UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Laurea triennale in Informatica
Anno accademico 2021/2022

Fondamenti di Intelligenza Artificiale

Lezione 12 - Ricerca con Avversari (3)

Ricerca con Avversari

Hands-On Minimax

Abbiamo visto la **teoria** dietro la ricerca con gli avversari e l'algoritmo **minimax.**
Ma come si usano **in pratica?**

Ricerca con Avversari

Hands-On Minimax

Abbiamo visto la **teoria** dietro la ricerca con gli avversari e l'algoritmo **minimax.**
Ma come si usano **in pratica?**

Il caso d'uso più semplice per minimax è un **gioco di strategia** a due avversari.

Inizieremo da questo e poi vedremo come **riutilizzare** l'implementazione del nostro agente per un **problema** del **mondo reale.**

Ricerca con Avversari

Hands-On Minimax

Abbiamo visto la **teoria** dietro la ricerca con gli avversari e l'algoritmo **minimax.**
Ma come si usano **in pratica?**

Il caso d'uso più semplice per minimax è un **gioco di strategia** a due avversari.

Inizieremo da questo e poi vedremo come **riutilizzare** l'implementazione del nostro agente per un **problema** del **mondo reale.**

Il gioco che implementeremo sarà:

Ricerca con Avversari

Hands-On Minimax

Abbiamo visto la **teoria** dietro la ricerca con gli avversari e l'algoritmo **minimax.**
Ma come si usano **in pratica?**

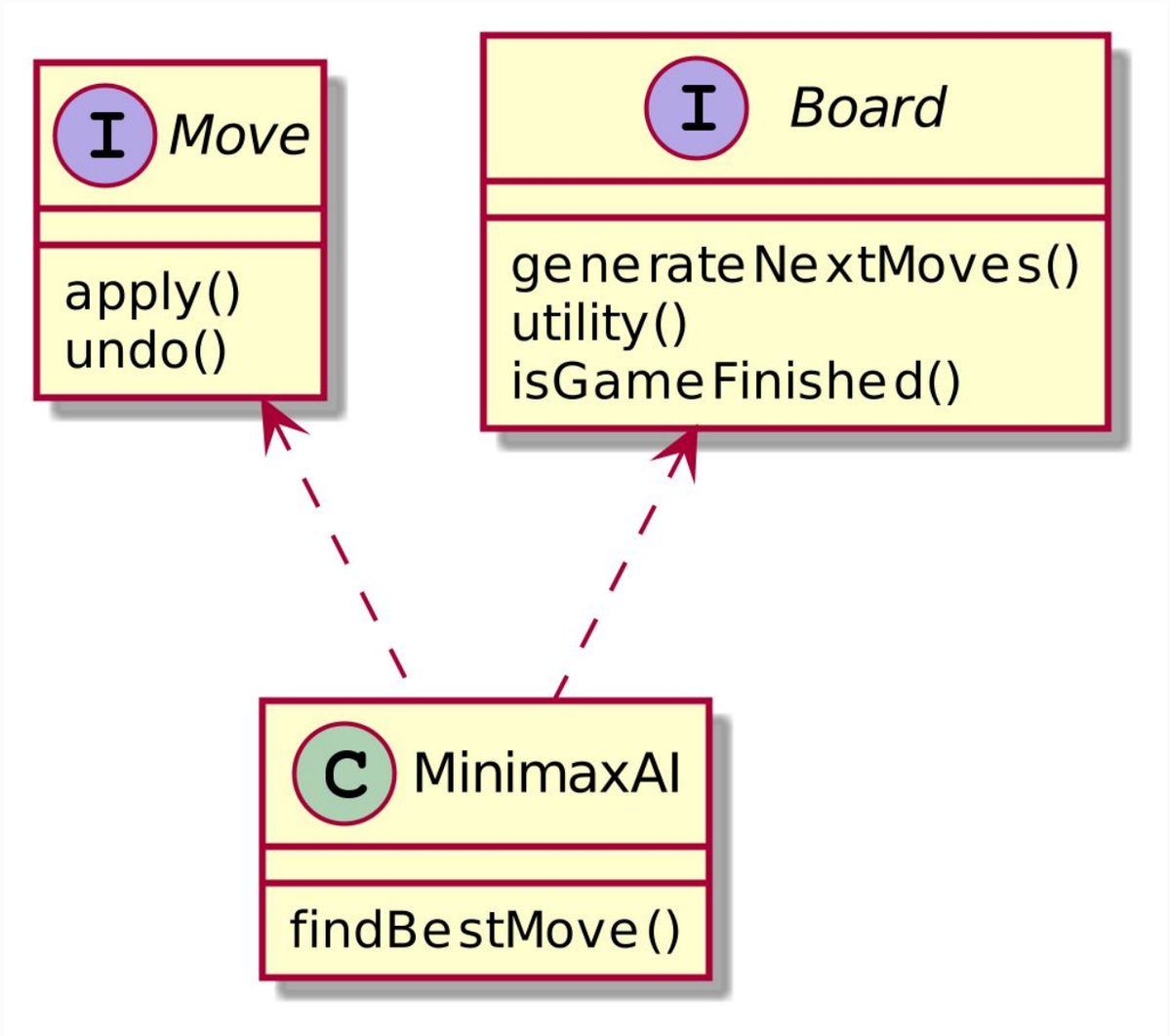
Il caso d'uso più semplice per minimax è un **gioco di strategia** a due avversari.

