Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Studio della Piattaforma Ludii per il Gioco a Informazione Incompleta Kriegspiel: Traduzione dei Trial in PGN

Relatore:
Chiar.mo Prof.
Paolo Ciancarini

Presentata da:
Fabio Chiarini

II Sessione
Anno Accademico 2023/2024

Teoria dei Giochi

Informazione Imperfetta

PGN (Portable Game Notation)

Ludii

Kriegspiel

A chi mi ha sempre sostenuto:
siete la forza che ha reso tutto possibile.
Grazie di cuore.

Abstract

Questa tesi inizia con un'analisi approfondita di Ludii, una piattaforma ludica per giochi di strategia, utile per modellare giochi di strategia sia antichi che moderni. Successivamente, si affrontano le sfide specifiche poste dal Kriegspiel, una variante degli Scacchi che richiede la gestione dell'incertezza e la definizione di un ruolo di arbitro, dettagliando come queste siano state risolte nell'implementazione su Ludii.

La parte originale di questa tesi è la progettazione e lo sviluppo di un traduttore TRIAL-to-PGN, uno strumento progettato per ricostruire dettagliatamente nel formato Portable Game Notation (PGN: uno standard per le visualizzazioni scacchistiche online) tutti gli aspetti di una partita di Kriegspiel giocata su Ludii. La tesi descrive l'architettura del traduttore, le tecniche di parsing degli input, la gestione della scacchiera e la generazione delle mosse PGN.

I risultati dimostrano la flessibilità di Ludii nel modellare giochi complessi a informazione incompleta come il Kriegspiel e l'efficacia del traduttore nel rendere le partite accessibili per ulteriori analisi.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 0

INTRODUZIONE

Il mondo dei giochi da tavolo ha sempre rappresentato un terreno fertile per lo studio dell'Intelligenza Artificiale e delle strategie decisionali basate su euristiche. In questo contesto, il Kriegspiel, una variante degli scacchi caratterizzata da informazione incompleta, emerge come un dominio particolarmente sfidante e ricco di implicazioni. Questa tesi si propone di esplorare l'implementazione del Kriegspiel all'interno della piattaforma Ludii, un ambiente innovativo per la modellazione e l'analisi di giochi, e di sviluppare un traduttore per convertire le partite dal formato Trial di Ludii al più diffuso formato PGN (Portable Game Notation).

L'obiettivo principale di questo lavoro è duplice: da un lato, esaminare come le peculiarità del Kriegspiel possano essere efficacemente rappresentate e gestite all'interno di un sistema general-game come Ludii; dall'altro, creare uno strumento che permetta di rendere le partite di Kriegspiel giocate su Ludii accessibili a una più ampia comunità di ricercatori e appassionati, facilitando così ulteriori studi e analisi. Le metodologie e gli strumenti sviluppati potranno non solo facilitare la ricerca sul Kriegspiel, ma anche fornire spunti e approcci applicabili ad altri giochi a informazione imperfetta.

Nel corso di questa tesi, esamineremo in dettaglio la struttura e le funzionalità di Ludii, con particolare attenzione alle sue capacità di modellare giochi a informazione imperfetta. Analizzeremo le sfide specifiche poste dal Kriegspiel, come la gestione dell'incertezza e il ruolo dell'arbitro, e come queste siano state affrontate nell'implementazione su Ludii. Successivamente, ci concentreremo sullo sviluppo del traduttore TRIAL-to-PGN, esplorando le complessità tecniche e le scelte progettuali necessarie per catturare accuratamente tutti gli aspetti di una partita di Kriegspiel in formato PGN.

La struttura di questa tesi è organizzata come segue:

Nel Capitolo 1 forniremo una panoramica dettagliata della piattaforma Ludii, esplorando le sue origini, le caratteristiche principali e il suo approccio innovativo basato sui ludemi. Analizzeremo anche come Ludii si confronta con altre piattaforme generali di gioco.

Il Capitolo 2 si concentra sul Kriegspiel, descrivendo le sue regole, le caratteristiche uniche e le strategie realizzative. Esamineremo le sfide specifiche poste da questo gioco a informazione imperfetta e come sono state affrontate nella piattaforma Ludii insieme a diversi approcci alla creazione di giocatori artificiali per il Kriegspiel.

Nel Capitolo 3 presenteremo in dettaglio il traduttore TRIAL-to-PGN sviluppato, che è un risultato originale della tesi. Discuteremo l'architettura del software, le tecniche di parsing degli input, la gestione della scacchiera e il processo di generazione delle mosse PGN.

Infine, nell'ultimo capitolo tireremo le nostre conclusioni e accenneremo ad alcuni sviluppi futuri.

Capitolo 1

LUDII

Ludii è una piattaforma innovativa per sviluppare, analizzare e giocare ai giochi da tavolo. Nata dalla visione di creare un framework unificato per la ricerca sui giochi è ora uno strumento potente per accademici, designer di giochi e appassionati. [BPSS23].

1.1 Contesto

La ricerca sui giochi da tavolo vanta una ricca storia che spazia dall'intelligenza artificiale alla teoria dei giochi, dalla storia culturale all'antropologia. Negli ultimi decenni, diverse piattaforme hanno tentato di creare sistemi capaci di giocare a una varietà di giochi. Tra queste, Zillions of Games, lanciato nel 1998, ha offerto un ambiente per sfide contro un'IA o altri giocatori umani [Zil98]. Tuttavia, Zillions of Games presentava limiti in termini di analisi approfondita e il suo linguaggio di descrizione dei giochi era relativamente ristretto, non permettendo, ad esempio, l'implementazione di giochi ad informazione imperfetta.

Nei primi anni 2000 emerse il concetto di *General Game Playing*, che suggeriva di sviluppare sistemi di IA in grado di apprendere le regole di giochi sconosciuti e imparare a giocare in modo competente [GLP05]. Nel 2005, il Logic Group di Stanford diede vita all'International General Game Playing Competition (IGGPC), un evento annuale per testare e confrontare i sistemi GGP.

Il Game Description Language (GDL) costituisce il nucleo del framework GGP. Concepito come linguaggio formale per definire giochi discreti con informazioni complete, GDL utilizza la logica del primo ordine per descrivere le regole dei giochi in modo conciso e *machine-readable*, e divenne lo standard *de facto* nella ricerca accademica sul GGP.

Nonostante i suoi meriti, GDL presenta notevoli limitazioni. La sua complessità richiede una comprensione avanzata della logica del prim'ordine, il che lo rende difficile da usare per i non esperti. La verbosità del linguaggio comporta un processo di creazione dei giochi lungo e laborioso, poiché ogni aspetto strutturale deve essere definito da zero. La mancanza di modularità ostacola anche piccole modifiche ai giochi esistenti. Inoltre, l'elaborazione delle descrizioni GDL risulta computazionalmente onerosa, il che limita la complessità dei giochi modellabili e giocabili efficacemente.

Per affrontare alcune di queste limitazioni, è stata sviluppata un'estensione chiamata GDL-II (Game Description Language with Incomplete Information)[Thi10]. GDL-II introduce il ruolo speciale "*random*" per modellare elementi di casualità come il lancio di dadi o la mescola di carte, e il keyword "*sees*" per controllare l'informazione che ogni giocatore riceve durante il gioco.

Questi miglioramenti hanno permesso a GDL-II di rappresentare giochi con informazione imperfetta e elementi stocastici. Tuttavia, GDL-II mantiene molte delle complessità e inefficienze del GDL originale ed è in questo scenario che Ludii ha fatto la sua comparsa, con l'ambizione di colmare le lacune delle piattaforme esistenti e fornire un framework completo per la ricerca sui giochi da tavolo.

1.2 Origini e Sviluppo

Il progetto Ludii[PSS+20] è stato avviato nel 2017 presso l'Università di Maastricht, sotto la guida del professor Cameron Browne. Questo progetto, la cui prima versione pubblica fu rilasciata nel 2020, fa parte del Digital Ludeme Project, un progetto di ricerca finanziato dal Consiglio Europeo della Ricerca (ERC) tramite un Consolidator Grant. Il team multidisciplinare di Ludii, composto da esperti in informatica, matematica e scienze dei giochi, ha sviluppato la piattaforma con tre obiettivi principali: facilitare l'analisi storica e culturale dei giochi, permettere la ricostruzione di giochi antichi[CS23] e studiare l'evoluzione del design dei giochi nel tempo. Questo approccio innovativo si basa sul concetto di "ludemi" - gli elementi fondamentali dei giochi - che ha permesso di creare un linguaggio di descrizione dei giochi (L-GDL) espressivo e conciso, fornendo al contempo un ambiente completo per l'analisi computazionale e la ricerca accademica.

Al cuore di Ludii c'è la sua vasta e diversificata libreria di giochi che include centinaia di titoli provenienti da varie culture e periodi storici. La libreria spazia dai giochi da tavolo astratti come Scacchi e Go ai giochi di carte e di percorso. Per gestire questa grande varietà di giochi e varianti, il team ha sviluppato un motore generale capace di giocare, altamente efficiente e scalabile. Questo motore non solo interpreta le descrizioni dei giochi basate su ludemi e ne esegue il *gameplay*, ma offre anche

funzionalità avanzate per il playback e l'analisi delle partite. Complementare al motore, è presente un'interfaccia utente grafica (GUI) intuitiva e flessibile che si adatta dinamicamente alle specifiche di ciascun gioco.

1.3 Caratteristiche

1.3.1 Approccio Ludemico

L'originalità di Ludii risiede nel suo approccio basato sui ludemi. I ludemi rappresentano gli elementi concettuali di un gioco, come i pezzi, le loro regole di movimento, le condizioni di vittoria, o la struttura della scacchiera su cui si muovono i pezzi. Esprimono i concetti principali di un gioco e li definiscono in modo significativo, rendendo le descrizioni dei giochi più comprensibili non solo per i programmatori, ma anche per i designer e i ricercatori.

I giochi possono essere composti combinando ludemi predefiniti, il che semplifica notevolmente il processo di creazione, modifica e studio dei giochi. Essendo elementi modulari, i ludemi possono essere condivisi tra diversi giochi, facilitando l'analisi comparativa e lo studio dell'evoluzione dei design dei giochi. Questo aspetto è particolarmente utile per la ricerca, permettendo di identificare pattern comuni e di tracciare lo sviluppo storico di meccaniche di gioco attraverso diverse culture e periodi

1.3.2 Ludii Game Description Language (L-GDL)

Ludii utilizza un linguaggio di descrizione dei giochi derivato direttamente dalla gerarchia di classi del codice sorgente sottostante noto come "class grammar". Il quale garantisce una corrispondenza 1:1 tra il codice sorgente e la grammatica, assicurando che le descrizioni dei giochi siano sempre valide e eseguibili. Inoltre, l'estensibilità del sistema è notevolmente migliorata, poiché nuove funzionalità possono essere aggiunte semplicemente estendendo il codice sorgente, con la grammatica che si aggiorna automaticamente. Infine, questo approccio favorisce l'efficienza, permettendo l'istanziazione diretta delle descrizioni dei giochi nel codice della libreria corrispondente per la compilazione, eliminando così la necessità di un interprete separato.

Un recente studio di Soemers et al.[SPSB24] ha dimostrato che L-GDL è un linguaggio universale, capace di rappresentare qualsiasi gioco finito in forma estesa, inclusi giochi con informazione imperfetta e elementi stocastici. Questo risultato teorico equipara la potenza espressiva di L-GDL a quella del Game Description Langua-

ge (GDL) originale e della sua estensione GDL-II, pur mantenendo i vantaggi di leggibilità e facilità d'uso che caratterizzano Ludii.

