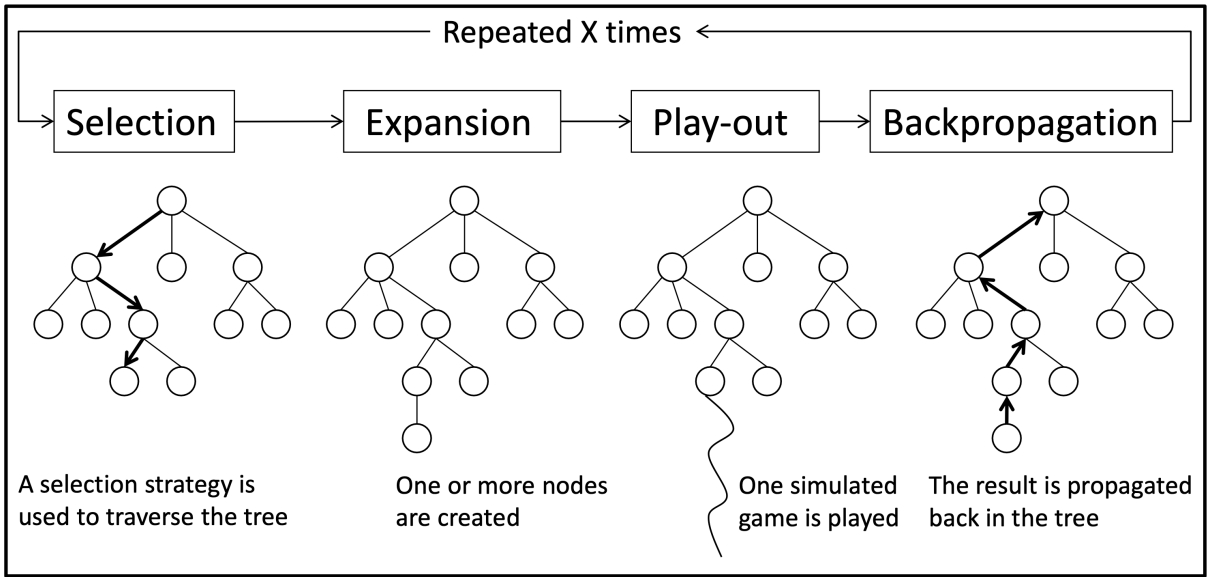

Figura 2.2: **Schema di ricerca di MCTS.**

l'esito delle simulazioni [9]. Numerose strategie per la selezione [7] sono state suggerite per MCTS, ma la più ampiamente usata è basata sull'algoritmo UCB1 [1] ed è chiamata UCT (Upper Confidence Bounds applied to Trees) [14]. UCT opera come segue.

Sia I l'insieme di nodi immediatamente raggiungibili dal nodo p. La strategia di selezione sceglie il figlio b del nodo p che soddisfa la formula di seguito.

$$b \in argmax_{i \in I}(v_i + C \times \sqrt{\frac{\ln n_p}{n_i}})$$

dove v_i è il valore del nodo i, n_i è il numero di visite di i, e n_p è il numero di visite di p. C è un parametro costante, che può essere modificato sperimentalmente. Nel caso di un pareggio tra i valori di due rami ne viene scelto uno randomicamente. Questo processo è ripetuto finché non viene trovato un nodo che non è stato completamente espanso [9].

2. **Espansione**: come detto prima il passo di selezione continua finché non è stato trovato un nodo i cui figli non sono stati tutti espansi. Tra i figli che non sono salvati nell'albero, ne viene selezionato uno in maniera casuale. Questo nodo L viene poi aggiunto come una nuova foglia e viene successivamente investigato.

3. **Simulazione**: dal nodo foglia viene eseguita la simulazione. Le mosse sono selezionate in auto-gioco finché non si arriva alla fine della partita. Questo passo consiste nel giocare con mosse totalmente casuali o, semi-casuali scelte seguendo una strategia di simulazione.

4. **Backpropagation**: nel passo finale, il risultato R di una simulazione k viene retropropagato dal nodo foglia L, passando attraverso tutti i nodi attraversati precedentemente, fino a giungere al nodo radice. Il risultato viene valutato positivamente ($R_k = +1$) se la partita viene vinta, negativamente ($R_k = -1$) se la partita viene persa. Il pareggio porta ad un risultato $R_k = 0$. Una strategia di retropropagazione viene applicata al valore v_i di un nodo i. Il risultato è quindi calcolato prendendo la media dei risultati di tutte le simulazioni fatte attraverso questo nodo (cita 1), $v_i = (\sum_k R_k)/n_i$.

Una delle ragioni per cui MCTS è tra i metodi più utilizzati nell'ambito del GGP consiste nel fatto che è in grado di operare su qualsiasi gioco, senza alcuna conoscenza pregressa né del gioco in se, né delle sue dinamiche specifiche, rendendolo estremamente versatile [22]. Questa sua stessa caratteristica però può essere anche considerata un limite se si considera il fatto che, anche nei casi in cui alcuni metodi euristici sono noti, MCTS non ne fa uso.

2.3.2 Proof-Number Search

Proof Number Search (PNS) è un algoritmo di ricerca per alberi di gioco che usa un approccio best-first. L'algoritmo è molto indicato per risolvere obiettivi binari, come dimostrare una vittoria o una sconfitta a partire da un certo stato del gioco.

Nel caso di un GGP un algoritmo come PNS è utile per portare il giocatore ad una vittoria forzata a partire da uno stato. I nodi dell'albero che l'algoritmo crea hanno tre valori possibili: vero, falso e indeterminato. In caso di una vittoria l'albero è provato e il suo valore è vero; nel caso di una sconfitta o di un pareggio l'albero è smentito e il suo valore è falso; il valore dell'albero è indeterminato negli altri casi. L'algoritmo procede ad espandere sempre il nodo più promettente fino a che il valore della radice dell'albero rimane indeterminato. Così come MCTS poi, PNS non necessita di funzioni euristiche particolari per determinare quale è il nodo più promettente da espandere: questo nodo in PNS è chiamato 'most-proving'. Questo nodo viene selezionato seguendo due criteri: la forma dell'albero di ricerca (il fattore di ramificazione di ogni nodo interno) e i valori delle foglie. Questi due criteri consentono a PNS di trattare in modo efficiente alberi di gioco con un fattore di ramificazione non uniforme [9].