Inizieremo da questo e poi vedremo come **riutilizzare** l'implementazione del nostro agente per un **problema** del **mondo reale.**

Il gioco che implementeremo sarà: **TRIS**

Ricerca con Avversari

Progettazione della IA basata su Minimax



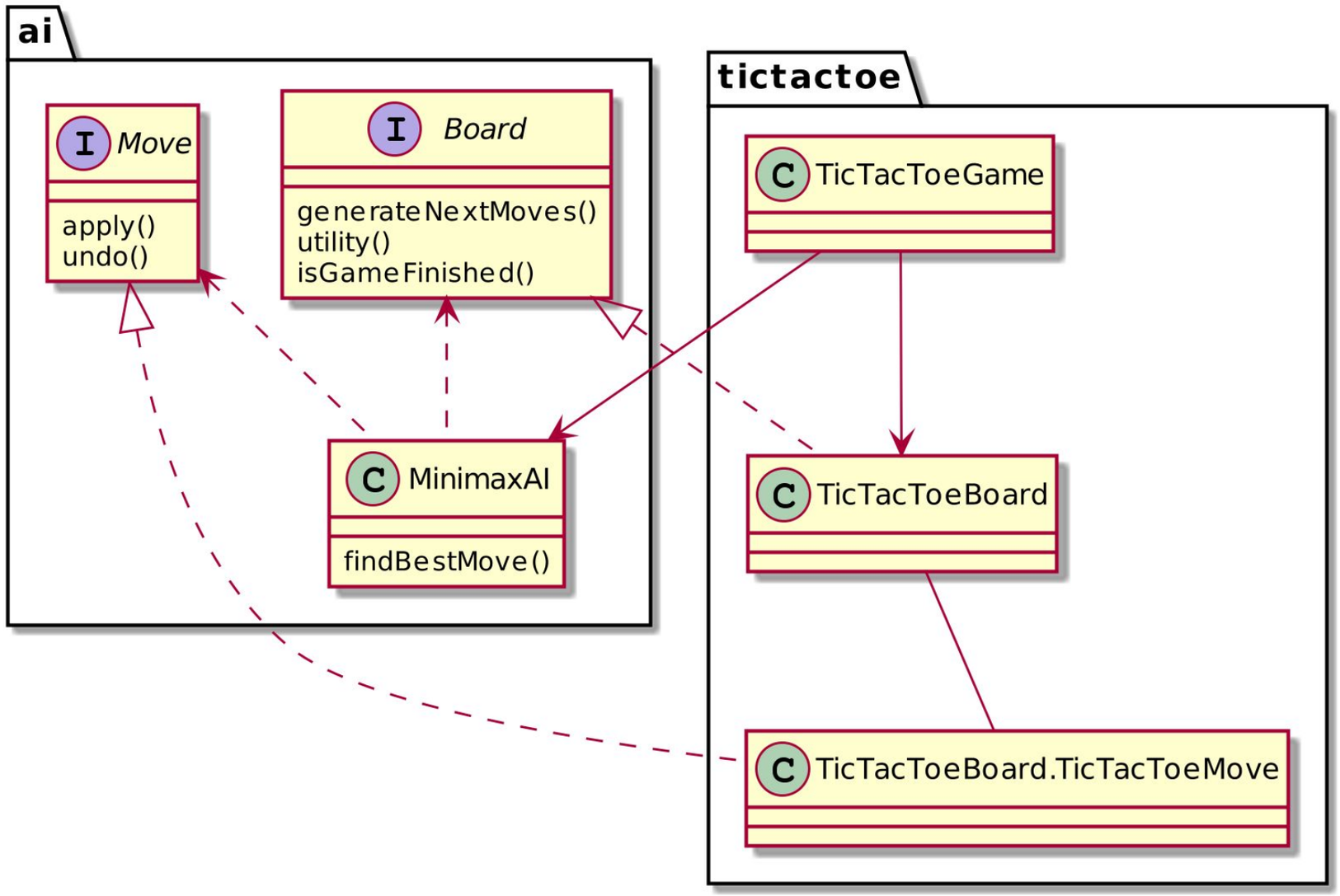
La classe **MinimaxAI** è la classe che conterrà la logica della nostra IA. Implementa un solo metodo (*findBestMove*) che è la nostra implementazione dell'algoritmo minimax.