Gli autori propongono una procedura dettagliata per costruire una descrizione L-GDL equivalente per qualsiasi gioco in forma estesa dato. I punti chiave di questa procedura includono:

- **Rappresentazione del grafo di gioco**: Utilizzo di una struttura a grafo in L-GDL per rappresentare l'intero albero di gioco.
- **Gestione dell'informazione imperfetta**: Impiego di "copie" multiple del grafo di gioco per tracciare gli insiemi di informazione di ciascun giocatore.
- **Implementazione di mosse e transizioni**: Utilizzo di regole L-GDL per definire le mosse legali e le transizioni di stato
- **Modellazione della casualità**: Impiego del ludema (random ...) per gestire nodi di caso e transizioni probabilistiche, in modo analogo a GDL-II.
- **Definizione di condizioni di terminazione e payoff**: Utilizzo di regole di fine gioco per specificare stati terminali e payoff associati.

1.3.3 Strumenti di analisi

Una delle caratteristiche principali di Ludii è il suo ricco set di strumenti per l'analisi computazionale dei giochi. Questi strumenti consentono ai ricercatori di studiare le proprietà dei giochi utilizzando metodi matematici e tecniche di intelligenza artificiale, offrendo un'ampia gamma di funzionalità per esplorare e comprendere la struttura e la dinamica dei giochi. Offre la capacità di generare l'intero albero di gioco per un determinato gioco, mappando tutti i possibili stati e le transizioni tra loro implementa una varietà di tecniche per esplorare lo spazio di stato del gioco e trovare strategie ottimali e sono inclusi algoritmi di ricerca ad albero come la ricerca minimax o ad albero Monte Carlo.

Per alcuni giochi, Ludii può utilizzare algoritmi di ricerca esaustiva o retrograde analysis per risolvere completamente il gioco, determinando il risultato ottimale con un gioco perfetto, fornendo preziose informazioni sulla complessità del gioco e l'esistenza di strategie dominanti. Include strumenti per valutare l'equità di un gioco, esaminando fattori come il vantaggio del primo giocatore e la distribuzione dei risultati tra i giocatori ed anche strumenti per l'analisi della complessità, fornendo metriche per quantificare la complessità di un gioco. Queste includono la dimensione dello spazio di stato, il fattore di ramificazione e la profondità dell'albero di gioco. Tali metriche forniscono informazioni sulle richieste cognitive del gioco e pos-

sono essere utilizzate per confrontare la complessità tra diversi giochi, offrendo una prospettiva quantitativa sulla struttura e la difficoltà dei giochi analizzati.

1.3.4 Agenti

Ludii si configura come una piattaforma innovativa per la ricerca e lo sviluppo di agenti intelligenti nel campo dell'IA applicata ai giochi. La sua architettura è concepita per offrire un ambiente versatile e ricco di funzionalità, capace di supportare l'implementazione e il testing di una vasta gamma di strategie di IA.

La piattaforma consente la creazione di agenti in grado di adattarsi a diverse meccaniche di gioco, dai tradizionali giochi a turni alternati fino a quelli con mosse simultanee. Possono essere progettati per prendere decisioni basate su molteplici fattori, tra cui lo stato corrente del gioco e i vincoli temporali. Ludii offre notevole flessibilità nello sviluppo di questi agenti, supportando l'implementazione in diversi linguaggi di programmazione come Java, C++ e Python.

1.3.5 Trial Partita

Il formato Trial (TRL) di Ludii è una rappresentazione testuale dettagliata dei "trial" del gioco, che sono essenzialmente registrazioni complete di partite. La struttura di un file TRL è organizzata in modo sequenziale, riflettendo l'evoluzione della partita mossa dopo mossa. Ecco una descrizione più approfondita:

1. **Intestazione**:
    - Inizia con "game=" seguito dal nome del gioco.
    - Può includere una sezione "START GAME OPTIONS" con le opzioni specifiche del gioco, seguita da "END GAME OPTIONS".
    - Contiene lo stato iniziale del generatore di numeri casuali, indicato da "RNG internal state=" seguito da una serie di numeri separati da virgole. Questo elemento è importante in alcuni giochi, ma non in tutti:
        - In giochi che incorporano elementi di casualità, come il lancio di dadi o la mischia di carte, l'RNG è fondamentale per garantire la riproducibilità delle partite. Lo stato iniziale dell'RNG permette di ricreare esattamente la stessa sequenza di eventi casuali in una riproduzione della partita.
        - Anche se il Kriegspiel non utilizza direttamente l'RNG per il gameplay, Ludii include questa informazione per compatibilità e per

supportare potenziali varianti o analisi che potrebbero richiedere elementi casuali. In questi casi questa informazione sull'RNG sarebbe cruciale per la fedele riproduzione della partita.

2. **Setup iniziale**: Il file inizia con una serie di mosse di setup, identificate da `Move=[Move:mover=0,...]`. Queste mosse rappresentano il posizionamento iniziale dei pezzi sulla scacchiera da parte del sistema Ludii. Ogni pezzo viene posizionato individualmente, permettendo una flessibilità nella configurazione iniziale.

3. **Sequenza di mosse**: Dopo il setup, ogni mossa successiva è rappresentata da una riga che inizia con "Move=" seguita da dettagli come:

   - `mover`: il giocatore che effettua la mossa (1 per il bianco, 2 per il nero)
   - `from` e `to`: posizioni di partenza e arrivo in formato numerico
   - `actions`: azioni associate alla mossa, come aggiungere o rimuovere pezzi

4. **Ricostruzione dello stato**: È importante notare che il file TRL non fornisce uno "snapshot" completo della scacchiera dopo ogni mossa. Invece, lo stato della scacchiera deve essere ricostruito incrementalmente, applicando ogni mossa in sequenza.

5. **Promozioni**: Le promozioni dei pedoni sono rappresentate come mosse separate, immediatamente successive alla mossa del pedone che raggiunge l'ultima traversa.

6. **Note e messaggi**: Possono essere incluse per fornire informazioni aggiuntive, come notifiche di scacco o catture en passant.

7. **Mosse illegali**: Il formato registra anche i tentativi di mosse illegali, fornendo insight sulle strategie e gli errori dei giocatori.

8. **Risultato e metadati**: Alla fine del file, vengono incluse diverse informazioni:

   - "numInitialPlacementMoves=" seguito dal numero di mosse di posizionamento iniziale. Questo indica quante mosse sono state fatte per impostare la configurazione iniziale del gioco.
   - "winner=" seguito dall'identificatore del vincitore. Specifica quale giocatore ha vinto la partita.
   - "endtype=" seguito dal tipo di conclusione della partita. Indica come è terminata la partita (ad esempio, scacco matto, stallo, o altre condizioni specifiche del gioco).

- "rankings=" seguito da una lista di valori separati da virgole, rappresentanti il ranking finale dei giocatori.
  - Il ranking migliore è 1.0, con valori più alti che indicano posizioni inferiori.
  - Per esempio, in una partita a due giocatori:
    * "rankings=1.0,2.0" indica che il Giocatore 1 ha vinto.
    * "rankings=2.0,1.0" indica che il Giocatore 2 ha vinto.
    * "rankings=1.5,1.5" rappresenta un pareggio.
  - In giochi con più giocatori, i valori riflettono l'ordine di arrivo.
- "SANDBOX=" seguito da un valore booleano.
  - Indica se la partita è stata giocata in modalità sandbox.
  - La modalità sandbox è uno strumento avanzato utile per debugging, test, esplorazione di scenari specifici e creazione di situazioni di gioco per analisi o insegnamento. Permette modifiche libere allo stato del gioco:
    * Spostare, aggiungere o rimuovere pezzi dalla scacchiera.
    * Modificare variabili di stato dei pezzi (conteggio, rotazione, ecc.).
    * Creare configurazioni di gioco specifiche per test o analisi.
  - Un valore "true" indica che sono state utilizzate funzionalità sandbox, potenzialmente alterando il corso normale del gioco. Quindi la partita registrata potrebbe non seguire le regole standard e non essere rappresentativa di una partita normale.
- "LUDII_VERSION=" seguito dal numero di versione di Ludii utilizzato. Questo è importante per la compatibilità e la riproducibilità, in quanto diverse versioni di Ludii potrebbero avere implementazioni leggermente diverse delle regole del gioco.

1.4 Confronto con Altri Sistemi

Un recente studio comparativo[PSSB19] ha messo a confronto Ludii con un altro sistema emergente di gioco generale, il Regular Boardgames (RBG) language. Questo confronto è particolarmente significativo in quanto sia Ludii che RBG rappresentano

alternative più efficienti allo stato dell'arte accademico precedente, il Game Description Language (GDL). Entrambi i sistemi sono stati dimostrati essere universali per la classe dei giochi deterministici finiti con informazione completa.

Tabella 1.1: Confronto sperimentale tra GDL, RBG e Ludii (Ratio = Ludii / RBG). [PSSB19]

Gioco	GDL	RBG	Ludii	Ratio	GDL	RBG Int	RBG Comp	Ludii	Ratio Int	Ratio Comp
Amazons	1158	195	51	0.26	185	307	625	4349	14.17	6.96
Arimaa	–	735	359	0.49	–	0.01	0.11	714	71400.00	6490.91
Breakthrough	670	134	65	0.49	1123	4962	16694	4741	0.96	0.28
Chess	4392	641	186	0.29	0.06	79	714	720	9.11	1.01
Chinese Checkers	–	418	243	0.58	–	232	1090	1105	4.76	1.01
Connect-4	751	155	31	0.20	13664	41897	84124	94077	2.25	1.12
English Checkers	1282	263	161	0.61	872	2963	14286	8135	2.75	0.57
Double Chess	–	790	202	0.26	–	6	50	81	13.50	1.62
Gomoku	514	324	21	0.06	927	1330	2212	42985	32.32	19.43
Hex	–	245	81	0.33	–	2741	5787	11077	4.04	1.91
Int. Checkers	–	498	244	0.49	–	262	1941	3444	13.15	1.77
Reversi	894	311	103	0.33	203	1468	2012	2081	1.42	1.03
The Mill Game	–	296	103	0.15	–	1063	7423	72734	68.42	9.80
Tic-Tac-Toe	381	101	25	0.25	85319	139312	400000	535294	3.84	1.34

Nota:

- **Colonne 1-4 (Numero di token, un Ratio inferiore a 1 indica una maggiore concisione di Ludii rispetto a RBG):**
  - GDL: Numero di token necessari per descrivere il gioco in Game Description Language
  - RBG: Numero di token necessari per descrivere il gioco in Regular Boardgames language
  - Ludii: Numero di token necessari per descrivere il gioco in Ludii
  - Ratio: Rapporto tra il numero di token di Ludii e RBG (Ludii/RBG)
- **Colonne 5-10 (Playouts al secondo, un Ratio superiore a 1 indica una maggiore efficienza di Ludii rispetto a RBG):**
  - GDL: Numero di playouts al secondo per GDL
  - RBG Int: Numero di playouts al secondo per l'interprete RBG
  - RBG Comp: Numero di playouts al secondo per il compilatore RBG
  - Ludii: Numero di playouts al secondo per Ludii
  - Ratio Int: Rapporto tra i playouts di Ludii e dell'interprete RBG (Ludii/RBG Int)
  - Ratio Comp: Rapporto tra i playouts di Ludii e del compilatore RBG (Ludii/RBG Comp)

1.4.1 Approcci alla Descrizione dei Giochi

Ludii utilizza un approccio basato su ludemi, con classi Java che definiscono concetti di gioco intuitivi. RBG, d'altra parte, si basa su espressioni regolari per descrivere i giochi offrendo sia una versione di linguaggio di alto livello per la leggibilità umana, sia una versione di basso livello ottimizzata per l'efficienza computazionale.