Nella Figura 1.2 è mostrato un albero AND/OR, la struttura dati di riferimento dell'algoritmo PNS. In questo albero sono rappresentati come quadrati i nodi OR e come cerchi i nodi AND. Accanto ad ogni nodo troviamo due numeri: quello posto come apice è il numero di prova (pn) che rappresenta il minimo numero di foglie che devono essere dimostrate al fine di provare il nodo come vincente; come pedice invece è posto il

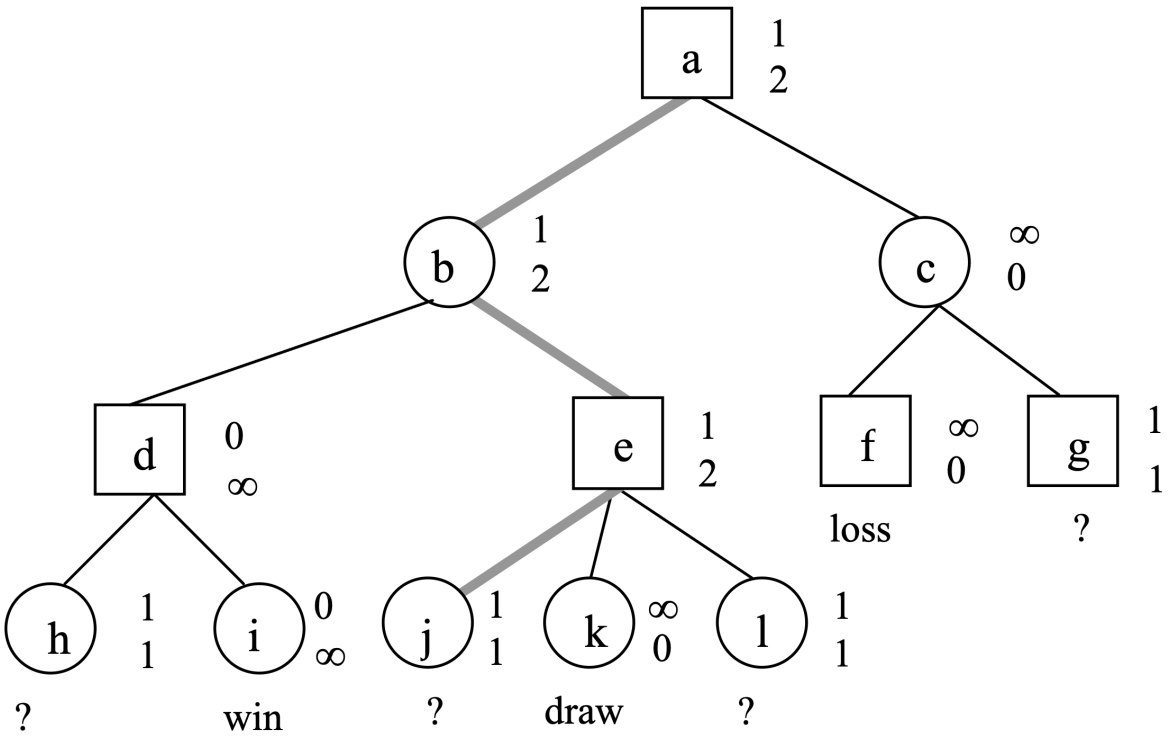

Figura 2.3: **Un albero AND/OR con numeri di prova e smentita.**

numero di smentita (*dpn*, dall'inglese disproof number) che rappresenta il minimo numero di foglie che devono essere dimostrate al fine di provare il nodo come perdente. Dal momento che l'obiettivo dell'algoritmo consiste nel provare una vittoria forzata, consideriamo i nodi vincenti come provati. Un nodo è vincente se ha $pn = 0$ e $dpn = \infty$. I nodi che portano a sconfitte o pareggi sono considerati smentiti. Questi nodi hanno $pn = 0$ e $dpn = 1$. I nodi indeterminati hanno $pn = 1$ e $dpn = 1$.

Un nodo interno OR ha *pn* uguale al minimo del numero di prova dei suoi figli, visto che per provare un nodo OR è sufficiente provare un figlio. Il *dpn* di un nodo interno OR invece è uguale alla somma del numero di smentita di tutti i suoi figli, considerato che per smentire un nodo è necessario smentirli tutti. Un nodo interno AND invece ha *pn* uguale alla somma dei suoi figli e il *dpn* uguale al minimo numero di smentita dei

suoi figli. La procedura per scegliere il nodo 'most proving' parte dalla radice, ad ogni nodo OR sceglie il figlio con il *pn* più piccolo e ad ogni AND sceglie il figlio con il minor *dpn*; una volta raggiunta una foglia questa viene espansa (di conseguenza la foglia diventa un nodo interno) e vengono valutati i suoi figli appena creati. Questo processo è chiamato 'immediate evaluation' [9].

In [9] è inoltre mostrato un approccio che combina MCTS e PNS, chiamato PN-MCTS, che combina i punti di forza dei due algoritmi integrando il meccanismo di prova e smentita nell'algoritmo UCT che viene usato nella fase di selezione di MCTS. Questo approccio ha un costo computazionale leggermente superiore a quello del solo MCTS, ma nei test su diversi giochi ha mostrato percentuali di vittoria maggiori.

2.3.3 Transfer Learning

Il Transfer Learning è una tecnica utilizzata nell'ambito del Machine Learning in cui la conoscenza di una nozione appresa in un determinato contesto viene riutilizzata per migliorare i risultati di un concetto diverso, ma che presenta caratteristiche simili [24]. Nell'ambito dei GGP questa tecnica viene utilizzata per trasferire le nozioni apprese da un dominio (come euristiche o funzioni obiettivo) ad altro dominio che presenta caratteristiche simili nella struttura degli stati. Nel concreto consiste nel trasferimento dei pesi e/o le rappresentazioni nella rete neurale.

Il Transfer Learning presenta un approccio più umano all'apprendimento in quanto, così come gli umani tendono a riutilizzare in ambiti diverse le conoscenze che hanno appreso, anche qui si tenta di ricostruire questa dinamica, trasferendo la conoscenza di una rete su un altra. Questo ap-

proccio è infatti favorito da quei sistemi di GGP che piuttosto che puntare a performance ottimali, puntano ad imitare lo stile di gioco umano [20].

Se però da una parte il TL offre un modo di operare interessante per via del riutilizzo delle conoscenze apprese, dall'altra presenta però numerose limitazioni a causa della sua scarsa flessibilità ed espandibilità [22].

2.3.4 Game Independent Feature Learning

L'idea alla base del Game Independent Feature Learning (GIFL) consiste nell'eseguire simulazioni casuali dei giochi così da costruire una struttura ad albero a partire dagli stati terminali dei giochi una volta terminata l'esecuzione. Nel gioco vengono poi identificate delle caratteristiche offensive (nel caso in cui queste portano alla vittoria) o difensive (nel caso in cui prevengono la sconfitta). Queste caratteristiche riempiono quindi il database per orientare poi le simulazioni di gioco [22].

Capitolo 3

Ludii

3.1 Introduzione

Ludii è un General Game System sviluppato nell'ambito del Digital Ludeme Project, progetto dell'università di Maastricht volto a modellare, giocare ed analizzare un ampia gamma di giochi tradizionali e non. La caratteristica che distingue Ludii da altri modelli di GGP consiste nella struttura basata sui Ludeme, unità sintattiche che racchiudono concetti di gioco che vengono usate per creare un linguaggio per il General Game Player facilmente modellabile, efficiente e leggibile.

3.2 Digital Ludeme Project

Il Digital Ludeme Project (DLP) è un progetto condotto dall'università di Maastricht nel periodo 2018-2023 e finanziato dal Consiglio europeo della ricerca (European Research Council).

Gli obiettivi del progetto consistono in:

1. Modellare un ampio spettro di giochi di strategia tradizionali per renderli disponibili e giocabili tramite un unico database online.

Figura 3.1: **Esempi di come appaiono alcuni giochi nella libreria dei giochi di Ludii [6]**

2. Ricostruire la conoscenza mancante riguardo i giochi di strategia tradizionale con un livello di precisione senza precedenti.

3. Mappare lo sviluppo di giochi di strategia tradizionali ed esplorare il loro ruolo nello sviluppo della cultura umana e nella diffusione di idee matematiche.

Il progetto si pone, infine, come ulteriore obiettivo quello di produrre un 'albero genealogico' di tutti i giochi di strategia tradizionali del mondo, grazie al quale ricostruire la diffusione dei giochi e delle relative idee matematiche.