Questa classe si serve di due interfacce (**Move** e **Board**) che rappresentano, appunto, una generica mossa ed una generica board di gioco.

Implementarli con le interfacce risulta **fondamentale** per il **riuso** del codice!!!

Ricerca con Avversari

Dettagli Implementativi del Gioco

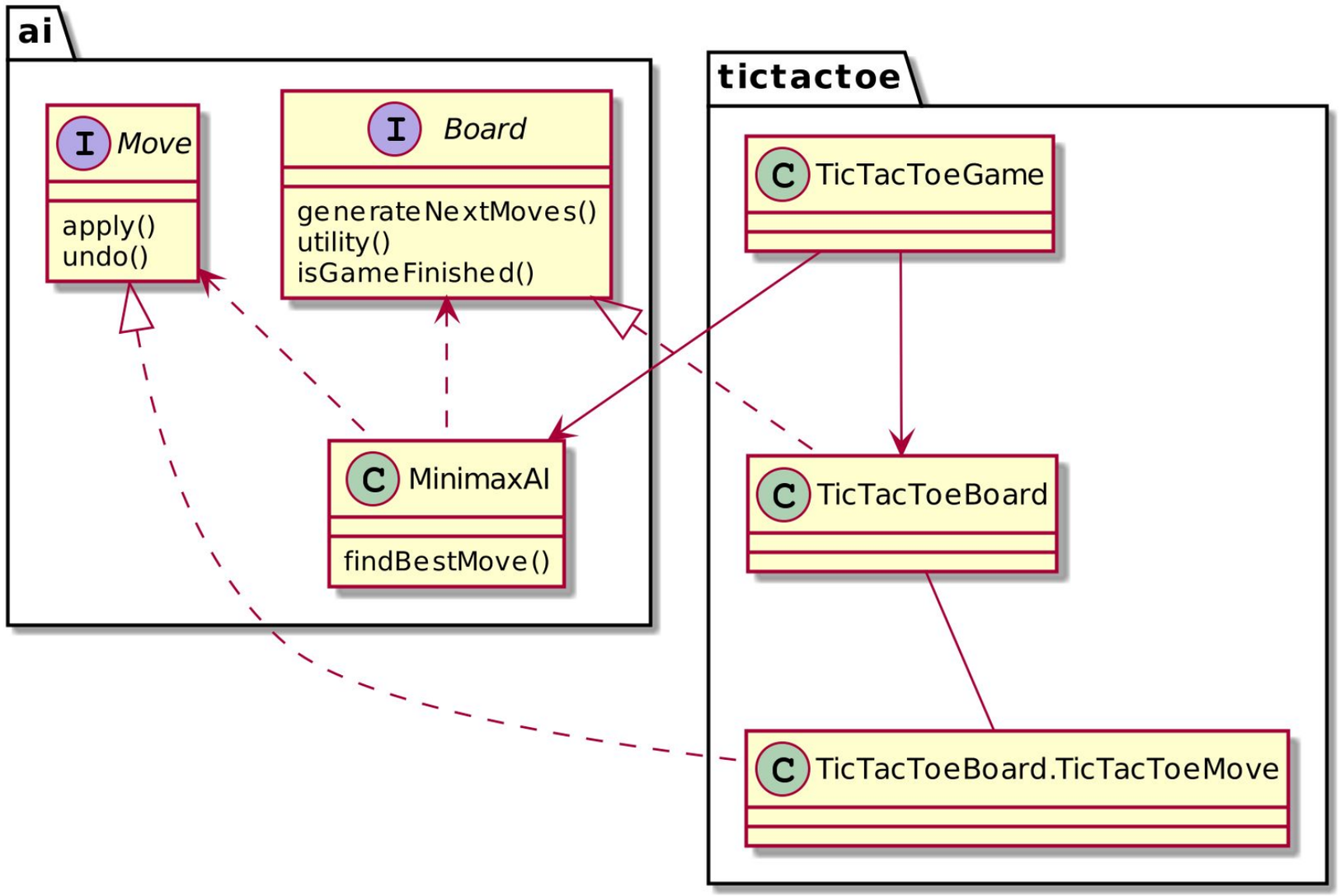


Aggiungiamo le classi specifiche del gioco: **TicTacToeBoard** e **TicTacToeBoard.TicTacToeMove** sono una implementazione concreta delle due interfacce **Board** e **Move**, mentre **TicTacToeGame** gestirà il loop di gioco.

Si ok, ma come si codificano **Move** e **Board**?

