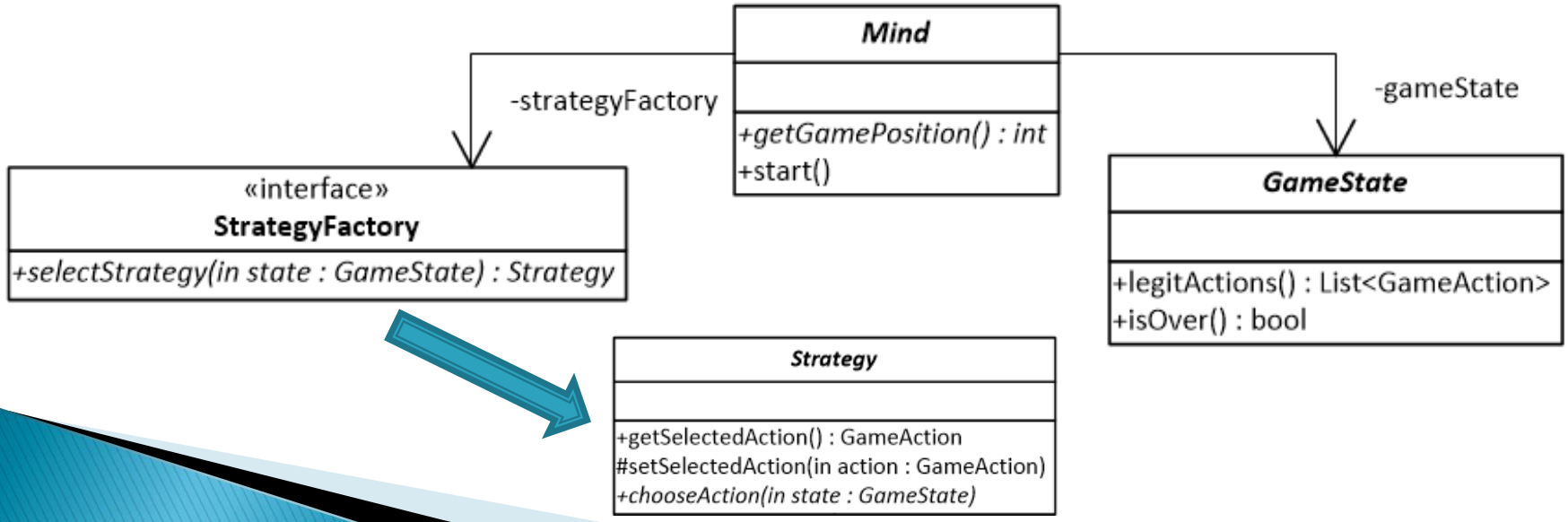
MulinoException

Mulino Challenge 2018

di Filippo Frabetti, Paolo Magnani e Nicola Semprini

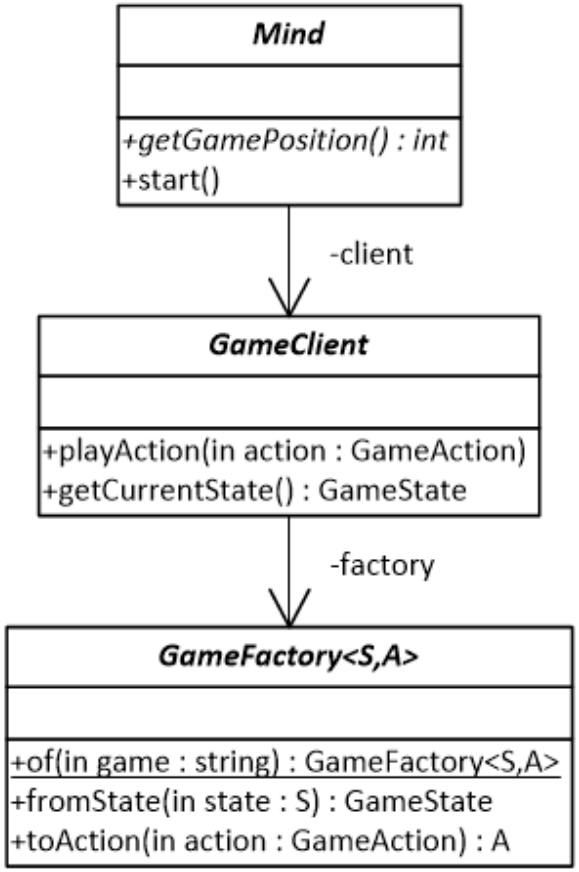
Architettura del giocatore

- La **Mind** è la parte attiva del giocatore: essa gestisce le fasi di gioco e funge da main thread per l'applicazione
- Possiede una **StrategyFactory** che si occupa di scegliere la strategia migliore in base allo stato
- Grazie allo stato di gioco corrente(**GameState**), la **Strategy** può scegliere la mossa