3.2.1 Modellare i giochi

Il modello di riferimento usato nel DLP per dare una forma struturata ai giochi è il Ludeme. I Ludemes sono unità concettuali di informazioni rela-

tive al gioco [15]. Questi sono i concetti ad alto livello che i programmatori usano per sviluppare i giochi per la piattaforma, distinguendo facilmente tra la forma (regole ed attrezzatura) e la funzione (il comportamento che emerge durante il gioco). La divisione del gioco in Ludemes permette di spezzare il gioco in diverse unità che sono facili da manipolare, da leggere e offiono ampia flessibilità permettendo di ottenere l'ampio database di giochi eseguibili che viene offerto.

Il concetto di Ludeme e lo sviluppo di giochi intorno ad esso è la base per il General Game System Ludii, nato per giocare, valutare ed ottimizzare un ampio numero di giochi da tavolo di ogni epoca. Per farlo Ludii non usa approcci di intelligenza artificiale volti a creare un intelligenza superumana che massimizza l'output di ogni mossa, ma cerca di ricreare quanto più possibile un agente che giochi in maniera più umana possibile. Non si affida quindi a metodi che coniugano MCTS e deep learning come avviene per agenti specifici ad esempio per il Go [18], bensì viene usato il solo MCTS [7] con esercitazioni influenzate dalle caratteristiche apprese attraverso l'auto-gioco [4].

3.2.2 Ricostruzione dei giochi

Nel ricostruire la conoscenza mancante nei giochi storici, gli obiettivi principali consistono nel massimizzare, per ogni gioco sia l'autenticità storica, sia la qualità delle partite e la relativa giocabilità [4].

Per tracciare la diffusione dei tradizionali giochi di strategia è utile inserire il loro sviluppo all'interno di un framework di evoluzione genetica biologica [4]. I Ludemes vengono visti quindi come il DNA di ciascun gioco, con l'approccio basato su questi che ha mostrato di essere un modello

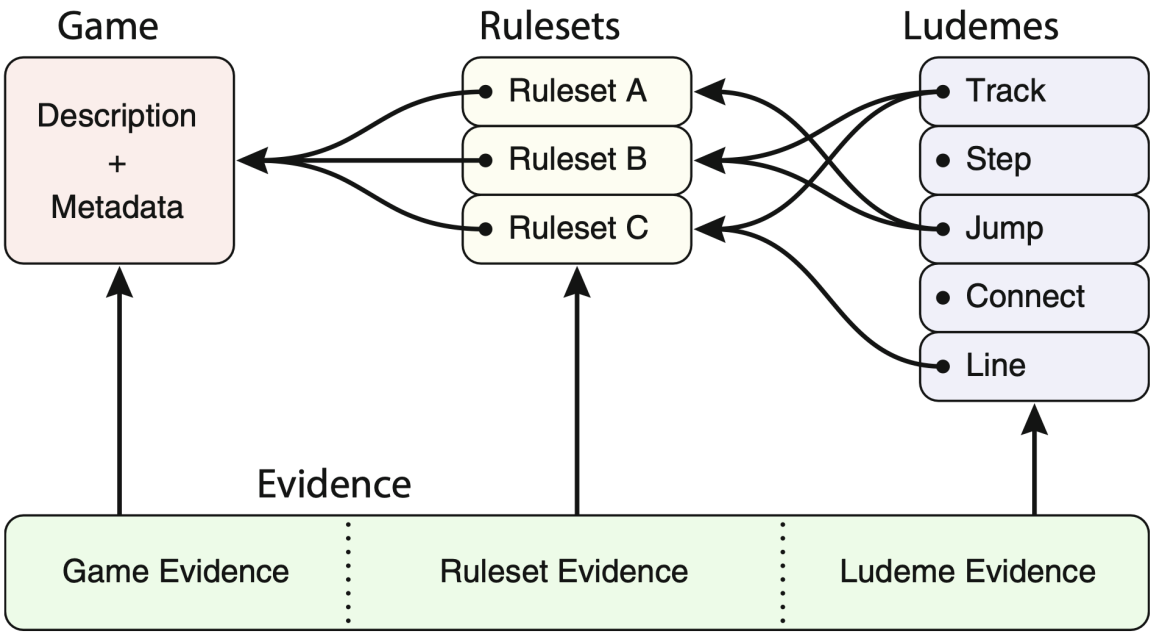

Figura 3.2: **Panoramica dei principali gruppi del database dei giochi Ludii e delle loro relazioni [6].**

valido per per mappare l'evoluzione dei giochi [2].

Una volta che un framework genetico è stato stabilito, vengono utilizzate tecniche della filogenetica computazionale, come quelle utilizzate per creare alberi filogenetici che mappano la diffusione dei linguaggi umani [12]. Queste tecniche permettono la ricostruzione degli stati ancestrali, permettono sia di stimare le probabilità che determinati tratti compaiano in giochi più antichi, sia di stimare l'impatto di possibili anelli mancanti tra i vari giochi, nella forma di giochi sconosciuti che vengono suggeriti e dedotti dai record filogenetici per i quali non ci sono prove.

3.2.3 Mappare lo sviluppo

Ad ogni classe di Ludeme vengono assegnati dei tag che permettono di classificare sia gli aspetti matematici che vengono rappresentati da ogni Ludeme, sia quelli storici e culturali (dove e quando venivano giocati).