In media, Ludii necessita di solo un terzo dei token utilizzati da RBG per modellare un gioco. Questo rende le descrizioni dei giochi in Ludii più concise e potenzialmente più facili da creare e modificare. L'approccio ludemico di Ludii, con i suoi nomi di classi intuitivi (come Board, Hand, Deck, Slide, Step), rende le descrizioni dei giochi più comprensibili anche per utenti non esperti. Al contrario, RBG richiede l'uso di macro specifiche per il gioco per descrivere concetti di alto livello, il che può rendere le descrizioni meno immediatamente comprensibili. Sebbene la chiarezza di una descrizione di gioco sia soggettiva, gli autori dello studio argomentano che la maggior parte degli utenti, specialmente quelli non esperti, troverebbero le descrizioni di Ludii più chiare in generale.

1.4.2 Efficienza Computazionale

Nell'articolo[PSSB19] la valutazione dell'efficienza computazionale di Ludii si è basata su un confronto diretto con RBG, utilizzando come metrica il numero di playouts Monte Carlo casuali per secondo, un parametro particolarmente rilevante per molti algoritmi di IA applicati ai giochi. I test sono stati condotti su un singolo core di CPU, con una durata fissa di 10 minuti per ciascun test, fornendo così un quadro comparativo dettagliato delle prestazioni dei due sistemi.

I risultati hanno evidenziato una netta superiorità di Ludii in termini di prestazioni computazionali per la maggior parte dei giochi testati. In particolare, per giochi di elevata complessità come Amazons, Scacchi o Dama Internazionale, Ludii ha dimostrato di essere almeno dieci volte più veloce dell'interprete RBG. Un caso emblematico è rappresentato da Arimaa, un gioco notoriamente complesso, dove Ludii è riuscito a eseguire 700 playouts al secondo, mentre l'interprete RBG non è stato in grado di completarne nemmeno uno. Questo vantaggio prestazionale si è mantenuto anche nel confronto con la versione compilata di RBG, generalmente più efficiente dell'interprete. Tuttavia, è importante notare che RBG ha superato Ludii in termini di efficienza per due giochi specifici: Breakthrough e English Checkers. Questi risultati suggeriscono che per giochi con regole relativamente semplici e movimenti regolari, l'approccio di RBG potrebbe offrire alcuni vantaggi in termini di efficienza.

Un aspetto particolarmente interessante di questo confronto riguarda le differenze di implementazione tra i due sistemi. Mentre RBG è implementato in C++, un linguaggio noto per le sue elevate prestazioni, Ludii è sviluppato in Java. Nonostante il potenziale vantaggio prestazionale del C++, Ludii è riuscito a superare RBG nella maggior parte dei casi testati sottolineando l'efficacia dell'architettura basata su ludemi per la ricerca su giochi complessi.

Capitolo 2

KRIEGSPIEL

Gli Scacchi discendono da una lunga tradizione di giochi di simulazione militare [Mur13]. Molti ritengono tuttavia che la natura del gioco ad informazione completa sia inadeguata a modellare realisticamente una battaglia o una guerra. È per questo che è stata inventata una variante degli Scacchi chiamata Kriegspiel. Il Kriegspiel appartiene alla categoria dei wargame, giochi ad *informazione imperfetta* (o *incompleta*) [KP95], che rappresentano un modello per lo studio e lo sviluppo di agenti artificiali operanti in situazioni decisionali complesse del mondo reale con condizioni di incertezza [Ass21], una capacità fondamentale per molte applicazioni pratiche dell'intelligenza artificiale.

Nel Kriegspiel si applicano tutte le regole degli Scacchi, ma i giocatori sanno solo dove l'avversario si trova all'inizio della partita, ma non durante il gioco. Un giocatore Kriegspiel può solo supporre le mosse dell'avversario, in base alle limitate informazioni a disposizione. Poiché nessuno dei due giocatori può verificare autonomamente la legalità delle mosse, è necessario introdurre un terzo partecipante, il direttore (o arbitro), che ha piena visibilità della scacchiera e funge da arbitro controllando le comunicazioni tra i giocatori [WBB72].

I wargame sono in genere molto difficili da giocare algoritmicamente in modo efficace, richiedendo una combinazione di compiti complessi come ricerca euristica, ricostruzione probabilistica dell'informazione di stato del gioco, e modellazione dell'avversario [YT18]. Molti giochi da tavolo ricadono in questa categoria, derivando da giochi ad informazione perfetta a cui è stato aggiunto un elemento di "nebbia di guerra" che nasconde alcune informazioni ai giocatori, come appunto nel caso di Kriegspiel [CF07] e Phantom-Go [Caz06] (la variante a informazione parziale del Go).

Nonostante sia un gioco che risale alla fine del XIX secolo, i primi tentativi di costruire programmi efficienti per giocare a Kriegspiel risalgono a tempi recenti, basandosi inizialmente sul metodo Monte Carlo e poi su varianti dell'algoritmo minimax applicate a strutture dati dedicate chiamate "metaposizioni", in grado di catturare l'incertezza sulla posizione dei pezzi. Con il tempo, la ricerca ad albero Monte Carlo (MCTS) si è affermata come

una tecnica molto promettente per i giochi in cui il tradizionale minimax fatica a causa dell'enormità dello spazio degli stati e della difficoltà nel definire buone funzioni di valutazione. Essendo Kriegspiel caratterizzato proprio da questi elementi, diversi studi hanno iniziato a esplorare l'applicazione di MCTS a questo dominio [CF10a].

2.1 Regole e caratteristiche

Il Kriegspiel si basa sulle stesse regole degli Scacchi per quanto riguarda la disposizione iniziale dei pezzi, il loro movimento e l'obiettivo di dare scacco matto al Re avversario. Tuttavia, presenta alcune differenze fondamentali che lo rendono un gioco a informazione imperfetta, in contrapposizione alla perfetta informazione disponibile nelle partite di Scacchi standard.

La differenza principale sta nel fatto che il Kriegspiel viene giocato su tre scacchiere: una per ciascun giocatore, visibile solo a lui, e una per l'arbitro che ha una visione completa del gioco. I giocatori non vedono i pezzi dell'avversario e non vengono informati direttamente delle sue mosse, ma devono cercare di dedurle sulla base degli annunci dell'arbitro. Quest'ultimo comunica ai giocatori informazioni riguardanti le catture (specificando se è stato catturato un pezzo o un pedone e la coordinate di cattura), gli scacchi (direzione e tipo di scacco), l'eventuale illegalità di una mossa tentata e gli annunci dei "tentativi di cattura" dei pedoni ossia il numero di possibili di catture che possono eseguire i propri pedoni.

Nella pratica del gioco dal vivo, questa configurazione assume una forma fisica distintiva. I giocatori sono tipicamente seduti uno di fronte all'altro, ciascuno con la propria scacchiera nascosta alla vista dell'avversario da uno schermo o una barriera. L'arbitro si posiziona tra i due giocatori, con una terza scacchiera che riflette lo stato reale del gioco. Come si può vedere nella Figura 2.1, la disposizione per una partita di Kriegspiel differisce significativamente da quella degli scacchi tradizionali e crea un'atmosfera unica nei club di Kriegspiel storici .

Il ruolo cruciale dell'arbitro nel Kriegspiel è stato oggetto di tentativi di automazione fin dagli anni '60. Un esempio pionieristico è rappresentato dal programma UMPIRE [Bur67], sviluppato da John F. Burger presso la System Development Corporation nel 1967. UMPIRE, scritto in LISP 1.5 per il Sistema Time-Sharing Q-32, implementava un sistema di annunci simile a quello descritto, comunicando ai giocatori informazioni su catture, scacchi e tentativi di cattura dei pedoni dimostrando la fattibilità di automatizzare questo ruolo complesso.

Questo introduce nel Kriegspiel un forte elemento di deduzione e intuizione. I giocatori devono cercare attivamente di acquisire informazioni sulla posizione avversaria, sia tentando mosse esplorative (che possono essere accettate o rifiutate in base alla legalità delle stesse) sia interpretando gli annunci ricevuti per restringere le possibilità. Le stesse mosse "impossibili" assumono quindi un importante valore informativo.

Figura 2.1: Partita di Kriegspiel, Gambit Chess Rooms, Londra, 1946

Inoltre, l'elevato numero di possibili stati della scacchiera rende impossibile una strategia che miri a eliminare tutte le componenti avversarie: a differenza del Phantom Go, nel Kriegspiel non esiste una sequenza di mosse che permetta di svelare completamente la posizione dell'avversario, tranne forse nelle fasi finali. La riduzione dell'incertezza diventa più difficile man mano che vengono catturati i pezzi, rendendo più preziosa ogni nuova informazione.

2.1.1 Implementazione in GDL

Nikola Novarlic nella sua tesi di master [Nov23] ha esplorato l'integrazione di Kriegspiel all'interno di Ludii. Il suo primo contributo è stato la formalizzazione delle regole utilizzando il Game Description Language (GDL) di Ludii, introdotto nella Sezione 1.3.2. La descrizione GDL risultante, pur essendo significativamente più complessa di quella degli Scacchi standard, cattura in modo conciso ed espressivo le meccaniche chiave di Kriegspiel come i concetti di conoscenza parziale e comunicazione dell'arbitro.

La complessità dell'implementazione del Kriegspiel in Ludii è evidenziata dal fatto che ha richiesto 958 righe di codice, molte di più delle 208 righe degli Scacchi tradizionali. Questa differenza sottolinea la complessità aggiuntiva necessaria per gestire l'informazione nascosta e il ruolo dell'arbitro.

Novarlic ha affrontato la sfida di implementare le nuove regole utilizzando i ludemi, riusando la definizione della scacchiera e dei pezzi standard degli Scacchi. Per gestire l'informazione nascosta ha sfruttato la classe `ItemStateContainerHidden` di Ludii, che utilizza un `ChunkSet` per tracciare la visibilità dei pezzi per ciascun giocatore in ogni stato del

gioco. Un elemento centrale dell'implementazione è stato il ruolo dell'arbitro, realizzato attraverso un sistema di messaggi che si attiva dopo ogni mossa, informando i giocatori sulla legalità delle loro azioni e su eventi come catture o scacchi al re. 2.2

Novarlic ha sviluppato diverse funzioni specifiche per il Kriegspiel, essenziali per replicare accuratamente le regole e le dinamiche del gioco: LegalMove verifica la legalità delle mosse considerando l'informazione nascosta, PerformLegalMove esegue le mosse legali e notifica le catture ai giocatori, CountTries conta le possibili catture di pedoni, CheckType determina e comunica il tipo di scacco (diagonale lungo, diagonale corto, fila, colonna o cavallo), e KingInCheck verifica se il re è sotto scacco.

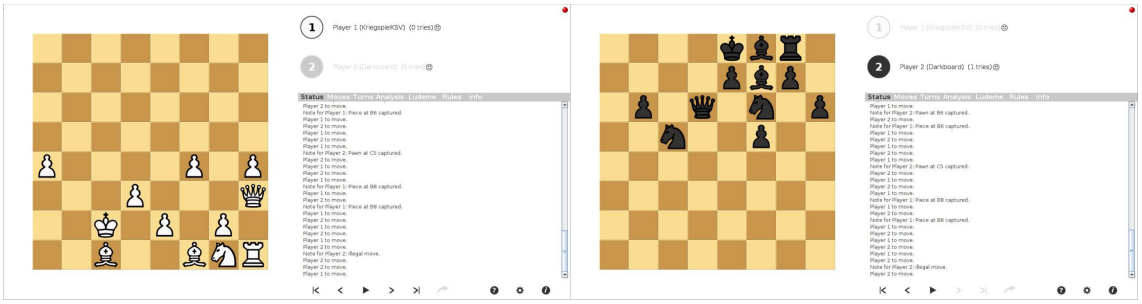
Oltre a introdurre le mosse pseudo-legali, Novarlic ha apportato alcune modifiche proprio al sistema Ludii per supportare meglio le caratteristiche di Kriegspiel. In particolare, ha implementato un protocollo di comunicazione tra gli agenti e il sistema attraverso una nuova funzione chiamata `getNotes`, che permette agli agenti di accedere ai messaggi dell'arbitro essenziali per ricostruire lo stato corrente del gioco. Questa funzione itera attraverso le azioni registrate dopo l'ultima mossa del giocatore e filtra quelle con il tag `note`, che corrisponde al nome della funzione di messaggio nel linguaggio del ludeme. I messaggi estratti vengono poi aggiunti a una lista risultante, che viene restituita all'agente.