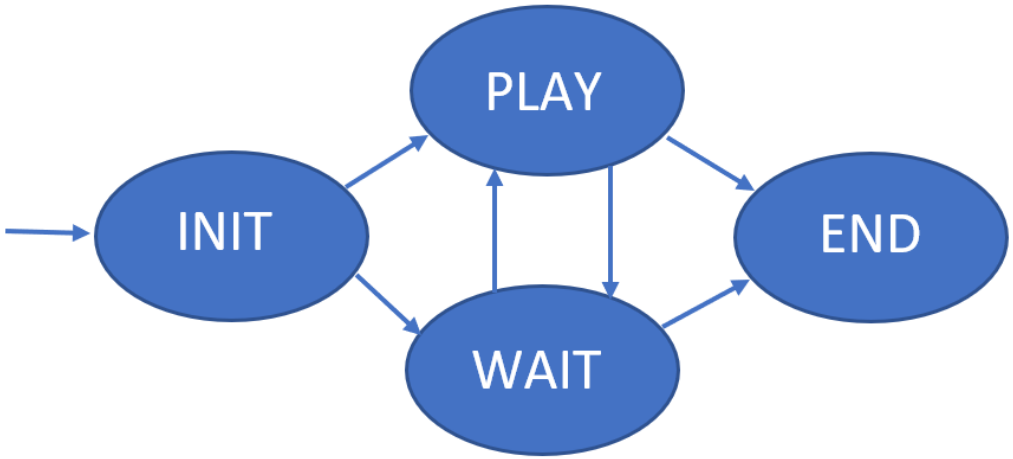
Client e Server (I/O)

- La Mind invia le mosse decise e riceve lo stato di gioco attraverso un **GameClient**, che fornisce **primitive di I/O**
- Una **GameFactory** effettua le conversioni fra la rappresentazione degli stati/azioni lato server e quella usata internamente



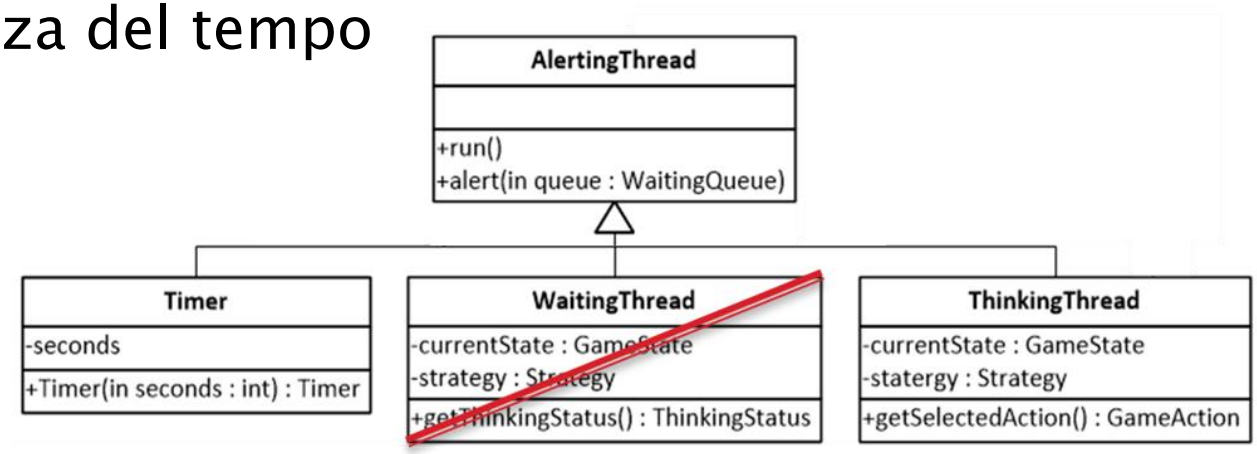
Modularità e riusabilità

- L'architettura è **generale**: ci sono **classi astratte** e **interfacce** che permettono, con le opportune implementazioni, di giocare a tris, dama, scacchi, ecc…
- La Mind infatti coordina le mosse del giocatore con i **4 stati** tipici di giochi 1 vs 1 a turni: **Init**, **Play**, **Wait** e **End**