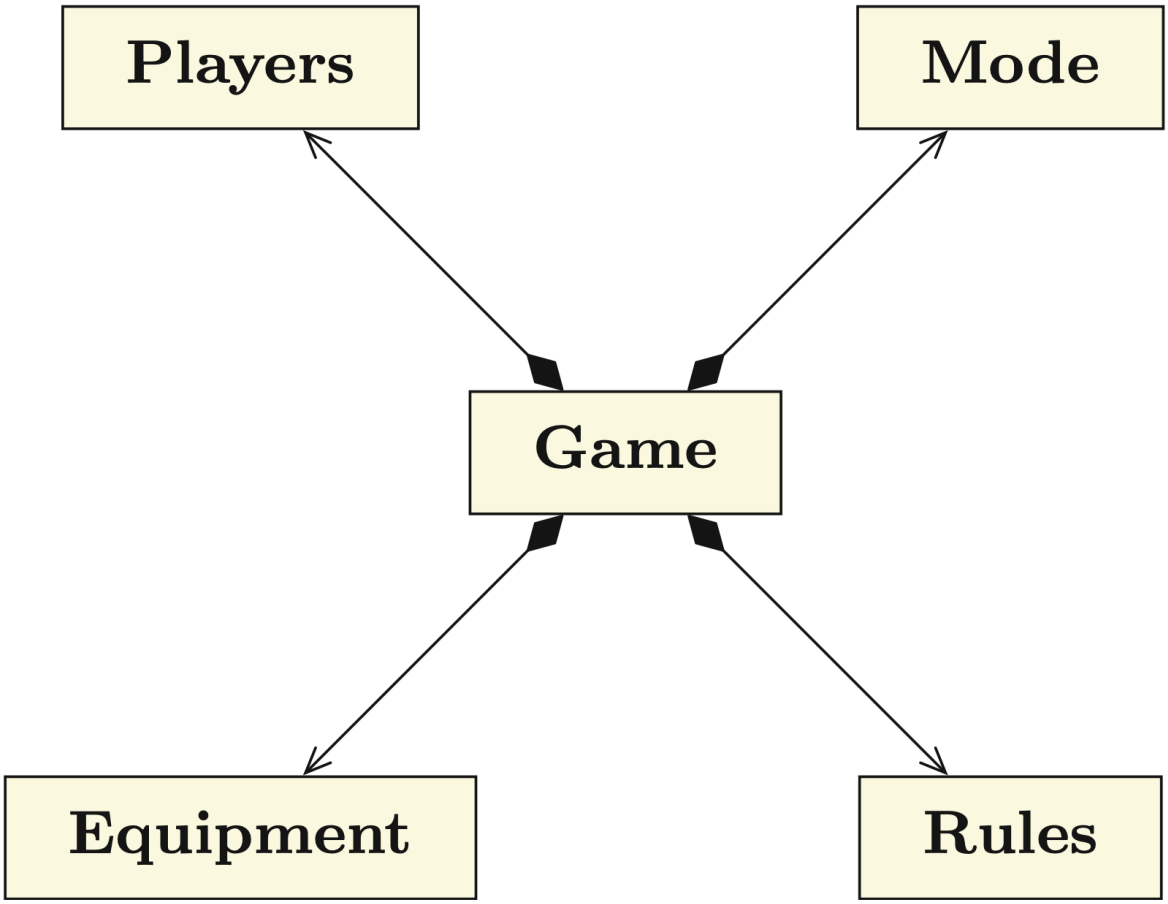

Figura 3.3: **Le componenti di un gioco in Ludii [6].**

Questi metadati confluiscono in un database che permette di creare grafi di conoscenza che forniscono modelli probabilistici [10] delle relazioni tra le loro dimensioni geografiche, storiche e matematiche [2].

3.3 Struttura Ludemes

In Ludii i giochi sono descritti in termini di distinti Ludemes. Ogni Ludeme rappresenta una particolare nozione del gioco che permette di descrivere strutture più complesse e ad alto livello e possono essere usati sia per rappresentare sia aspetti fisici del gioco (come la forma, la funzione delle pedine o la forma della tavola), sia l'insieme delle mosse.

A titolo esemplificativo di come viene rappresentato un gioco in Ludii,

di seguito vi è il codice di Tic-Tac-Toe:

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

Per ogni descrizione di un gioco il numero dei Ludemes varia e si può passare da una dozzina per giochi molto semplici fino a migliaia per giochi complessi che prevedono molti casi specifici [21]. Questo approccio racchiude concetti chiave dei giochi e conferisce loro etichette utili e significative [25]. Grazie a questa proprietà è possibile usare un approccio basato sull'uso delle classi nella grammatica [3] per derivare il linguaggio direttamente dalla gerarchia delle classi del codice sorgente delle librerie usate nel sistema.

3.3.1 Ludeme Clusters

La figura 2.1 mostra un grafico che riproduce la distribuzione dei giochi in Ludii in base al tipo di Ludeme utilizzato nella game description. Il grafico è realizzato applicando il t-distributed Stochastic Neighbor Embedding (t-SNE) [23] per ridurre il dataset di Ludeme a due dimensioni. Nel grafico sono rappresentati 3 cluster: Il numero 1 (arancione) contiene 88 giochi, il

numero 2 (verde) ne contiene 85, mentre il numero 3 (rosso) ne contiene 522.

Il cluster arancione contiene giochi contenenti il Ludeme Sow, un Ludeme appartenente alla famiglia di giochi del "Mancala". Questi giochi sono tutti contraddistinti da uno stile di gioco unico in cui i giocatori ad ogni turno "seminano" pezzi lungo un anello di buchi, spesso con molto semi locati in un singolo spazio.

Il cluster verde contiene giochi che contengono i Ludemes Track e Dice. Questa è una combinazione comune di Ludemes per quella famiglia di giochi come Snakes & Ladders e Backgammon. Il Ludeme Dice indica che i dadi sono parte delle regole del gioco e che, di conseguenza, vi è una parte di fortuna inevitabilmente coinvolta nello svolgimento della partita. Il Ludeme Track indica che ci sono alcuni percorsi prestabiliti nel gioco che devono essere necessariamente seguiti, limitando quindi la libertà di movimento di ogni pezzo. Questi due Ludeme indicano quindi che il gioco ha uno spazio di interazione libertà limitato, e di conseguenza il risultato del turno di ciascun giocatore è il risultato spesso più della fortuna che dell'abilità.

Il cluster rosso contiene tutti i giochi che non appartengono a nessuno dei due cluster precedenti, comprendendo sia i giochi senza nè Track nè Dice, sia quelli che ne contengono solo uno dei due. [21]

3.4 Euristiche

Ludii contiene numerose euristiche che possono essere utilizzate per funzioni di valutazione euristica per algoritmi di gioco che ne hanno esplicito bisogno, come $\alpha\beta - search$ [13]. Nella Figura 2.2 sono elencate numerose

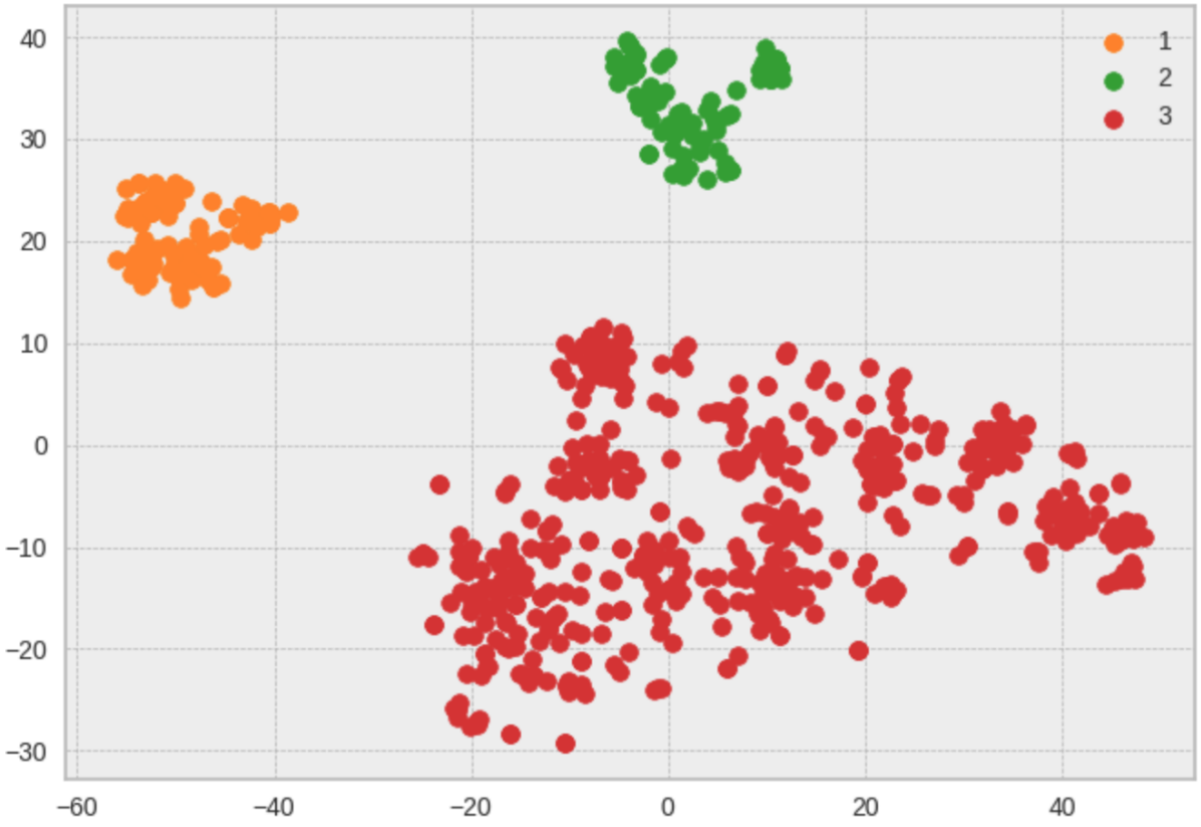

Figura 3.4: **Esempio di cluster di giochi basati sul dataset di Ludemes. I punti sono stati ridotti a due dimensioni utilizzando t-SNE.**