2.1.1.1 Potenziali falle

Quando si implementano giochi a informazione imperfetta, gli agenti dovrebbero astenersi dall'utilizzare funzioni che forniscono informazioni complete sullo stato del gioco, in modo da rispettare le regole e lo spirito del Kriegspiel.

Novarlic e Ciancarini in [CN24] hanno discusso le potenziali falle che potrebbero permettere ai giocatori di barare in Kriegspiel su Ludii. Una di queste falle riguarda l'accesso alle informazioni sullo stato del gioco attraverso l'API di Ludii.

In particolare, alcune funzioni dell'API, come `getFullState()`, permettono di accedere allo stato completo del gioco, incluse le informazioni normalmente nascoste ai giocatori



(a) Ciò che vede il Bianco **(b)** Ciò che vede il Nero

Figura 2.2: Visualizzazioni della scacchiera nel Kriegspiel di Ludii. [CN24]

nel Kriegspiel. Vanno quindi considerate attentamente le implicazioni delle API e delle funzionalità offerte da una piattaforma come Ludii, se un agente utilizzasse queste funzioni, potrebbe ottenere un vantaggio sleale avendo accesso a informazioni che non dovrebbe possedere.

Per prevenire questo tipo di barare, Novarlic ha sottolineato l'importanza di progettare gli agenti in modo che si affidino esclusivamente alle informazioni fornite attraverso i messaggi dell'arbitro, accessibili tramite la funzione `getNotes`, per garantire fairness e integrità del gioco.

2.1.2 PGN Partita Esteso

Il formato PGN (Portable Game Notation) esteso per Kriegspiel è una variante specializzata del PGN standard, progettata per catturare le particolarità e le informazioni aggiuntive specifiche di questa variante degli scacchi. Questo formato mantiene la struttura di base del PGN standard, ma incorpora elementi cruciali per rappresentare accuratamente una partita di Kriegspiel. Ecco una descrizione dettagliata delle sue caratteristiche:

1. **Tag [Variant]**:
   - Impostato su "Kriegspiel (chess)" per identificare chiaramente la variante di gioco, essenziale per i software di scacchi per interpretare correttamente il contenuto del file.
2. **Commenti estesi**:
   - Dopo ogni mossa, vengono aggiunti commenti in formato {(info_umpire:mosse_illegali)} fornendo informazioni cruciali sulla dinamica della partita non visibili in una notazione standard.
3. **Componenti di info_umpire**:
   - `Xsquare`: Indica una cattura sulla casella specificata. Ad esempio, "Xe4" significa che è avvenuta una cattura sulla casella e4.
   - `Ctype`: Indica il tipo di scacco, dove:
     - F = scacco di fila (rank check)
     - R = scacco di traversa (file check)
     - L = scacco di diagonale lunga (long diagonal check)
     - S = scacco di diagonale corta (short diagonal check)
     - N = scacco di cavallo (knight check)
   - `Pn`: Indica, solo in assenza di scacco, il numero n di tentativi di cattura con pedone disponibili per il giocatore successivo.
4. **mosse_illegali**:

- Lista delle mosse illegali tentate dal giocatore prima di effettuare una mossa valida in notazione algebrica standard.

5. **Struttura delle mosse**:
   - Ogni mossa è numerata e include la mossa del bianco seguita da quella del nero.
   - I commenti estesi seguono immediatamente la mossa a cui si riferiscono.
6. **Informazioni aggiuntive**:
   - Il formato può includere anche altre informazioni come il risultato della partita e dettagli sulle promozioni.
7. **Compatibilità**:
   - Questo formato esteso rimane leggibile dai software PGN standard, che ignoreranno semplicemente i commenti aggiuntivi.
   - Software specializzati per Kriegspiel possono utilizzare queste informazioni extra per una riproduzione accurata della partita, includendo informazioni su tentativi falliti e sull'evoluzione dell'incertezza nel gioco.

2.2 Strategia e tattiche

L'approccio strategico nel Kriegspiel si discosta significativamente da quello degli Scacchi, l'esperienza negli scacchi tradizionali potrebbe non tradursi direttamente in un vantaggio a causa dell'informazione incompleta sulla posizione e le mosse dell'avversario [AS23]. Mentre negli Scacchi i giocatori possono pianificare sequenze di mosse basate sulla conoscenza della posizione, nel Kriegspiel la strategia deve essere molto più flessibile e adattiva, modificando continuamente piani e assunzioni alla luce dei nuovi frammenti di informazione rivelati dagli annunci dell'arbitro.

Questo non significa che i principi strategici fondamentali degli Scacchi, - come il controllo del centro, lo sviluppo dei pezzi, la sicurezza del Re e la creazione di minacce - perdano completamente di validità nel Kriegspiel. Tuttavia, il modo in cui questi principi possono essere messi in pratica è diverso. Ad esempio, il concetto di "sviluppo" deve tenere conto non solo della mobilità e dell'attività dei propri pezzi, ma anche della quantità di informazioni che le loro mosse potrebbero rivelare all'avversario.

Dal punto di vista tattico, il Kriegspiel premia la capacità di estrarre valore da ogni singola mossa, sia in termini di avanzamento del piano sia di acquisizione di informazioni. Tattiche tipiche includono:

- Mosse esplorative: mosse che mirano principalmente a ottenere informazioni sulla posizione dell'avversario, anche a costo di perdere un po' di tempo o posizione.

- Uso attivo dei pezzi minacciati: cercare di trarre vantaggio dai pezzi che si sospetta siano sotto attacco, ad esempio tentando una cattura o un sacrificio prima che vengano catturati.
- Creazione di minacce multiple: cercare di creare situazioni in cui l'avversario deve difendere più minacce contemporaneamente, costringendolo a rivelare informazioni.
- Protezione preventiva: posizionare i propri pezzi in modo da prevenire possibili minacce, anche se non ancora pienamente confermate.

Per quanto riguarda le fasi di gioco, il Kriegspiel presenta alcune differenze rispetto agli Scacchi. Nelle aperture, i giocatori tendono a seguire schemi familiari per limitare le possibilità dell'avversario e indirizzare il gioco verso situazioni note. Tuttavia, devono essere pronti a deviare dal piano non appena le informazioni ricevute lo richiedono.

Il mediogioco è probabilmente la fase in cui il Kriegspiel si discosta di più dagli Scacchi, a causa della grande incertezza sulla posizione. I giocatori devono costantemente rivedere il proprio piano e le relative assunzioni, cercando di individuare le debolezze dell'avversario e al contempo di proteggere le proprie.

Nei finali, le differenze tendono ad assottigliarsi man mano che i pezzi vengono catturati e l'incertezza si riduce; alcuni finali classici hanno addirittura soluzione algoritmica [CF10b]. Tuttavia, il Kriegspiel rimane più imprevedibile rispetto agli Scacchi: una singola informazione, come la spinta di un pedone che si rivela libero di avanzare, può ribaltare completamente la valutazione di una posizione.

Come sottolineato da Ciancarini [CDM97], il Kriegspiel riflette fedelmente le sfide decisionali in contesti economici e strategici reali, dove l'adattabilità e la gestione dell'incertezza sono cruciali. Questa dinamica è perfettamente illustrata nel finale Re e pedone contro Re, dove la "strategia Blotto" bilancia l'avanzamento verso un obiettivo (la promozione del pedone) con l'acquisizione di informazioni vitali:

1. Posizionare il Re dietro al pedone sulla sesta traversa.
2. Alternare probabilisticamente tra:
   (a) Spingere il pedone
   (b) Muovere il Re lateralmente per sondare la posizione avversaria

Ogni mossa diventa un compromesso tra progresso e esplorazione, richiedendo un approccio adattivo che supera la pianificazione a lungo termine tipica degli scacchi tradizionali. Il difensore, cercando di minimizzare le informazioni rivelate, trasforma questo finale apparentemente semplice in un'intensa battaglia di deduzione e bluff, dove la gestione dell'incertezza diventa tanto importante quanto la posizione dei pezzi stessi. Questa dinamica rispecchia perfettamente le sfide decisionali in ambienti economici e strategici caratterizzati da informazione incompleta e rapidi cambiamenti.

2.3 Avversari Artificiali

La crezione di un agente Kriegspiel presenta difficoltà, comuni a molti giochi ad informazione imperfetta [ST19], dovute alla natura dinamica dell'informazione parzialmente disponibile. Gli algoritmi classici di ricerca ad albero come il minimax, che hanno ottenuto grandi successi negli Scacchi grazie a funziooni di valutazione sempre più raffinate, si scontrano nel Kriegspiel con la complessità di gestire alberi di gioco con informazioni incomplete e stati di credenza (*belief states*) in continua evoluzione.

In questo contesto, possono essere utilizzati framework come OpenSpiel [LLL+19], un ambiente open-source sviluppato da DeepMind per la ricerca nel campo dell'apprendimento per rinforzo e della teoria dei giochi. OpenSpiel può essere definito come una collezione completa di ambienti e algoritmi per la ricerca nel campo della teoria dei giochi e dell'apprendimento per rinforzo, offrendo implementazioni di una vasta gamma di giochi, incluso il Kriegspiel. Per quest' ultimo offre un'implementazione del protocollo Kriegspiel, una base solida per lo sviluppo di agenti che include la gestione dello stato del gioco, le regole e la logica dell'arbitro. Inoltre, mette a disposizione una varietà di algoritmi preimplementati, come Monte Carlo Tree Search (MCTS) e varianti di apprendimento per rinforzo, che possono essere adattati al Kriegspiel senza doverli implementare da zero. OpenSpiel offre anche strutture dati efficienti per rappresentare la scacchiera e lo stato del gioco, facilitando l'accesso e la manipolazione delle informazioni necessarie per gli agenti.

2.3.1 Darkboard

Uno dei primi tentativi di affrontare queste sfide è rappresentato dal programma *Darkboard*.

Questo programma, sviluppato in Java e descritto in [Fav10], è stato recentemente adattato per funzionare come avversario all'interno di una moderna piattaforma web nell'ambito di un progetto che ho sviluppato per il corso di Ingegneria del Software 2023-2024. Il lavoro è stato realizzato in team con altri studenti adottando un approccio Agile, in particolare la metodologia Scrum, usando User Stories come unità primaria per la definizione e la prioritizzazione dei requisiti nel Product Backlog , strutturando il lavoro in sprint di due settimane e includendo sessioni regolari di Sprint Planning, Daily Stand-ups, Sprint Reviews e Sprint Retrospectives. L'integrazione ha comportato lo sviluppo di un middleware per la comunicazione tra il frontend Next.js/React e il backend Java di Darkboard, permettendo ai giocatori di sfidare direttamente questo giocatore artificiale. Il sistema utilizza Socket.IO per la comunicazione in tempo reale, consentendo una rapida trasmissione delle mosse del giocatore al backend Darkboard e l'aggiornamento della posizione in base alle sue risposte. L'arbitraggio è gestito da una funzione specifica di Darkboard (*StepwiseLocalUmpire*). Questa integrazione serve anche come banco di prova per lo sviluppo e il test di bot personalizzati attraverso un'API dedicata.