Ricerca con Avversari

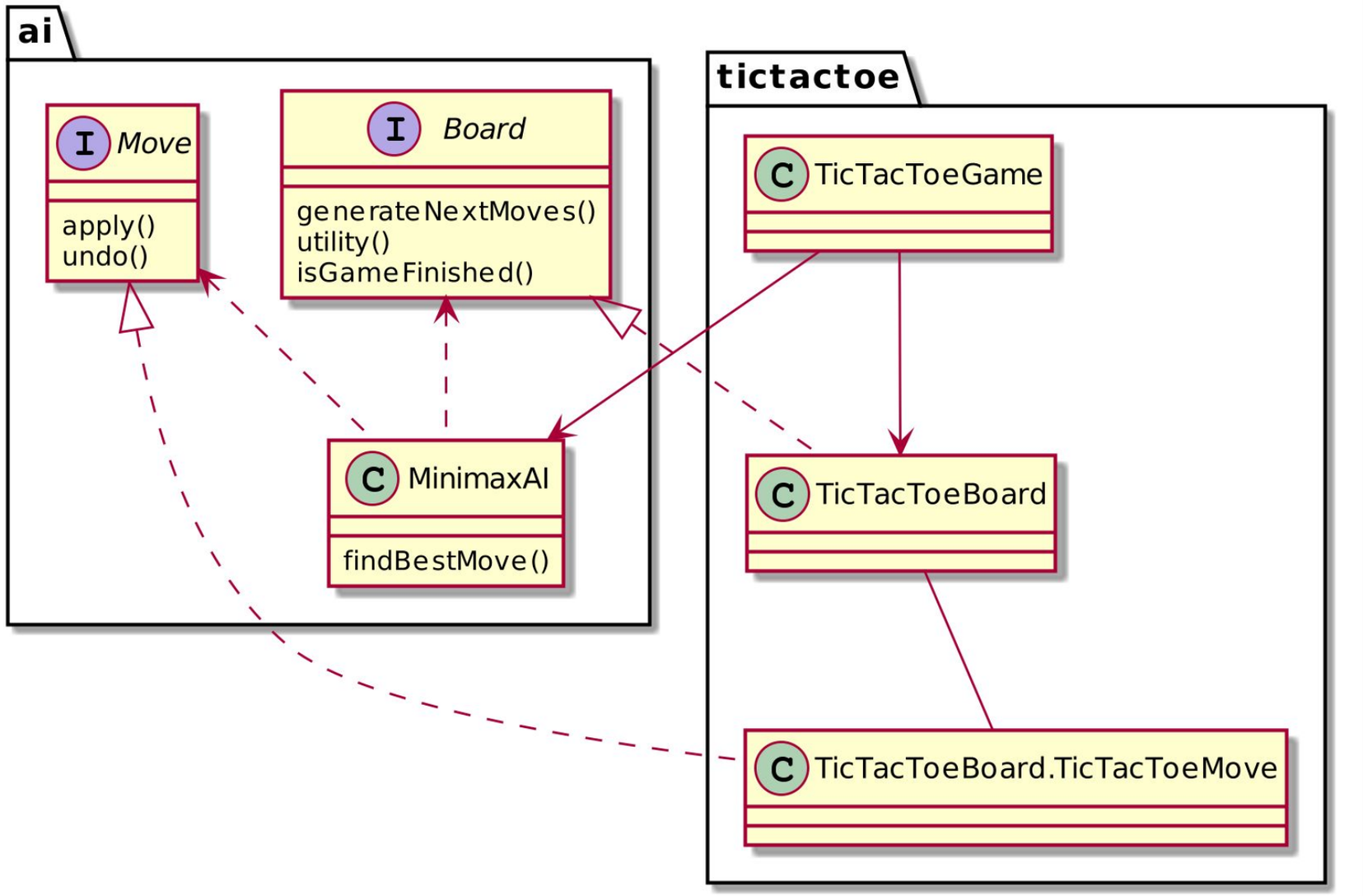
Dettagli Implementativi del Gioco



La **board** mantiene semplicemente una matrice *n*n* al suo interno, che rappresenta la famosa griglia del gioco del tris. La **move** invece, contiene al suo interno una tripla formata da riga, colonna e simbolo da inserire**.**
Quando viene invocato il metdodo *apply()*, viene modificata la **board** inserendo il simbolo nella riga e nella colonna contenuti in **move.** Quando viene invocato il metodo *undo()*, invece, il simbolo presente alle coordinate presenti in **move** viene rimosso**.**

Ricerca con Avversari

Dettagli Implementativi del Gioco



La **board**, dunque, genera le mosse possibili ad ogni turno tenendo conto degli spazi vuoti e del prossimo simbolo da inserire.
Quando la board è piena, il metodo *isGameFinished* restituisce **true** ed è possibile calcolare la *utility()* della partita.

Più Facile a Farsi che a Dirsi

“Talk is cheap, show me the code.”

Ricerca con Avversari

Problemi del Mondo Reale - Team Members Selection

Proviamo adesso ad applicare **minimax** in un contesto diverso da quello di un semplice gioco, ma in una scena di vita reale.
Sarà capitato a tutti, ad esempio durante l'ora di educazione fisica a scuola, di dover formare una squadra, scegliendo a turno con il capitano avversario, suddividendo i giocatori.

Con una corretta codifica del problema, potremo affrontare tale scenario usando **minimax**.

Diamogli una corretta definizione, e poi proviamo a codificarlo.

Ricerca con Avversari

Problemi del Mondo Reale - Team Selection

Problem Statement

Immaginiamo uno scenario che vede due capitani che a turno devono scegliere un membro della propria squadra tra i giocatori possibili.
L'obiettivo è quello di avere al termine della selezione la squadra che è *sulla carta* la favorita per la vittoria.