Heuristic	Description	Avg. Win % (# top performances) Positive	Negative
Material	Sum of owned pieces.	62.09% (181)	39.88% (12)
Mobility	Number of legal moves.	57.75% (65)	43.58% (13)
Influence	Number of legal moves with distinct destination positions.	57.59% (53)	44.70% (6)
CornerProximity	Sum of owned pieces, weighted by proximity to nearest corner.	57.27% (44)	40.92% (9)
SidesProximity	Sum of owned pieces, weighted by proximity to nearest side.	56.36% (31)	43.86% (14)
LineCompletion	Measure of potential to complete line(s) of owned pieces.	54.86% (73)	44.28% (3)
CentreProximity	Sum of owned pieces, weighted by proximity to centre.	54.53% (37)	44.59% (13)
RegionProximity	Sum of owned pieces, weighted by proximity to a predefined region.	53.07% (21)	47.60% (7)
OwnRegionsCount	Sum of (piece) counts in owned regions.	50.35% (19)	47.66% (10)
PlayerRegionsProximity	Sum of owned pieces, weighted by proximity to owned region(s).	50.01% (2)	49.50% (2)
PlayerSiteMapCount	Sum of (piece) counts in sites mapped to by player ID.	49.99% (18)	47.86% (2)
Score	Score variable of game state corresponding to player.	49.83% (17)	47.39% (0)
ComponentValues	Sum of values of sites occupied by owned pieces.	48.68% (0)	48.39% (0)
Null	Always returns a value of 0.	48.52% (0)	-

Figura 3.5: **Euristiche implementate in Ludii. La penultima e l'ultima colonna mostrano le percentuali di vittoria delle euristiche in media su tutte le partite (e il numero di partite in cui un'euristica è l'unica a ottenere il massimo risultato), per i casi in cui all'euristica viene assegnato un peso positivo o negativo, rispettivamente.**

euristiche implementate in Ludii: le prime due colonne mostrano nomi e brevi descrizioni per ciascuna di esse; per ogni euristica viene inclusa una variante con pesi positivi e una con pesi negativi (ad esempio l'euristica "Material" con pesi positivi indica che nel gioco è preferibile avere molti pezzi, mentre con pesi negativi indica che sono preferibili stati della partita in cui il giocatore possiede meno pezzi).

In [21] è stata testata per ciascuno dei 695 giochi in Ludii l'utilità di ciascuna euristica per quel gioco particolare. I risultati, elencati nella Figura 2.2, sono ottenuti misurando le percentuali medie di vittoria degli agenti che usano $\alpha\beta - search$ con ogni euristica contro le combinazioni di tutte le altre euristiche. Dato quindi un gioco di n-giocatori con k euristiche applicabili, per ogni euristica h, generiamo tutte le combinazioni di $n-1$ euristiche con le altre $k-1$ come avversarie. Se ci sono più di 10 combinazioni di euristiche che rispettano queste specifiche, allora ne scegliamo casualmente 10 da considerare. Per ogni euristica viene giocato

un minimo di 10 partite contro ogni combinazione di avversari.

I risultati mostrano che alcune euristiche con pesi positivi (come Material, Mobility, Influence) hanno una percentuale piuttosto alta di win-rate, il che suggerisce che queste euristiche possono essere buone candidate per essere euristiche di default in giochi in cui non si ha alcuna conoscenza pregressa. Quasi tutte le euristiche (anche quelle con win-rates intorno al 50%) ottengono comunque risultati molto buoni in alcuni giochi, il che suggerisce che sono in ogni caso utili da tenere in un portfolio di euristiche per un GGP. Quando vengono usati pesi negativi però in media tutte le euristiche non offrono performance di alto livello, infatti il win-rate di ciascuna euristica mostrata in Figura 2.2 si attesta sotto il 50%, ma mostrano ottimi risultati in giochi specifici [21].

Capitolo 4

Implementazione di Tic-Tac-Toe ad informazione incompleta in Ludii

In questo capitolo verrà presentata l'implementazione di Tic-Tac-Toe ad informazione incompleta in Ludii. Essendo una versione ad informazione incompleta il gioco prevede che ciascun giocatore non potrà vedere tutti i pezzi presenti sulla tavola, ma potrà vedere solo quelli che ha posizionato, lasciando ad un "arbitro" la decisione sull'approvazione o meno della mossa. Verrà introdotto anche il concetto di gioco ad informazione incompleta e di come è stato implementato in Ludii.

4.1 Giochi ad informazione incompleta

In un gioco ad informazione incompleta i giocatori sono semplicemente all'oscuro delle scelte prese dagli altri giocatori. Tutti i giocatori hanno però conoscenza comune di chi sono gli altri giocatori, quali sono le loro possibili strategie e le preferenze / obiettivi di ciascuno di loro. Di conseguenza le informazioni riguardo gli altri giocatori sono complete. I giochi ad informazione incompleta sono quindi quelli in cui alcuni aspetti del gioco sono completamente nascosti agli altri giocatori: alcuni esempi sono i

giochi di carte come il poker in cui ogni giocatore ha coscienza solo delle carte che ha in mano e di quelle presenti sul tavolo, ma non di quelle in mano agli avversari. Nei giochi ad informazione incompleta ogni giocatore deve pianificare le mosse tenendo conto di tutte le possibilità.

4.1.1 Giochi ad informazione incompleta in Ludii

Tra le caratteristiche che rendono Ludii un GGP molto efficiente, oltre alla facilità d'uso e all'alta leggibilità del linguaggio, troviamo l'espressività che è pari a quella di GDL. In [19] infatti è dimostrato che l'espressività del linguaggio di Ludii è tale da poter rappresentare oltre a tutti i giochi finiti, arbitrari, deterministici e ad informazione perfetta, anche i giochi non deterministici e ad informazione incompleta. Nel database dei giochi disponibili e giocabili in Ludii troviamo infatti due giochi ad informazione incompleta: Phantom Go e Kriegspiel.

Phantom Go

Phantom Go è una versione del gioco Go ad informazione incompleta. Si gioca con due giocatori ed un arbitro, ogni giocatore col proprio tabellone. I due giocatori possono vedere solo le rispettive tavole sulle quali sono poggiate solo le loro pedine, mentre l'arbitro dispone di una tavola completa delle pedine di ciascuno dei due giocatori. I giocatori, prima di ogni mossa, dovranno chiedere all'arbitro se una certa intersezione è stata già riempita dall'avversario: se nel tabellone dell'arbitro questa è libera allora la pedina potrà essere inserita, altrimenti il giocatore dovrà far richiesta fino a che non trova un'intersezione libera. Nel caso di una cattura poi l'arbitro annuncia il numero di pezzi catturati e dice al giocatore quali sono

stati presi, così da aggiornare la tavola. Il gioco termina quando entrambi i giocatori passano.