2.3.1.1 Metaposizioni

Darkboard introduce il concetto di "metaposizione" [BC06], una struttura dati compatta per rappresentare insiemi di possibili configurazioni della scacchiera compatibili con le informazioni disponibili. Ogni metaposizione è un array di 64 elementi, uno per ogni casa, dove ogni elemento è un intero a 8 bit che codifica la presenza o l'assenza dei vari pezzi attraverso la sua rappresentazione binaria. Questo permette di modellare l'incertezza sulla posizione dei pezzi avversari in modo efficiente, evitando l'esplosione combinatoria del numero di stati espliciti.

2.3.1.2 Algoritmo di ricerca

L'algoritmo di ricerca di Darkboard è un adattamento dell'algoritmo minimax che opera sulle metaposizioni anziché sugli stati di gioco completi. Data una metaposizione di partenza, Darkboard genera tutte le mosse "pseudo-legali" (cioè legali rispetto alle informazioni parziali disponibili) e le utilizza per espandere un albero di ricerca delle metaposizioni risultanti. La profondità di ricerca è limitata a pochi turni a causa della complessità del dominio. Per valutare le metaposizioni foglia, Darkboard estrae un campione casuale di stati di gioco completi coerenti con le informazioni contenute nella metaposizione, e li valuta utilizzando una funzione euristica. Questa funzione euristica è una combinazione pesata di diversi componenti che stimano vantaggi materiali, posizionali e informativi, cercando di catturare i principi strategici del Kriegspiel nonostante l'incertezza..

2.3.1.3 Approcci basati sulla simulazione Monte Carlo

Ricerche più recenti si sono orientate verso algoritmi basati su simulazioni Monte Carlo (MC) e ricerca ad albero, come il Monte Carlo Tree Search (MCTS), che hanno dimostrato grande efficacia in altri giochi complessi come il Go. L'idea di base del MCTS è di stimare il valore di ogni mossa generando in modo casuale o semi-casuale un gran numero di possibili continuazioni (simulazioni) della partita a partire dalla posizione risultante, e valutando ogni simulazione solo al raggiungimento di uno stato terminale. I risultati delle simulazioni vengono poi propagati all'indietro nell'albero di ricerca per aggiornare le stime dei nodi interni e guidare iterativamente l'esplorazione verso le mosse più promettenti.

L'applicazione del MCTS al Kriegspiel richiede però diversi adattamenti per tenere conto dell'informazione imperfetta. In particolare, la generazione di simulazioni adeguate e informative è complicata dall'incertezza sulla posizione dei pezzi avversari: simulazioni basate su configurazioni casuali rischiano di essere poco rappresentative e di produrre valutazioni inaffidabili. Inoltre, la costruzione dell'albero di ricerca deve modellare non solo le mosse dei giocatori, ma anche i possibili annunci dell'arbitro che potrebbero rivelare nuove informazioni.

2.3.1.3.1 Varianti

In [CF10a], gli autori esplorano diverse varianti del MCTS per il Kriegspiel, identificando due approcci principali. Il primo, chiamato "approccio A", genera all'inizio di ogni simulazione una configurazione casuale completa della scacchiera, coerente con le informazioni disponibili, e la utilizza per simulare in modo dettagliato una continuazione della partita fino ad uno stato terminale. Il secondo, chiamato "approccio B", rinuncia invece a modellare esplicitamente gli stati di gioco completi e simula direttamente le interazioni ad alto livello tra i giocatori e l'arbitro. In particolare, data una mossa del giocatore, l'algoritmo stima probabilisticamente il messaggio dell'arbitro risultante sulla base di un modello astratto che mantiene stime aggiornate della probabilità di presenza dei vari pezzi avversari in ogni casa. Queste stime vengono aggiornate dopo ogni mossa o annuncio simulato utilizzando sia le regole del gioco sia euristiche basate su dati raccolti da partite reali.

2.3.1.3.2 Risultati sperimentali

I risultati sperimentali riportati mostrano che l'approccio B, pur basandosi su un modello semplificato e astratto del gioco, ottiene prestazioni nettamente superiori sia all'approccio A sia al programma Darkboard basato su minimax e metaposizioni. L'approccio di quest'ultimo rimane limitato dalla difficoltà intrinseca di valutare staticamente una posizione incerta e di esplorare in profondità un albero di gioco così complesso e ramificato. Una variante ancora più semplificata, chiamata "approccio C", in cui le simulazioni vengono ridotte ad un singolo turno più eventuali mosse forzate (quiescence), raggiunge un punteggio Elo di circa 1900 su Internet Chess Club, superiore a quello di Darkboard e comparabile a quello di forti giocatori umani. Questo suggerisce che, nel Kriegspiel, una modellazione accurata dell'incertezza posizionale e dell'acquisizione incrementale di informazioni nel breve termine possa essere più importante della capacità di elaborare piani a lungo termine.

2.3.1.4 Conversione in agente Ludii

Dopo aver formalizzato con un ludeme le regole del Kriegspiel, Novarlic ha portato su Ludii l'agente Darkboard.

Per integrare Darkboard, ha creato una classe wrapper `Agent` che funge da interfaccia tra Ludii e la logica interna di Darkboard. 2.3

Questa classe estende la classe astratta `AI` di Ludii e implementa metodi come `initAI()` e `selectAction()` per gestire l'inizializzazione dell'agente, la selezione delle mosse, il trattamento delle mosse illegali, l'elaborazione dei messaggi dell'arbitro e la conversione delle mosse tra i formati Ludii e Darkboard.

Si è reso necessario adattare il sistema di messaggi dell'arbitro per renderlo comprensibile a Darkboard, implementando un meccanismo per interpretare i messaggi di Ludii e convertirli in un formato elaborabile da Darkboard. In particolare, ha implementato

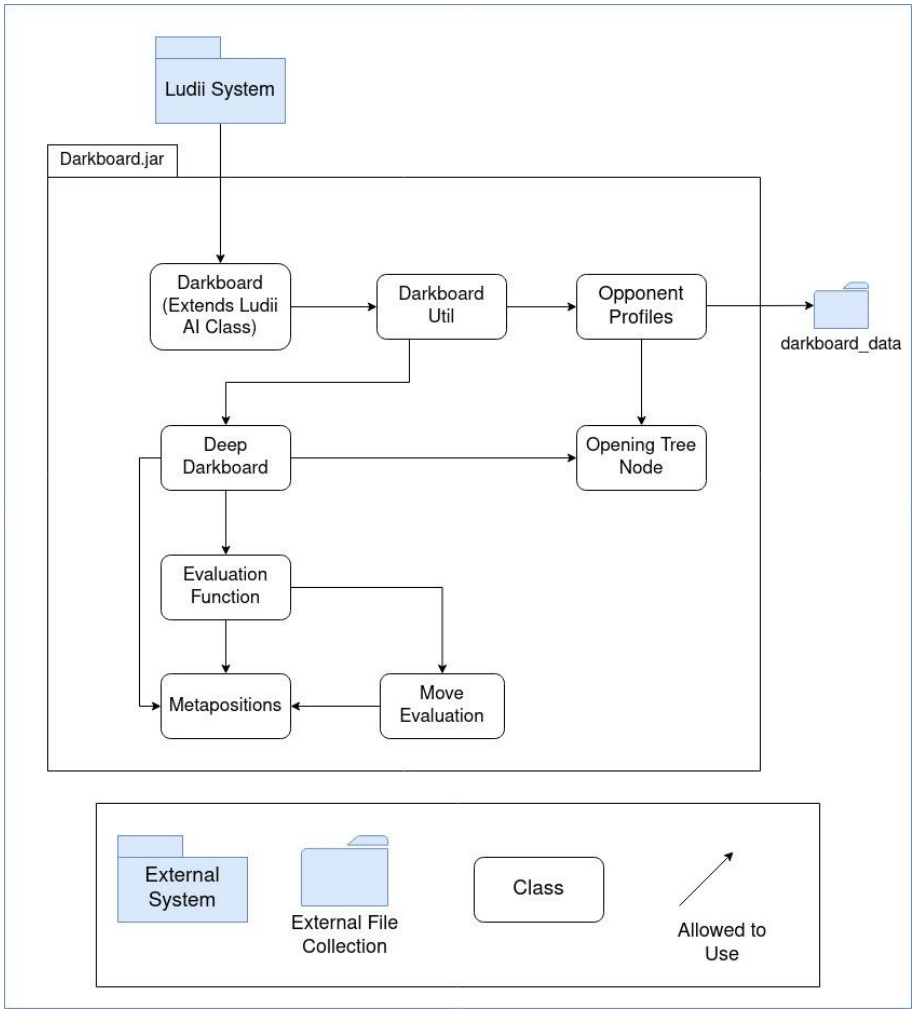


Figura 2.3: Darkboard su Ludii, diagramma statico. [Nov23]

un protocollo di comunicazione tra gli agenti e il sistema attraverso una nuova funzione chiamata `getNotes`, che da accesso ai messaggi dell'arbitro essenziali per stimare lo stato corrente del gioco. Questa funzione itera attraverso le azioni registrate dopo l'ultima mossa del giocatore e filtra quelle con il tag `note`, che corrisponde al nome della funzione di messaggio nel linguaggio ludeme. I messaggi estratti vengono poi aggiunti a una lista risultante, che viene restituita all'agente.

L'integrazione ha anche comportato l'incorporazione del codice di valutazione e ricerca di Darkboard, contenuto nella classe `DeepDarkboard101`, incluso l'uso di strutture dati specializzate come l'`OpeningProfile` per gestire le aperture, caricate da file esterni per influenzare le decisioni nelle prime fasi del gioco. L'implementazione include anche la gestione di scenari complessi come le promozioni dei pedoni e l'uso di libri di apertura, dimostrando la capacità di Ludii di supportare strategie di gioco avanzate tipiche degli

agenti di alto livello.

Le metaposizioni vengono aggiornate dopo ogni messaggio ricevuto dal sistema/arbitro attraverso metodi dedicati integrati con il sistema di stato del gioco offerto da Ludii. Questi metodi si occupano di aggiornare le informazioni sulle case visitate e sulla posizione dei pezzi in base alle indicazioni dei messaggi, generando tutte le possibili evoluzioni coerenti con le informazioni disponibili.

La classe `Agent` funge quindi da ponte, traducendo informazioni e comandi tra i due formati, permettendo di sfruttare appieno le capacità avanzate di Darkboard all'interno dell'architettura di Ludii.

2.3.2 KSV

Il programma KSV[gre] è un software progettato per giocare a Kriegspiel. KSV utilizza un sistema sofisticato per selezionare le mosse, basato su una serie di priorità. Tuttavia, un'analisi approfondita rivela che alcune delle sue strategie si basano su informazioni che non sarebbero normalmente disponibili in una partita reale di Kriegspiel, sollevando così questioni sulla validità del suo approccio.

Il cuore del processo decisionale di KSV si trova nel metodo decideMove(), che segue una serie di passaggi per selezionare la mossa ritenuta ottimale. Questo processo inizia con la ricerca di opportunità di promozione dei pedoni, dando priorità a quelle che coinvolgono potenziali catture. KSV esamina tutte le mosse possibili dei pedoni, verificando se si trovano nell'ultima traversa prima della promozione. Se un pedone può muoversi diagonalmente nell'ultima traversa, questa mossa viene considerata una potenziale promozione con cattura. Tuttavia, questo approccio è problematico nel contesto del Kriegspiel, poiché un giocatore reale non saprebbe se può catturare in diagonale senza tentare effettivamente la mossa.

Se non sono disponibili promozioni con cattura, KSV considera le promozioni senza cattura, garantendo così di sfruttare sempre le opportunità di promozione. In risposta a una promozione del pedone avversario, il programma implementa una strategia difensiva, cercando di muovere pezzi verso la prima traversa, con priorità al re se sotto scacco.