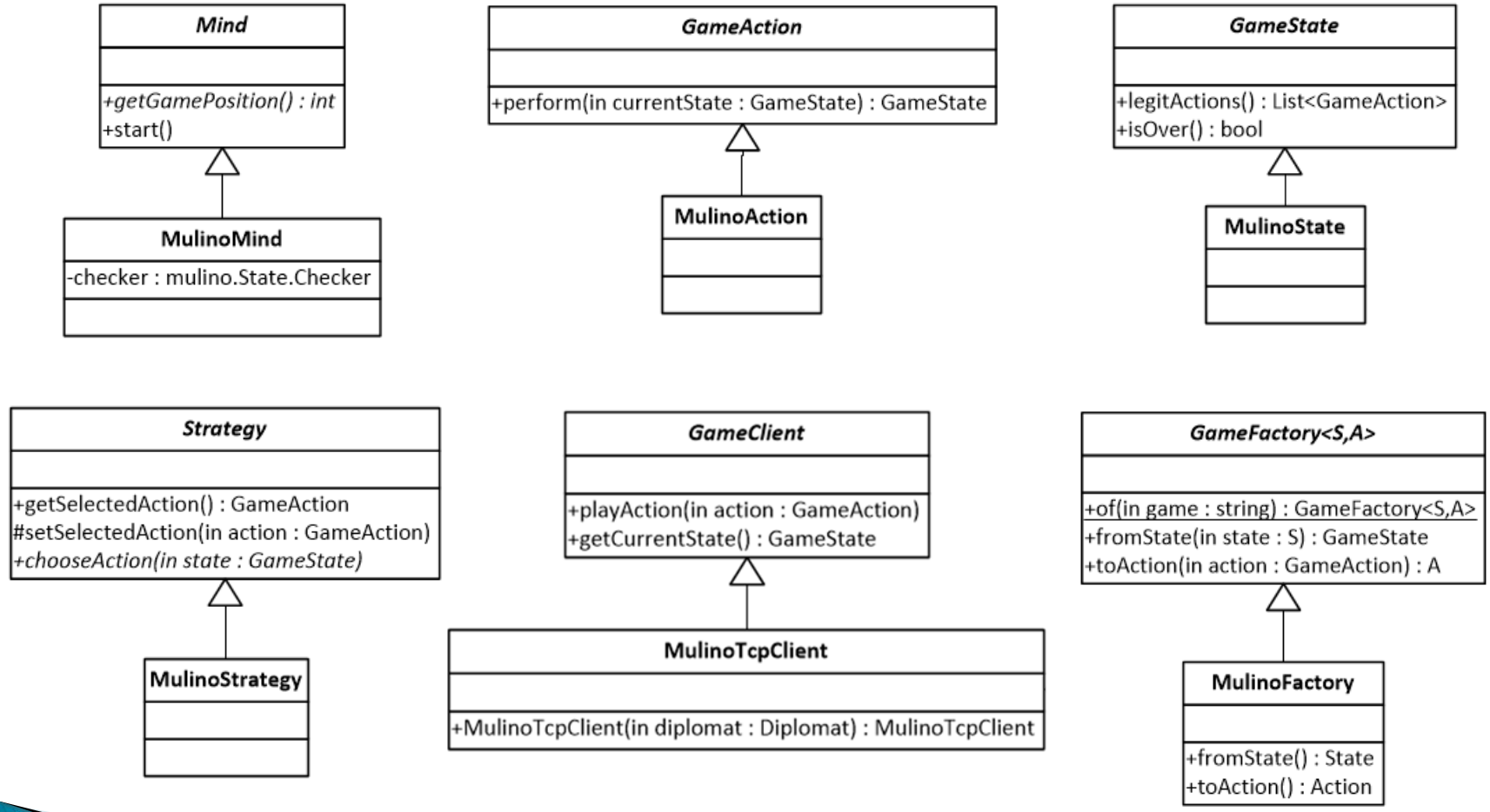


Pensare alle mosse

- Il **ThinkingThread** è il vero e proprio componente che sceglie la mossa tramite la Strategy
- L'idea iniziale era quella di cominciare a pensare già nella fase di *wait* (con il **WaitingThread**) e di passare poi alla fase di *play* la ricerca degli stati già fatta
- I thread sono coordinati con un **Timer** per interrompersi dopo la scadenza del tempo