Kriegspiel

Kriegspiel è una versione ad informazione incompleta del gioco degli scacchi. Le regole generali del gioco rimangono quindi uguali a quelle tradizionali scacchi, ma ciascun giocatore può vedere solo le proprie pedine, senza possibilità di vedere quelle dell'avversario. È quindi necessaria la presenza di un arbitro che possiede informazioni complete sullo sviluppo del gioco. L'arbitro può dichiarare le mosse legali o illegali: se la mossa è illegale dovrà essere ripetuta, mentre se è legale la mossa è valida. Il gioco quindi, così come avviene anche per Phantom Go si svolge su tre tavole separate: una per il giocatore bianco, una per il giocatore nero e una per l'arbitro. Ad ogni turno l'arbitro annuncia, qualora fosse necessario:

- Di chi è il turno.
- Quale pezzo è stato catturato, e in quale cella di preciso.
- Tutti i tipi di scacco.
- Ogni tentativo di mossa illegale in cui un giocatore cerca di spostare un pezzo su una casella occupata dall'avversario.

4.2 Tic-Tac-Toe

Tic-Tac-Toe (noto in Italia anche come Tris) è un gioco ad informazione perfetta che si gioca su un tabellone di 3 celle x 3 celle in due giocatori. Ogni giocatore, nel momento del suo turno, sceglie una cella vuota e disegna il proprio simbolo (di solito i due simboli più utilizzati sono una "X" o un cerchio vuoto). Vince il giocatore che riesce ad inserire una sequenza

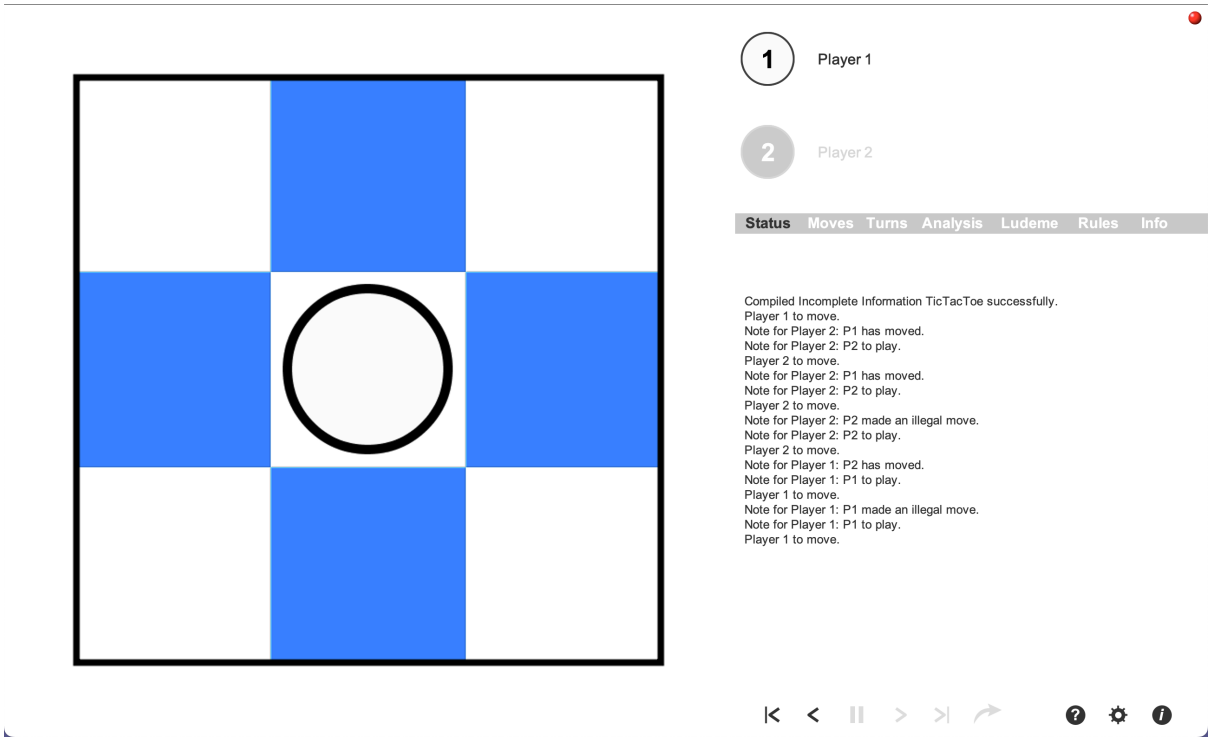

Figura 4.1: **Screenshot di una partita di Tic-Tac-Toe ad informazione incompleta in Ludii.**

ininterrotta di simboli in linea retta in 3 celle consecutive (sia in orizzontale, che in verticale che in diagonale). Qualora ogni cella del tabellone venisse riempita di simboli senza che uno dei due giocatori sia riuscito ad occupare 3 celle consecutive, allora la partita verrà terminata col risultato di parità.

Tic-Tac-Toe, inoltre, è un gioco ”risolto” in quanto sono note alcune mosse in sequenza che, se applicate, garantiscono al giocatore di non perdere la partita (cioè di pareggiare o di vincere).

4.3 Tic Tac Toe ad informazione incompleta

La versione di Tic-Tac-Toe presentata di seguito è una versione ad informazione incompleta del gioco. Nel concreto questo significa che, così come su Phantom Go, ogni giocatore dispone di una propria scacchiera in cui

sono disposti solo i propri pezzi: al momento di fare la mossa il giocatore chiederà ad un arbitro (dotato di una tavola completa dei pezzi di ciascun giocatore) se la mossa che desidera fare è legale o meno (cioè se la casella scelta per posizionare il pezzo è occupata o meno dall'avversario) fino a che non troverà una casella libera adeguata. Le regole di terminazione sono le stesse del Tic-Tac-Toe tradizionale in quanto il gioco termina solo in caso di vittoria di uno dei due giocatori o di parità.

4.4 Dettagli implementativi

Il codice è disponibile per intero nell'appendice di questo elaborato ed è stato scritto utilizzando il GDL fornito da Ludii, usando quindi il modello dei Ludemes.

Nella prima parte del codice sono definite 3 funzioni che illustro di seguito:

1. **IllegalMove**: viene chiamata quando si verifica una mossa illegale (cioè quando un giocatore pone un pezzo su una casella occupata dall'avversario) e avvisa il giocatore scrivendo un avviso nella sezione 'Status' dell'interfaccia utente di Ludii oltre a rinnovare il turno del giocatore.

2. **NotEmpty**: definisce formalmente quando la cella selezionata è una cella occupata dall'avversario, ma che il giocatore non può vedere sulla sua tavola.

3. **MadeALegalMove**: definisce che cosa è una mossa legale, ovvero la scelta da parte del giocatore di una cella non occupata né da lui né

dall'avversario. Nel farlo rende nascosto all'avversario il pezzo appena posto e aggiunge degli avvisi nella sezione 'Status' dell'interfaccia.

Dopo le funzioni prima descritte nel codice vengono definiti, con gli appositi Ludemes, il resto delle regole e delle specifiche: all'inizio viene descritto il numero di giocatori e il tipo di attrezzatura del gioco (pezzi e tipo di tavola), di seguito poi vengono descritte le regole del gioco e le condizioni di vittoria.