Ricerca con Avversari

Problemi del Mondo Reale - Team Selection

Problem Statement

Immaginiamo uno scenario che vede due capitani che a turno devono scegliere un membro della propria squadra tra i giocatori possibili.
L'obiettivo è quello di avere al termine della selezione la squadra che è *sulla carta* la favorita per la vittoria.

Com'è noto, non è sufficiente avere solamente i giocatori più forti, ma occorre tener conto anche del gioco di squadra.
Dunque ogni calciatore sarà caratterizzato, per semplicità, da una coppia di valori, uno per le abilità tecniche, ed uno per il gioco di squadra.

Hands-On: Progettare una codifica della board, della mossa, e della funzione di utilità e riutilizzare senza modifiche l'implementazione di minimax.

Ricerca con Avversari

Problemi del Mondo Reale - Team Selection

La Board

Per rappresentare la board di gioco, abbiamo bisogno di tre insiemi. Uno che rappresenti la prima squadra, es A, uno che rappresenti la seconda squadra (B), e l'insieme dei giocatori non ancora scelti (P).

$$A = \{p | p \text{ belongs to team A}\}$$

$$B = \{p | p \text{ belongs to team B}\}$$

$$P = \{p | p \notin A \cup B\}$$

Dove *p* indica un generico calciatore.

Ricerca con Avversari

Problemi del Mondo Reale - Team Selection

La Mossa

Per rappresentare la mossa, ed aderire al design della nostra implementazione, occorre implementare una funzione che sia *annullabile*.
Nel nostro caso, una mossa consiste nell'aggiungere un giocatore non ancora selezionato all'interno di una squadra.

In simboli, definiamo la funzione ***m*** tale che:

$$m(p, S) = S \cup \{p\}$$

La quale è facilmente annullabile perché basta rimuovere l'elemento appena aggiunto.

Problemi del Mondo Reale - Team Selection

La Funzione di Utilità

Per definire correttamente una funzione di utilità, dobbiamo tenere a mente due considerazioni fatte in precedenza:

1. Ogni giocatore è caratterizzato sia dall'abilità tecnica, sia dal gioco di squadra, e dunque occorrono due valori per rappresentarlo.
2. Il nostro obiettivo è quello di avere come risultato la squadra che *sulla carta*, e dunque una *combinazione* delle abilità dei giocatori, sia più forte dell'avversario

La nostra funzione di utilità deve dunque tener conto di due aspetti *complementari* di ogni giocatore, deve rappresentare la forza di una squadra come combinazione di questi, e deve, infine, darci l'informazione di quale squadra sia la più forte.

Come è possibile rappresentare tutto ciò in una sola funzione?

Problemi del Mondo Reale - Team Selection

La Funzione di Utilità

Per poter correttamente rappresentare le caratteristiche di un singolo giocatore, possiamo utilizzare un vettore, che ci dia indicazione della sua abilità tecnica e di gioco di squadra.

$$\vec{p} = (s, t)$$

Dove *s* indica l'abilità tecnica, e *t* il gioco di squadra.

Problemi del Mondo Reale - Team Selection

La Funzione di Utilità

Con questa rappresentazione, possiamo definire la forza *f* di una squadra *T* come la somma vettoriale delle abilità di un singolo giocatore.

$$\vec{f_T} = \sum_{\vec{p} \in T} \vec{p}$$

Problemi del Mondo Reale - Team Selection

La Funzione di Utilità

Infine per calcolare la nostra funzione di utilità occorrerà calcolare la differenza del modulo dei due vettori, che ci dirà quale delle due squadre *sulla carta* è la più favorita.

$$u = \left\|\vec{f}_a\right\| - \left\|\vec{f}_b\right\|$$

UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Laurea triennale in Informatica
Anno accademico 2021/2022

Fondamenti di Intelligenza Artificiale

Lezione 12 - Ricerca con Avversari (3)