KSV ha anche strategie specifiche per gestire situazioni di scacco, cercando di muovere il re verso "caselle verdi" che rappresentano la direzione dello scacco. Implementa inoltre tattiche di contrattacco basate sulle informazioni di cattura e strategie difensive per evitare potenziali catture da parte dei pedoni avversari. Tuttavia, l'uso di informazioni dettagliate sulle "caselle di try" dell'avversario non è conforme alle regole del Kriegspiel, dove l'arbitro fornisce informazioni limitate sulle catture disponibili.

Il programma utilizza anche sequenze di mosse pre-programmate per gestire aperture e situazioni di fine partita, permettendogli di seguire strategie predefinite in fasi critiche. Se nessuna delle strategie precedenti è applicabile, KSV si concentra sull'avanzamento dei

pedoni, con una soglia di "significatività" che varia in base al numero di pezzi rimasti sulla scacchiera.

Come ultima risorsa, KSV seleziona una mossa casuale, cercando di evitare le caselle dove l'avversario poteva tentare una cattura con un pedone. Tuttavia, anche questo approccio è problematico secondo le regole del Kriegspiel, che non prevedono la comunicazione di informazioni così dettagliate sulle possibili catture.

Nel complesso, il processo decisionale di KSV nel Kriegspiel dimostra un approccio sofisticato e multi-livello alla selezione delle mosse, combinando strategie aggressive e difensive, e integrando sia piani pre-programmati che decisioni casuali. Per migliorare KSV, gli sviluppi futuri dovrebbero concentrarsi sull'adattare l'algoritmo per operare esclusivamente all'interno dei vincoli informativi imposti dalle regole ufficiali del Kriegspiel.

Capitolo 3

TRADUTTORE PARTITE TRIAL IN PGN

Il Traduttore TRIAL-to-PGN rappresenta un'innovativa soluzione software per la conversione di file partita dal formato Ludii Trial (.trl) 1.3.5 al formato PGN (Portable Game Notation) esteso 2.1.2 [Fav10]. L'obiettivo primario del convertitore è fornire un meccanismo robusto e flessibile per l'esportazione di partite Ludii in un formato ampiamente compatibile col software esistente e largamente utilizzato dalla comunità scacchistica.

Trial Ludii	**PGN Kriegspiel**
Move=[Move:mover=1,from=52,to=38,actions=[Select:typeFrom=Cell,from=52,typeTo=Cell,to=38,levelTo=0,decision=true],[SetVar:name=NextLost,value=0],[SetVar:name=DrawCondition,value=0],[Note:message=Illegalmove,to=1],[Note:message=Illegalmove,to=2],[SetNextPlayer:player=1]]Move=[Move:mover=1,from=48,to=50,actions=[Select:typeFrom=Cell,from=48,typeTo=Cell,to=50,decision=true],[SetVar:name=NextLost,value=0],[SetVar:name=DrawCondition,value=0],[SetVar:name=EnPassantLocation,value=-1],[Move:typeFrom=Cell,from=48,typeTo=Cell,to=50,decision=true],[Note:message=Filecheck,to=1],[Note:message=Filecheck,to=2],[SetVar:name=NextLost,value=1],[SetScore:player=2,score=0],[SetState:type=Cell,to=50,state=0]]	Qac7 {CF:Qe7-g5}
Move=[Move:mover=2,from=12,to=5,actions=[Select:typeFrom=Cell,from=12,typeTo=Cell,to=5,levelTo=0,decision=true],[SetVar:name=NextLost,value=0],[SetVar:name=DrawCondition,value=0],[SetVar:name=CapturedPiece,value=7],[Move:typeFrom=Cell,from=12,typeTo=Cell,to=5,decision=true],[Note:message=PieceatF1captured,to=1],[Note:message=PieceatF1captured,to=2],[SetNextPlayer:player=2],[SetScore:player=1,score=0],[SetCounter:counter=-1],[SetVar:name=EnPassantLocation,value=-1]]Move=[Move:mover=2,from=5,to=5,actions=[Promote:type=Cell,to=5,what=12,decision=true],[Note:message=Rankcheck,to=1],[Note:message=Rankcheck,to=2],[SetScore:player=1,score=0]]	exf1=Q {Xf1:}
winner=0 endtype=NaturalEnd rankings=0.0,1.5,1.5	1/2-1/2

Tabella 3.1: Confronto tra il formato Trial di Ludii e il formato PGN esteso del Kriegspiel

3.1 Architettura del Software

Il convertitore è implementato come un'applicazione a riga di comando in Python. La scelta di questo linguaggio di implementazione permette di sfruttare una vasta gamma di librerie e strumenti per l'analisi del testo scacchistico, la manipolazione di strutture dati complesse e la creazione di interfacce utente intuitive.

Il cuore del convertitore è costituito da una serie di moduli funzionali, ciascuno dedicato a un aspetto specifico del processo di conversione:

- **Modulo di Parsing (`parse_ludii_move`)**: Questo componente è responsabile dell'interpretazione del formato proprietario di Ludii. Utilizza tecniche di pattern matching basate su espressioni regolari per estrarre informazioni quali il nome del giocatore, le coordinate di partenza e arrivo della mossa, eventuali catture o promozioni, e i messaggi speciali forniti dall'arbitro. Il traduttore è capace di gestire le peculiarità del Kriegspiel, inclusa la gestione di mosse illegali e la normalizzazione delle mosse di promozione.
- **Gestione della Scacchiera (`setup_board`, `update_board`)**: Questi moduli sono responsabili della rappresentazione interna dello stato della partita. Utilizzano strutture dati efficienti, come dizionari Python, per memorizzare e manipolare lo stato della scacchiera. `setup_board` si occupa dell'inizializzazione della posizione di partenza, mentre `update_board` gestisce l'aggiornamento dinamico dello stato dopo ogni mossa.
- **Generazione PGN (`generate_pgn_move`, `convert_kriegspiel`)**: Questi moduli specializzati si occupano della conversione delle mosse dal formato Ludii al formato PGN. `generate_pgn_move` fornisce la logica di base per la generazione di mosse PGN, mentre `convert_kriegspiel` implementa la logica specifica per gestire le complessità del Kriegspiel, come le mosse illegali e le informazioni parziali dell'arbitro.
- **Costruzione Output (`build_pgn_header`, `build_pgn_moves_with_notes`)**: Questi componenti sono responsabili della generazione dell'output PGN finale. `build_pgn_header` si occupa della creazione dell'intestazione PGN con metadati della partita, mentre `build_pgn_moves_with_notes` assembla il corpo principale del PGN, incorporando mosse e annotazioni.
- **Sistema di Logging e Debug (`debug_print`)**: Controllato da un flag globale `DEBUG`, questo sistema permette una granularità fine nella registrazione di eventi e stati intermedi del processo di conversione per facilitare il debugging e l'analisi del processo di conversione.

3.1.1 Flusso di Elaborazione

Il flusso di elaborazione del convertitore segue un approccio pipeline, dove ogni componente elabora i dati in modo sequenziale:

1. Inizializzazione e parsing degli argomenti da riga di comando.
2. Lettura e parsing del file di input Ludii.
3. Inizializzazione della rappresentazione interna della scacchiera.
4. Iterazione attraverso le mosse, con parsing e conversione di ciascuna mossa.
5. Generazione dell'output PGN, inclusi header e corpo delle mosse.
6. Scrittura dell'output su file e finalizzazione del processo.

Questo approccio modulare non solo migliora la leggibilità e la manutenibilità del codice, ma facilita anche l'estensione del software in futuro per supportare nuove varianti di Scacchi o formati di input/output.

3.2 Parsing degli Input

Il parsing degli input rappresenta una fase rilevante nel processo di conversione, in quanto il formato .trl utilizzato da Ludii presenta caratteristiche e strutture specifiche che richiedono un'interpretazione accurata e robusta. La funzione `parse_ludii_move` costituisce il fulcro di questo processo: implementa un algoritmo di pattern matching basato su espressioni regolari.

3.2.1 Tecniche di Pattern Matching

L'utilizzo della libreria `re` di Python permette l'implementazione di pattern matching, e consente l'estrazione di informazioni utili da stringhe di testo complesse. I pattern utilizzati sono stati ottimizzati per catturare efficacemente:

- Identificazione del giocatore (`mover=(`$\dot{+}$`)`)
- Coordinate di partenza e arrivo (`from=(`$\dot{+}$`)`, `to=(`$\dot{+}$`)`)
- Rilevamento di catture (`'Remove:'` o `'CapturedPiece'`)
- Identificazione di promozioni (`'Promote:'`)
- Estrazione di note dell'arbitro (`[Note:message=(.*?),to=(`$\dot{+}$`)]`)

3.2.2 Gestione delle Mosse Illegali

Una delle sfide più significative nel parsing delle mosse del Kriegspiel è la gestione delle mosse illegali. Data la natura del gioco, dove i giocatori hanno informazioni incomplete sulla posizione, è comune che vengano tentate mosse non valide. Il parser implementa una logica sofisticata per:

- Rilevare e isolare le mosse illegali
- Estrarre informazioni rilevanti da queste mosse (es. pezzo mosso, coordinate tentate)
- Incorporare queste informazioni nelle annotazioni PGN per una rappresentazione fedele della partita

3.2.3 Normalizzazione delle Promozioni

Il parser gestisce anche la discrepanza nella rappresentazione delle promozioni tra il formato di Ludii e il PGN. In Ludii, le promozioni sono rappresentate come mosse separate, mentre in PGN devono essere incorporate nella mossa principale. Il parser implementa un meccanismo di "look-ahead" per:

- Identificare mosse di promozione
- Fondere le informazioni di promozione con la mossa principale
- Generare una singola rappresentazione PGN coerente per l'intera sequenza di promozione

3.2.4 Raggruppamento delle Note dell'Arbitro

Un'altra caratteristica avanzata del parser è la capacità di raggruppare e consolidare le note dell'arbitro. Questo è particolarmente rilevante nel Kriegspiel, dove l'arbitro fornisce informazioni critiche ai giocatori. Il parser:

- Identifica note multiple relative alla stessa mossa
- Raggruppa queste note in base al destinatario (giocatore 1, giocatore 2, o entrambi)
- Genera una rappresentazione consolidata delle note per una maggiore chiarezza nel PGN risultante

L'implementazione di queste tecniche avanzate di parsing assicura che il convertitore possa gestire efficacemente la complessità e le peculiarità sia degli scacchi standard che del Kriegspiel, producendo un output PGN accurato e ricco di informazioni.

3.3 Gestione della scacchiera

La gestione efficiente e accurata dello stato della scacchiera è fondamentale per il corretto funzionamento del convertitore. Il software implementa un sofisticato sistema di rappresentazione e aggiornamento della scacchiera, utilizzando strutture dati ottimizzate e algoritmi efficienti per garantire prestazioni elevate anche con partite lunghe o complesse.

3.3.1 Rappresentazione interna della scacchiera

La scacchiera è rappresentata internamente utilizzando un dizionario Python, una scelta che offre un compromesso tra efficienza di accesso e flessibilità di manipolazione. La struttura del dizionario è la seguente:

- **Chiavi**: Stringhe rappresentanti le coordinate in notazione algebrica (es. "e4")
- **Valori**: Interi rappresentanti i pezzi, codificati secondo uno schema predefinito

Questa rappresentazione permette:

- Accesso O(1) allo stato di qualsiasi casella
- Facile aggiornamento e manipolazione dello stato della scacchiera
- Efficiente iterazione attraverso i pezzi presenti

3.3.2 Inizializzazione della Scacchiera

La funzione `setup_board` è responsabile dell'inizializzazione della scacchiera. Questo processo comporta:

1. Parsing delle mosse di setup nel file Ludii
2. Creazione della rappresentazione iniziale della scacchiera
3. Validazione della configurazione iniziale contro una posizione standard attesa

Il processo di validazione è particolarmente importante, in quanto permette di rilevare eventuali incongruenze nel file di input o errori nel processo di parsing.