Nelle regole si chiede ad ogni giocatore di scegliere una cella tra l'elenco di quelle vuote: se questa è non vuota vengono chiamate le funzioni 'NonEmpty' e 'IllegalMove', altrimenti, se la casella scelta è vuota, viene chiamata la funzione 'MadeALegalMove'.

In fondo al codice sono specificati dei metadati per modificare la parte estetica del gioco per renderla più piacevole alla vista: viene specificato prima il tipo di tavola, poi i colori che la compongono.

Le differenze più vistose tra il codice presentato e quello ad informazione completa di Tic-Tac-Toe, ovviamente risiedono nell'implementazione del concetto di informazione incompleta.

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

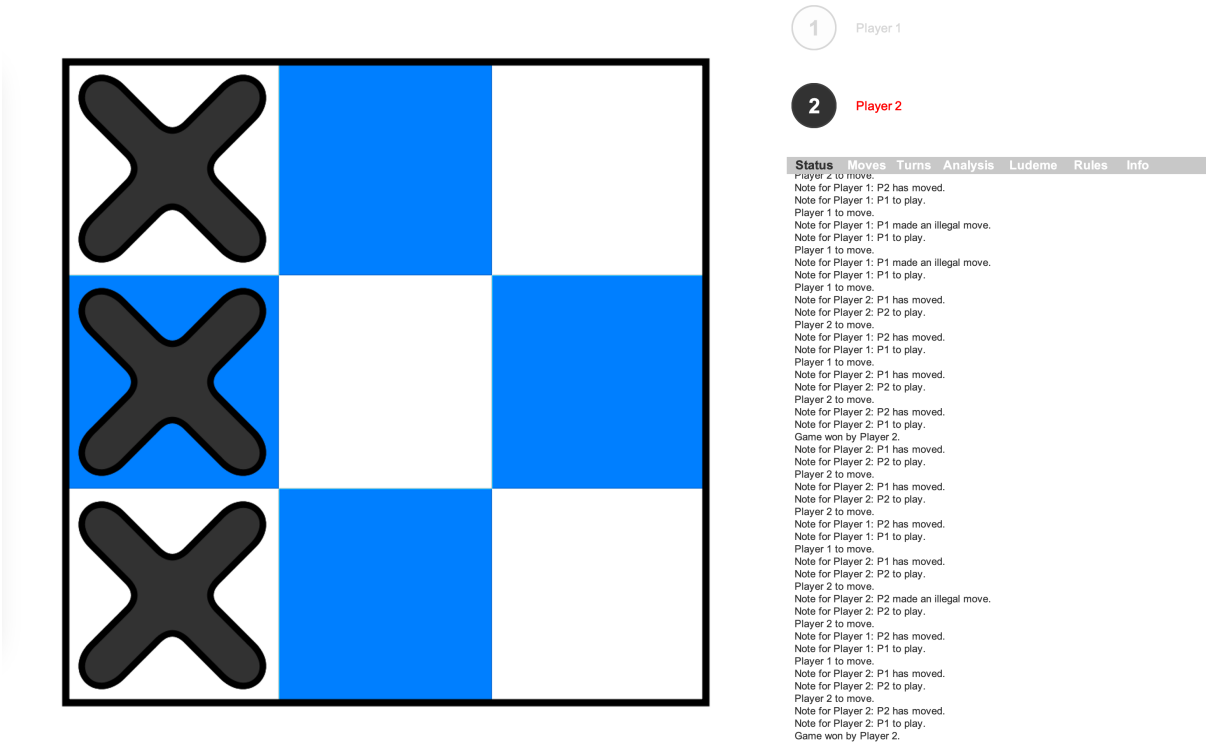

Figura 4.2: **Vittoria del giocatore 2 in Tic-Tac-Toe ad informazione incompleta.**

L'implementazione ad informazione completa è costituita solamente dai Ludemes necessari a rappresentare la tavola, i giocatori, le mosse legali e le condizioni di vittoria: questi meccanismi presentano un implementazione elementare in Ludii, per cui il codice risultante è estremamente compatto e leggibile.

D'altra parte, invece, l'implementazione ad informazione incompleta, oltre ad includere tutti i Ludemes prima descritti nella versione tradizionale, richiede le definizioni aggiuntive di mossa legale, di mossa illegale e delle loro conseguenze sulla tavola qualora si verificassero. Nel concreto vengono quindi definite delle funzioni ausiliarie aggiuntive, oltre che la distinzione tra i casi all'interno del corpo del codice del gioco. A queste differenze, inoltre, si aggiunge la presenza di metadati che regolano l'aspetto grafico del gioco, che nella versione tradizionale mancano.

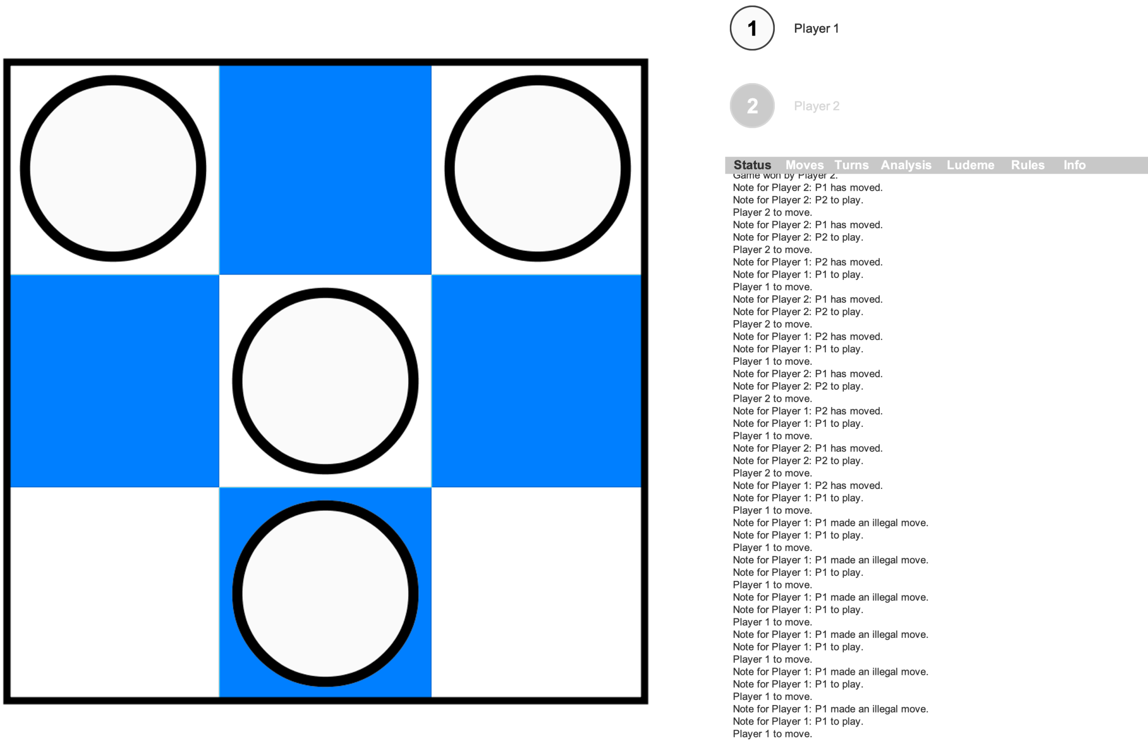

Figura 4.3: **Pareggio in Tic-Tac-Toe ad informazione incompleta.**

Capitolo 5

Conclusione

Nel corso di questo elaborato abbiamo visto i principali aspetti implementativi che hanno portato allo sviluppo di una nuova versione di Tic-Tac-Toe ad informazione incompleta per la piattaforma Ludii. Per lo sviluppo di questa versione del gioco la piattaforma Ludii si è rivelata perfetta, in quanto l'espressività del linguaggio ha permesso l'implementazione della parte ad informazione incompleta: questa caratteristica chiave, inoltre, si unisce agli altri due motivi che rendono Ludii uno dei GGP più moderni in circolazione, ovvero la grande efficienza e la leggibilità del codice, che ha permesso di rappresentare il gioco in un formato molto compatto e facilmente comprensibile.