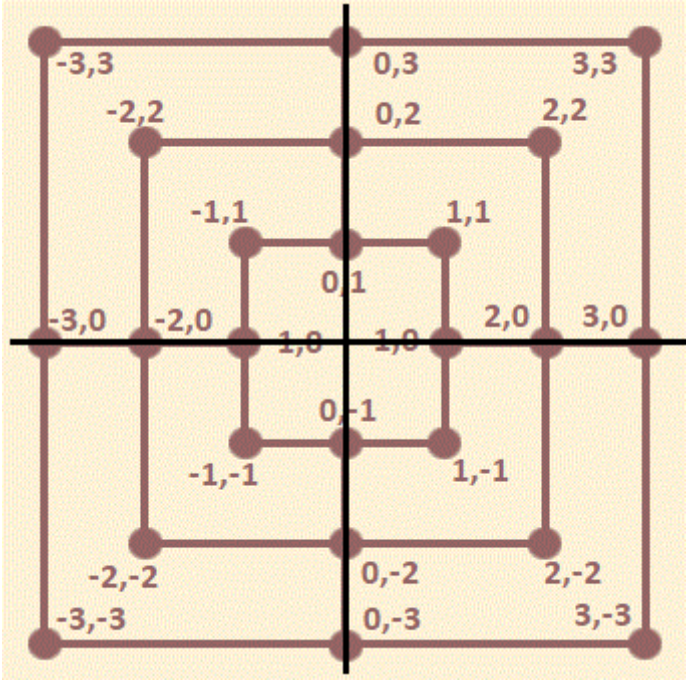


Implementazioni



Rappresentazione dello stato

- Posizioni sul piano cartesiano, board centrata sull'origine (simmetrie)
  - HashMap posizione→pedina in gioco
  - Alternative: due liste/array con le posizioni occupate dai due giocatori
- Contatori per pedine ancora da posizionare e in gioco
- Giocatore di turno
- Fase calcolata al momento
- Azioni ad hoc
- Conversione necessaria per comunicare con il server



Algoritmo di ricerca – 1

- **Iterative deepening Alpha-Beta search**:
  - Basato su AIMA, classe omonima e `AdversarialSearch<S, A>`
  - Usa l'interfaccia `Game<S, A, P>`
- Ordina le azioni da esaminare in base ad una euristica
- Ad ogni interazione:
  - Esplora un livello aggiuntivo
  - Riordina le azioni
- Terminazione se:
  - La mossa migliore è una vittoria/sconfitta
  - La mossa migliore supera di molto la seconda
  - Sono stati valutati solo stati terminali

Algoritmo di ricerca – 2

- Ottimizzazioni:
  - Valori di **alpha** e **beta** propagati anche al primo livello
  - Interruzione se trovata una mossa vincente (taglio appena scopro che un'azione ha "bontà" massima)

```
currDepthLimit = 0;
do {
    currDepthLimit++;
    heuristicEvaluationUsed = false;
    double alpha = Double.NEGATIVE_INFINITY;
    double beta = Double.POSITIVE_INFINITY;

    newResults = new ActionStore<>();
    for (A action : actions) {
        double value = minValue(game.getResult(state, action), player, alpha, beta, 1);
        newResults.add(action, value);

        if(value > alpha) {
            alpha = value;
            if(alpha >= utilMax) {
                break; // winning action found
            }
        }
    }
```

Funzione Euristica – 1

- **h(stato)** = valore miglior mossa + valore stato

- Il valore di una mossa dipende dalla riga e dalla colonna in cui posiziono la pedina:

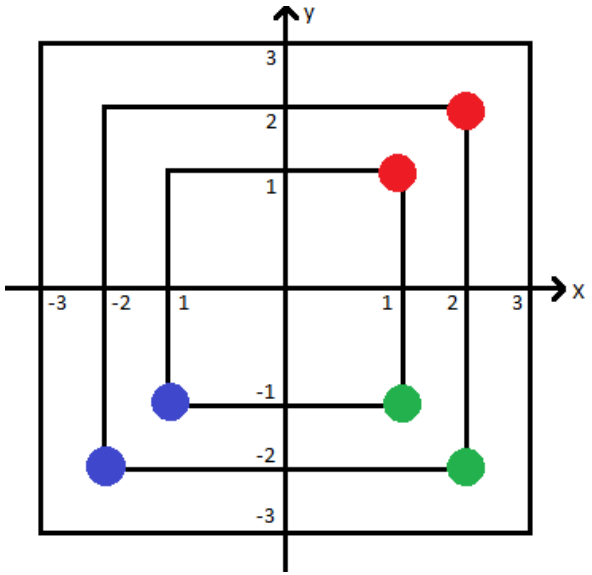
Riga/Colonna	Punti	Situazione
nessuna pedina	1	O—O—O
1 pedina nemica	3	☻—O—O
1 pedina amica	7	☺—O—O
2 pedine nemiche	36	☻—☻—O
2 pedine amiche	56	☺—☺—O
1 pedina amica ed 1 nemica	0	☺—☻—O

Funzione Euristica – 2

- **h(stato)** = valore miglior mossa + valore stato
- Valore stato = Δpedine*M
  - Δpedine dal punto di vista del giocatore di turno
  - M = 100
- Dopo un mulino, Δpedine*M >> valore mossa: gli stati in cui sono in vantaggio di materiale verranno riconosciuti come più vicini alla vittoria e dunque preferiti dall'algoritmo

Punti aperti

- Sfruttamento stato idle
  - Classe **WaitingThread** per «pensare» durante il turno avversario
- Simmetrie
  - Tabellone centrato sugli assi cartesiani
  - **Position** rappresenta delle coppie (x,y)
  - Simmetrie cartesiane e rotazioni



Grazie per l'attenzione

Grazie per l'attenzione