3.3.3 Aggiornamento Dinamico

La funzione `update_board` gestisce l'aggiornamento della scacchiera dopo ogni mossa. Questa funzione implementa una logica sofisticata per:

- Spostare pezzi da una casella all'altra
- Gestire catture, rimuovendo i pezzi catturati dalla scacchiera
- Implementare regole speciali per tenere conto di casi come:
  - **Arrocco**: Sposta sia il re che la torre nelle posizioni appropriate
  - **En Passant**: Gestisce la cattura speciale del pedone, rimuovendo il pedone catturato dalla casella corretta
  - **Promozione**: Sostituisce il pedone con il pezzo promosso sulla casella di arrivo

3.4 Generazione delle Mosse PGN

La generazione accurata e dettagliata delle mosse in formato PGN è un aspetto cruciale del convertitore, in particolare per la variante Kriegspiel che richiede annotazioni estese. Il processo di generazione delle mosse PGN è gestito principalmente dalle funzioni `generate_pgn_move` e `convert_kriegspiel`, che implementano algoritmi sofisticati per tradurre le mosse Ludii in notazione PGN standard e estesa.

3.4.1 Generazione di Base delle Mosse

La funzione `generate_pgn_move` implementa la logica fondamentale per la conversione delle mosse in formato PGN. Questo processo include:

- Identificazione del pezzo mosso
- Generazione della notazione algebrica della mossa (es. "e4", "Nf3")
- Gestione di casi speciali come catture, promozioni e arrocco
- Risoluzione di ambiguità quando più pezzi dello stesso tipo possono muovere nella stessa casella

L'algoritmo utilizza la rappresentazione interna della scacchiera per determinare la notazione corretta, implementando regole sofisticate per gestire tutti i possibili scenari di mossa.

3.4.2 Gestione delle Peculiarità del Kriegspiel

La funzione `convert_kriegspiel` estende le capacità di base di `generate_pgn_move` per gestire le complessità uniche del Kriegspiel:

- **Tracciamento delle Mosse Illegali**: Implementa un sistema di buffer per accumulare e annotare le mosse illegali tentate prima di una mossa valida. Queste mosse vengono poi incorporate nel PGN come commenti, fornendo una rappresentazione fedele del processo decisionale del giocatore.
- **Incorporazione delle Informazioni dell'Arbitro**: Traduce i messaggi dell'arbitro in annotazioni PGN standardizzate. Utilizza un sistema di codifica per rappresentare efficacemente informazioni come catture, scacchi e tentativi di cattura dei pedoni.
- **Gestione dei Tentativi di Cattura dei Pedoni**: Implementa l'algoritmo `calculate_pawn_tries` per analizzare la posizione e determinare tutti i possibili tentativi di cattura dei pedoni, incluse le catture en passant. Queste informazioni vengono codificate nelle annotazioni PGN.

3.4.3 Algoritmo di Calcolo dei Tentativi di Cattura dei Pedoni

L'algoritmo `calculate_pawn_tries` merita una menzione speciale per la sua complessità e importanza nel contesto del Kriegspiel:

1. **Analisi della Posizione**: Esamina la scacchiera dopo ogni mossa per identificare potenziali posizioni di cattura per i pedoni.
2. **Gestione dell'En Passant**: Implementa una logica speciale per rilevare le opportunità di cattura en passant, considerando la mossa precedente dell'avversario.
3. **Calcolo delle Diagonali**: Per ogni pedone, calcola le caselle diagonali adiacenti dove potrebbe potenzialmente catturare.
4. **Verifica della Legalità**: Applica le regole del Kriegspiel per determinare quali tentativi di cattura sarebbero considerati legali dal punto di vista del giocatore con informazioni limitate.
5. **Generazione delle Annotazioni**: Produce una stringa codificata che rappresenta il numero dei tentativi di cattura possibili.

3.4.4 Generazione di Annotazioni Estese

Il programma implementa un sistema avanzato per la generazione di annotazioni PGN estese, standard per rappresentare accuratamente le partite di Kriegspiel:

- **Codifica delle Informazioni dell'Arbitro**: Utilizza un sistema di simboli standardizzati per rappresentare le varie informazioni fornite dall'arbitro (es. "X" per cattura, "C" per scacco).

- **Incorporazione di Mosse Illegali**: Le mosse illegali vengono aggiunte come commenti tra parentesi graffe, seguendo la notazione standard PGN per i commenti.
- **Annotazione dei Tentativi di Cattura**: I risultati di `calculate_pawn_tries` vengono incorporati nelle annotazioni, fornendo informazioni cruciali sulla tensione posizionale nel Kriegspiel.

3.4.5 Supporto per Varianti Multiple

Il sistema di generazione delle mosse è progettato per essere facilmente estensibile a nuove varianti di scacchi:

- **Architettura Modulare**: Le funzioni di generazione delle mosse sono organizzate in modo modulare, permettendo l'aggiunta di nuove varianti con minime modifiche al codice esistente.
- **Interfaccia Unificata**: Implementa un'interfaccia comune per la generazione delle mosse, che può essere specializzata per varianti specifiche.
- **Sistema di Plugin**: Utilizza un sistema basato su plugin che permette di caricare dinamicamente il generatore di mosse appropriato in base alla variante di gioco rilevata.

Questa flessibilità assicura che il convertitore possa essere facilmente adattato per supportare future varianti di scacchi o modifiche alle regole esistenti.

3.4.5.1 Adattabilità agli Scacchi Standard

Sebbene il focus principale sia il Kriegspiel, il programma è progettato per essere adattabile anche agli scacchi standard. Gran parte delle funzionalità di base, come il parsing delle mosse, la gestione della scacchiera e la generazione delle mosse PGN, sono condivise tra le due varianti.

Per gestire gli scacchi standard, il convertitore include una funzione dedicata chiamata `convert_chess`. Questa funzione segue un flusso di lavoro simile a `convert_kriegspiel`, ma senza le complessità specifiche del Kriegspiel come le mosse illegali, le informazioni dell'arbitro e i tentativi di cattura dei pedoni.

La funzione `parse_ludii_move` è in grado di riconoscere automaticamente se una partita è Kriegspiel o scacchi standard analizzando la prima riga del file Ludii, che indica la variante di gioco. In base a questa informazione, il programma sceglie la funzione di conversione appropriata (`convert_kriegspiel` o `convert_chess`).

Durante la generazione dell'output PGN per gli scacchi standard, il programma utilizza le stesse funzioni ma senza includere le annotazioni specifiche del Kriegspiel. Il risultato è

un PGN standard che può essere facilmente importato e analizzato in qualsiasi software di scacchi compatibile.

3.5 Interfaccia Utente e Gestione dell'Input

L'interfaccia utente è stata progettata per offrire un'esperienza intuitiva e amichevole, adatta sia a utenti tecnici che non tecnici. Il sistema implementa un approccio multimodale che combina un'interfaccia a linea di comandi (CLI) con un'interfaccia grafica utente (GUI) opzionale, così garantendo la massima versatilità d'uso.

3.5.1 Interfaccia a linea di Comandi

L'interfaccia CLI è implementata utilizzando il modulo `argparse` di Python, che permette una gestione diretta degli argomenti da linea di comando:

- **Parsing degli Argomenti**: Implementa un sistema di parsing degli argomenti che supporta opzioni brevi e lunghe (es. `-f` e `--file`) per offrire flessibilità.
- **Validazione degli Input**: Incorpora controlli di validazione per assicurare che gli argomenti forniti siano corretti e completi.
- **Gestione di Input Multipli**: Supporta la specifica di file di input multipli per la conversione batch di più partite o tornei.
- **Opzioni Configurabili**: Offre opzioni per personalizzare il processo di conversione, come la specifica dei nomi dei giocatori.

3.5.2 Interfaccia Grafica per l'Utente (GUI)

La GUI opzionale, implementata utilizzando la libreria `tkinter`, offre un'alternativa user-friendly per gli utenti meno familiari con i comandi testuali:

- **Selezione File Intuitiva**: Implementa finestre di dialogo per la selezione dei file di input e output, il che semplifica l'uso per gli utenti meno esperti.
- **Configurazione Visuale**: Offre campi e opzioni interattive per la configurazione dei parametri di conversione.
- **Feedback in Tempo Reale**: Fornisce feedback visuale durante il processo di conversione, inclusi indicatori di progresso e notifiche di completamento.
- **Gestione degli Errori**: Implementa finestre di dialogo per la visualizzazione di messaggi di errore in modo chiaro e user-friendly.

3.5.3 Sistema di Input

Il convertitore implementa un sistema di input che si adatta dinamicamente alle capacità del sistema e alle preferenze dell'utente:

- **Rilevamento Automatico**: Utilizza tecniche di rilevamento per determinare se il contesto supporta l'interfaccia grafica.
- **Fallback Graceful**: Se l'interfaccia grafica non è disponibile, il sistema passa automaticamente alla modalità CLI senza interruzioni.
- **Input Interattivo**: In assenza di argomenti da linea di comando o selezioni GUI, il sistema entra in una modalità interattiva, e guida l'utente attraverso prompt testuali.

3.5.4 Gestione Avanzata dei File di Input

Il sistema implementa funzionalità specifiche per la gestione dei file di input, particolarmente utili per la conversione di tornei o serie di partite:

- **Ordinamento Intelligente**: La funzione `order_files_gui` offre un'interfaccia visuale per l'ordinamento manuale dei file di input, utile per mantenere la corretta sequenza in un torneo.
- **Raggruppamento Automatico**: Implementa algoritmi per il raggruppamento automatico di file correlati, basandosi su pattern nei nomi dei file o nei metadati.
- **Validazione dei File**: Esegue controlli preliminari sui file di input per assicurare la compatibilità e la completezza prima di iniziare la conversione.

3.5.5 Gestione dei Nomi dei Giocatori

Il convertitore offre un sistema di gestione dei nomi dei giocatori, utile per la corretta annotazione delle partite in un torneo:

- **Input Flessibile**: Supporta l'inserimento dei nomi dei giocatori tramite CLI o GUI.
- **Persistenza dei Dati**: Implementa un sistema per memorizzare e riutilizzare i nomi dei giocatori tra più conversioni in una sessione.
- **Rotazione Automatica**: Gestisce automaticamente lo scambio dei colori dei giocatori tra le partite in un torneo.
- **Validazione dei Nomi**: Implementa controlli per assicurare la coerenza e la correttezza dei nomi dei giocatori in tutto il torneo.

3.6 Logging e Debugging Avanzato

Il sistema di logging e debugging implementato rappresenta un componente critico per garantire la robustezza, la manutenibilità e l'estensibilità del software. Questo sistema è stato progettato per fornire rapporti dettagliati sul processo di conversione, facilitando l'identificazione e la risoluzione dei problemi, particolarmente frequenti nella gestione delle peculiarità del Kriegspiel.

3.6.1 Architettura del Sistema di Logging

Il sistema di logging è basato su un'architettura multi-livello che offre flessibilità e granularità nel controllo delle informazioni registrate:

- **Logging Condizionale**: Utilizza la variabile globale `DEBUG` per attivare o disattivare selettivamente il logging dettagliato, permettendo un controllo fine sulle prestazioni e sull'output.
- **Logging Contestuale**: Implementa un sistema di logging contestuale che associa automaticamente i messaggi alle funzioni e ai moduli rilevanti, facilitando la tracciabilità.
- **Rotazione dei Log**: Utilizza tecniche di rotazione dei file di log per gestire efficacemente grandi volumi di dati di debug senza compromettere le prestazioni o lo spazio su disco.