Un altro fattore di interesse per la piattaforma Ludii - e per i GGP in generale - viene soprattutto dagli ampi usi che si possono fare di piattaforme di questo genere nell'ambito dello studio sui giochi tradizionali e le intelligenze artificiali. Questo tipo di sistemi, infatti, permette sia di testare sistemi di intelligenza artificiale in contesti sempre nuovi e stimolanti, sia di studiare i giochi da tavola tradizionali (con tutti gli aspetti culturali e matematici che ne conseguono) attraverso modalità nuove e potenti. Si tratta quindi di strumenti che permetteranno alla ricerca interdisciplinare

in questo ambito di avanzare notevolmente e di portare a scoperte nuove e sconosciute.

Bibliografia

[1] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47:235–256, 2002.

[2] Cameron Browne. *Automatic generation and evaluation of recombination games.* PhD thesis, Queensland University of Technology, 2008.

[3] Cameron Browne. A class grammar for general games. In *Proc. International Conference on Computers and Games*, pages 167–182. Springer, 2016.

[4] Cameron Browne. AI for Ancient Games. In *Kunstliche Intelligenz*, pages 89–93. Springer, 2020.

[5] Cameron Browne and Frederic Maire. Evolutionary game design. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(1):1–16, 2010.

[6] Cameron Browne, Éric Piette, Matthew Stephenson, and Dennis JNJ Soemers. Ludii general game system for modeling, analyzing, and designing board games. In N. Lee, editor, *Encyclopedia of Computer Graphics and Games*. Springer Nature, 2023.

[7] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[8] Guillaume Chaslot, Mark Winands, H Jaap van den Herik, Jos Uiterwijk, and Bruno Bouzy. Progressive strategies for Monte-Carlo tree search. *New Mathematics and Natural Computation*, 4(03):343–357, 2008.

[9] Elliot Doe, Mark HM Winands, Dennis JNJ Soemers, and Cameron Browne. Combining monte-carlo tree search with proof-number search. In *2022 IEEE Conference on Games (CoG)*, pages 206–212. IEEE, 2022.

[10] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–610, 2014.

[11] Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the AAAI competition. *AI Magazine*, 26(2):62–62, 2005.

[12] Simon Greenhill. Evolution and language: phylogenetic analyses. *International Encyclopedia of the Social & Behavioral Sciences*, pages 370–377, 2015.

[13] Donald E Knuth and Ronald W Moore. An analysis of alpha-beta pruning. *Artificial intelligence*, 6(4):293–326, 1975.

[14] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[15] David Parlett. What'sa ludeme. *Game & Puzzle Design*, 2(2):81, 2017.

[16] Eric Piette, Dennis JNJ Soemers, Matthew Stephenson, Chiara F Sironi, Mark HM Winands, and Cameron Browne. Ludii–the ludemic general game system. *arXiv preprint arXiv:1905.05013*, 2019.

[17] Éric Piette, Matthew Stephenson, Dennis J. N. J. Soemers, and Cameron Browne. General board game concepts. *CoRR*, abs/2107.01078, 2021.

[18] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[19] Dennis J. N. J. Soemers, Éric Piette, Matthew Stephenson, and Cameron Browne. The Ludii Game Description Language is Universal, 2022.

[20] M. Stephenson, É. Piette, D. J. N. J. Soemers, and C. Browne. Ludii as a competition platform. In *Proc. IEEE Conference on Games (CoG)*, pages 634–641, Aug 2019.

[21] Matthew Stephenson, Dennis JNJ Soemers, Éric Piette, and Cameron Browne. General game heuristic prediction based on ludeme descriptions. In *Proc. IEEE Conference on Games (CoG)*, pages 1–4, 2021.

[22] Maciej Świechowski, HyunSoo Park, Jacek Mańdziuk, Kyung-Joong Kim, et al. Recent advances in general game playing. *The Scientific World Journal*, 2015, 2015.

[23] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[24] Dan Ventura and Sean Warnick. A theoretical foundation for inductive transfer. *Brigham Young University, College of Physical and Mathematical Sciences*, 19, 2007.

[25] Éric Piette, Matthew Stephenson, Dennis J. N. J. Soemers, and Cameron Browne. An Empirical Evaluation of Two General Game Systems: Ludii and RBG. In *Proc. IEEE Conference on Games*, pages 626–629, 2019.

Appendice A

Codice

```
(define "IllegalMove" //Definisce una mossa Illegale
    (and {            //mostra i due avvisi di seguito
                      //nella sezione 'Status' di Ludii
        (note player:Mover "made an illegal move")
        (note player:Mover "to play")
        (moveAgain)   //Rinnova il turno al giocatore
    })
)

//Definisce il significato di una cella
//Non vuota
(define "NotEmpty" (not (is In (last To) (sites Empty))))

(define "MadeALegalMove"  //Definisce una mossa legale
    (do
        (add
            (to (last To))
            (then    //Aggiunge un pezzo alla tavola
                     //per poi metterlo 'Hidden' per non mostrarlo
                     //all'avversario
                (and {
                    (set Hidden at:(last To) to:Next)
```

```
                    (note player:Mover "has moved")
                    (note player:Next "to play")
                })
            )
        )
        ifAfterwards:("HasFreedom")
    )
)

//-------------------------------------------------------------------
(game "Incomplete Information TicTacToe"  //Nome del gioco
    (players 2)                           //Numero di giocatori
    (equipment
        {
            (board (square 3))  //Tavola di forma quadrata 3x3
            (piece "Disc" P1)   //Cerchio come icona del giocatore 1
            (piece "Cross" P2)  //Cerchio come icona del giocatore 2
        }
    )
    (rules
        (play
            (move           //ad ogni turno devi scegliere una
                Select      //tra le caselle che vedi vuote
                (from (union (sites Empty) (sites Hidden to:Mover)))
                (then
                    (priority
                        {   //viene verificato se
                            //la cella e' gia' occupata
                            //tramite la funzione NotEmpty
                            (
                                if ("NotEmpty")
                                ("IllegalMove") //Se e' occupata
                                                //si chiama IllegalMove
                            )
```

```
                        ("MadeALegalMove") //Altrimenti si chiama
                        //la funzione MadeALegalMove

                        ("IllegalMove")
                    }
                )
            )
        )
    )
    (end ("Line3Win"))  //Regola di conclusione del gioco:
                        //la partita si vince se si forma una
                        //linea di 3 elementi consecutivi
)
)

//--------------------------------------------------------------------

(metadata
    (graphics {         //Aspetti grafici del gioco: tavola
                        //a scacchi, colore intetno dei riquadri e
                        //dei relativi bordi

        (board Checkered)
        (board Colour Phase0 (colour White))
        (board Colour Phase1 (colour Blue))
        (board Colour OuterEdges (colour Black))
        (board Colour InnerEdges (colour Hidden))

    })
)
```