3.6.2 Visualizzazione dello Stato della Scacchiera

Per facilitare il debugging delle partite di scacchi, specialmente nel contesto del Kriegspiel, il sistema implementa funzionalità avanzate di visualizzazione dello stato della scacchiera:

- **Rappresentazione ASCII**: La funzione `print_board` genera una rappresentazione ASCII della scacchiera, permettendo una rapida ispezione visiva dello stato del gioco.
- **Diff Visuale**: Implementa un sistema di differenziazione visuale che evidenzia i cambiamenti nella scacchiera tra mosse successive, facilitando l'identificazione di mosse critiche o errori subtili.
- **Visualizzazione delle Informazioni Nascoste**: Nel contesto del Kriegspiel, implementa una visualizzazione duale che mostra contemporaneamente lo stato reale della scacchiera e la percezione di ciascun giocatore, evidenziando le discrepanze.
- **Timeline Interattiva**: Sviluppa una rappresentazione grafica della sequenza di mosse, permettendo agli sviluppatori di navigare avanti e indietro attraverso la partita per un'analisi dettagliata.

3.6.3 Logging Specifico per il Kriegspiel

Il sistema di logging implementa funzionalità specializzate per gestire le complessità uniche del Kriegspiel:

- **Tracciamento delle Informazioni dell'Arbitro**: Registra dettagliatamente tutte le informazioni fornite dall'arbitro, permettendo un'analisi approfondita della progressione dell'informazione nel gioco.
- **Log delle Mosse Illegali**: Implementa un sistema di logging dedicato per le mosse illegali tentate, cruciale per comprendere il processo decisionale dei giocatori in condizioni di informazione incompleta.
- **Analisi dei Tentativi di Cattura**: Registra e analizza tutti i potenziali tentativi di cattura calcolati dalla funzione `calculate_pawn_tries`, fornendo insight sulle dinamiche tattiche nascoste del gioco.
- **Stato di Conoscenza del Giocatore**: Mantiene un log dettagliato dello stato di conoscenza di ciascun giocatore ad ogni mossa, essenziale per debuggare la logica di gioco del Kriegspiel.

Capitolo 4

CONCLUSIONI

L'integrazione del Kriegspiel all'interno della piattaforma Ludii ha dimostrato la flessibilità e la potenza di questo ambiente *general-game*. Nonostante le sfide poste dalla natura a informazione imperfetta del Kriegspiel, Ludii si è rivelato capace di modellare efficacemente le regole del gioco, il ruolo dell'arbitro e la gestione dell'incertezza. Questo successo sottolinea il potenziale di Ludii come piattaforma per lo studio e l'implementazione di una vasta gamma di giochi, inclusi quelli con meccaniche complesse e non convenzionali.

Lo sviluppo del traduttore TRIAL-to-PGN ha affrontato e risolto diverse sfide tecniche, in particolare la necessità di catturare accuratamente le peculiarità del Kriegspiel all'interno del formato PGN standard. Il risultato è uno strumento robusto e flessibile che non solo facilita la conversione delle partite, ma fornisce anche un'interfaccia tra il mondo di Ludii e gli strumenti di analisi scacchistica più diffusi.

Un aspetto critico emerso durante lo studio della piattaforma è stata la gestione delle potenziali falle che potrebbero permettere ai giocatori di barare, come l'accesso non autorizzato alle informazioni complete sullo stato del gioco attraverso l'API di Ludii. Per affrontare questo problema, si propone una soluzione che potrebbe essere implementata in futuri aggiornamenti di Ludii:

1. **Modifica del file ludeme:** Nel file ludeme che definisce la modalità di gioco (in questo caso, il Kriegspiel), si potrebbe introdurre una sezione specifica per definire il comportamento delle funzioni di utilità AI fornite da Ludii.

2. **Restrizioni sulle funzioni:** Per il Kriegspiel, si potrebbe specificare che funzioni come quelle che restituiscono mosse legali o lo stato completo della scacchiera siano limitate. In particolare:

    - Le funzioni che restituiscono mosse legali dovrebbero essere modificate per restituire solo "pseudo-mosse", ovvero mosse che appaiono legali dal punto di vista del giocatore con informazioni limitate.

- Le funzioni che forniscono lo stato della scacchiera dovrebbero restituire solo lo stato conosciuto dal bot giocatore, escludendo le informazioni sull'avversario non direttamente osservabili.

3. **Implementazione di controlli runtime:** Ludii potrebbe implementare controlli durante l'esecuzione per assicurare che gli agenti AI rispettino queste restrizioni, bloccando o segnalando eventuali tentativi di accesso a informazioni non autorizzate.

Queste modifiche non solo rafforzerebbero l'integrità del gioco nel Kriegspiel, ma fornirebbero anche un framework più robusto per l'implementazione in Ludii di altri giochi a informazione imperfetta. Ciò garantirebbe che gli agenti AI sviluppati per questi giochi siano veramente rappresentativi delle sfide poste dall'incertezza e dall'informazione incompleta, senza la possibilità di sfruttare involontariamente informazioni non disponibili in una partita reale.

L'analisi comparativa degli approcci di diversi avversari artificiali, come Darkboard e KSV, ha evidenziato la complessità delle strategie necessarie per giocare efficacemente al Kriegspiel. Mentre alcuni approcci dimostrano sofisticate tecniche di gestione dell'incertezza, altri rivelano potenziali problemi etici nell'uso di informazioni non disponibili in una partita reale. In questo contesto, il lavoro di Heinrich e Silver sul Neural Fictitious Self-Play (NFSP) [HS16] offre una prospettiva interessante. A differenza di approcci come Darkboard, che richiedono lo studio e l'implementazione di euristiche specifiche, NFSP utilizza il *deep reinforcement learning* per affrontare giochi ad informazione imperfetta senza la necessità di conoscenze specifiche del dominio. Questo metodo apprende direttamente dall'esperienza di gioco, potenzialmente eliminando la necessità di sviluppare complesse euristiche manuali per il Kriegspiel.

In conclusione, il ponte creato tra Ludii e il formato PGN, insieme alle proposte per migliorare l'integrità del gioco a informazione incompleta in Ludii, rappresenta un passo significativo verso una maggiore accessibilità, standardizzazione e robustezza nello studio di giochi complessi, aprendo nuove possibilità per la ricerca e lo sviluppo in questo affascinante campo rendendo più accessibili i dati delle partite a una comunità più ampia di ricercatori e appassionati.

Riferimenti bibliografici

[AS23] Iulia Armegioiu and Carl Swanberg. Imperfect Information in Chess Variants and Changes in Player Strategies and Perceptions. B.S. thesis, University of Uppsala, 2023.

[Ass21] Jorrit Assen. General Game Playing with imperfect information in GROOVE. B.S. thesis, University of Twente, 2021.

[BC06] Andrea Bolognesi and Paolo Ciancarini. Searching over Metapositions in Kriegspiel. In J. van den Herik et al., editors, *Revised papers from 4th Int. Conf. on Computer and Games*, number 3846 in LNCS, pages 246–261. Springer, 2006.

[BPSS23] 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.

[Bur67] John F Burger. UMPIRE: An automatic kriegsspiel referee for a time-shared computer. In *Proceedings of the 1967 22nd National Conference*, pages 187–193, 1967.

[Caz06] Tristan Cazenave. A Phantom-Go program. In *Proc. 11th Int. Conf. Advances in Computer Games*, volume 4250 of *LNCS*, pages 120–125. Springer, 2006.

[CDM97] Paolo Ciancarini, Francesco DallaLibera, and Fabio Maran. Decision making under uncertainty: a rational approach to Kriegspiel. *Advances in Computer Chess*, 8:277–298, 1997.

[CF07] Paolo Ciancarini and Gian Piero Favini. A Program to Play Kriegspiel. *ICGA Journal*, 30(1):3–24, 2007.

[CF10a] Paolo Ciancarini and Gian Piero Favini. Monte Carlo tree search in Kriegspiel. *Artificial Intelligence*, 174(11):670–684, 2010.

[CF10b] Paolo Ciancarini and Gian Piero Favini. Playing the perfect Kriegspiel endgame. *Theoretical computer science*, 411(40-42):3563–3577, 2010.

[CN24] Paolo Ciancarini and Nikola Novarlic. Kriegspiel on the Ludii Platform, 2024.

[CS23] Walter Crist and Dennis JNJ Soemers. The Digital Ludeme Project: Combining archaeological and computational methods for the study of ancient board games. *Journal of Archaeological Science: Reports*, 49:104005, 2023.

[Fav10] Gian Piero Favini. *The dark side of the board: advances in chess Kriegspiel*. PhD thesis, University of Bologna, Italy, 2010.

[GLP05] Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the aaai competition. *AI magazine*, 26(2):62–62, 2005.

[gre] greendate. Kriegspiel ksv. `https://github.com/greendate/KriegspielKSV`.

[HS16] Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*, 2016.

[KP95] Daphne Koller and Avi Pfeffer. Generating and solving imperfect information games. In *IJCAI*, pages 1185–1193. Citeseer, 1995.

[LLL+19] Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, et al. Openspiel: A framework for reinforcement learning in games. arXiv preprint arXiv:1908.09453, 2019.

[Mur13] Harold Murray. *A History of Chess*. Oxford University Press, 1913.

[Nov23] Nikola Novarlic. Implementing Kriegspiel on the Ludii General Game System. Master's thesis, Innopolis University, 2023. Supervisor: Paolo Ciancarini.

[PSS+20] É. Piette, D. J. N. J. Soemers, M. Stephenson, C. F. Sironi, M. H. M. Winands, and C. Browne. Ludii – the ludemic general game system. In G. De Giacomo, A. Catala, B. Dilkina, M. Milano, S. Barro, A. Bugarín, and J. Lang, editors, *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 411–418. IOS Press, 2020.

[PSSB19] Eric Piette, Matthew Stephenson, Dennis J.N.J. Soemers, and Cameron Browne. An Empirical Evaluation of Two General Game Systems: Ludii and RBG. In *Proc. First Conference on Games (CoG)*. IEEE, 2019.

[SPSB24] Dennis JNJ Soemers, Eric Piette, Matthew Stephenson, and Cameron Browne. The Ludii game description language is universal, 2024.

[ST19] Michael Schofield and Michael Thielscher. General game playing with imperfect information. *Journal of Artificial Intelligence Research*, 66:901–935, 2019.

[Thi10] Michael Thielscher. A general game description language for incomplete information games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pages 994–999, 2010.

[WBB72] Charles S Wetherell, TJ Buckholtz, and Kellogg S. Booth. A director for Kriegspiel, a variant of chess. *The Computer Journal*, 15(1):66–70, 1972.

[YT18] Georgios Yannakakis and Julian Togelius. *Artificial Intelligence and Games*. Springer, 2018.

[Zil98] Zillions Development Corporation. Zillions of games, 1998.

Ringraziamenti

Desidero esprimere la mia più profonda gratitudine al Professor Paolo Ciancarini per la sua straordinaria disponibilità e guida durante tutto il percorso di questa tesi. La sua esperienza e i suoi consigli sono stati fondamentali per la realizzazione di questo lavoro.

Un ringraziamento speciale va a Nikola Novarlic, il cui supporto ha contribuito in modo significativo allo sviluppo del programma.

Vorrei anche ringraziare calorosamente i miei colleghi di corso, che sono stati un costante punto di riferimento nei momenti di difficoltà. La loro amicizia ha reso questo percorso non solo più gestibile, ma anche più ricco e gratificante.

Infine, il mio più sentito ringraziamento va alla mia famiglia e ai miei cari. Senza il loro sostegno, nulla di tutto questo sarebbe stato possibile.