# Homework 2

## 1. Descrizione e rappresentazione del gioco

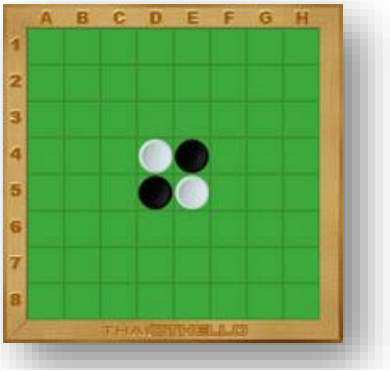


**Figura 1**

Il gioco che si è scelto di rappresentare è **Othello**.
Othello è un gioco da tavolo per due giocatori. Utilizza tre strumenti: l'othelliera (una scacchiera monocolore 8x8), pedine nere e pedine bianche.
Successore di Reversi (invenatato da Lewis Waterman nel 1880) è stato brevettato con il nome di Othello nel 1971 da Goro Hasegawa. I due giochi differiscono solo per la configurazione iniziale dell'othelliera: nel Reversi è completamente vuota, mentre nell'Othello inizialmente vi sono 4 pedine.
La configurazione iniziale di Othello è quella in Figura1: vi sono due pedine nere e due bianche posizionate al centro dell'othelleria in una configurazione ad X.
Le regole del gioco sono molto semplici: si muove alternativamente (inizia il nero) appoggiando una nuova pedina in una casella vuota in modo da imprigionare, tra la pedina che si sta giocando e quelle del proprio colore già presenti sulla scacchiera, una o più pedine avversarie. A questo punto le pedine imprigionate devono essere rovesciate e diventano di proprietà di chi ha eseguito la mossa. È possibile incastrare le pedine in orizzontale, in verticale e in diagonale e, a ogni mossa, si possono girare pedine in una o più direzioni. Sono ammesse solo le mosse con le quali si gira almeno una pedina, se non è possibile farlo si salta il turno. Non è possibile passare il turno se esiste almeno una mossa valida. Quando nessuno dei giocatori ha la possibilità di muovere o quando l'othelliera è piena, si contano le pedine e si assegna la vittoria a chi ne ha il maggior numero sull'othelliera.
Per quanto riguarda la rappresentazione si è scelto di rappresentare la scacchiera con una matrice 8x8. In ogni cella possono esserci tre tipi di valori:

- "-": vuol dire che la cella è vuota
- "k": vuol dire che nella cella c'è una pedina di colore NERO
- "w": vuol dire che nella cella c'è una pedina di colore BIANCO

Nella configurazione iniziale, quindi, la rappresentazione del gioco risulterà:

```
['-' '-' '-' '-' '-' '-' '-' '-']
['-' '-' '-' '-' '-' '-' '-' '-']
['-' '-' '-' '-' '-' '-' '-' '-']
['-' '-' '-' 'k' 'w' '-' '-' '-']
['-' '-' '-' 'w' 'k' '-' '-' '-']
['-' '-' '-' '-' '-' '-' '-' '-']
['-' '-' '-' '-' '-' '-' '-' '-']
['-' '-' '-' '-' '-' '-' '-' '-']
```

# 2. Descrizione del codice

## 2.1 GameModel

Nella cartella **GameModel** vi è il file python contenente la rappresentazione del gioco.
Vengono importate due librerie, **numpy** per la gestione della matrice che rappresenta l'othelliera e **copy** per gestire le copie di oggetti (in particolare degli states).
Viene poi dichiarata quella che è la configurazione iniziale dell'othelliera, da cui si partirà ad ogni giocata e descritta nel paragrafo precedente.
La prima classe che incontriamo è **Game**, dalla quale si può affermare che un oggetto di tipo Game è rappresentato da due valori: uno stato ed un'euristica. Di un Game, dato uno stato ed un turno si può conoscere il vicinato di quello stato, cioè tutti i possibili stati raggiungibili da quello in esame. E' inoltre possibile ottenere lo stato attuale del gioco e verificare se quello stato è una soluzione.
La seconda classe è **OthelloRepresentation**, la quale è la classe che, in particolare, rappresenta il gioco in esame. Ha come attributo board, dove viene salvata una copia della configurazione iniziale dell'othelliera. Permette inoltre, con i suoi metodi, di capire se una certa cella contraddistinta dalle coordinate (a, b) è vuota oppure no (metodo "is_empty"); in quest'ultimo caso ci permette anche di capire di che colore è la pedina che la riempie ("get_disc"). Vi sono poi un metodo che permette di cambiare il colore di una pedina in una cella ("set_disc") e un metodo statico che dato il turno, calcola il colore nemico ("get_enemy_color").
A seguire vi è la classe **OthelloState**, la quale è un modello per gli stati che rappresenteranno le configurazioni di ogni turno. Uno stato è una coppia formata da un valore h e una rappresentazione della scacchiera. Il valore h è il valore euristico associato alla particolare configurazione rappresentata sulla scacchiera. All'interno di questa classe vengono ridefiniti il metodo "equal", "not equal" e "hash" i quali, rispettivamente: verifica se due stati hanno uguale rappresentazione, verifica se non hanno uguale rappresentazione e genera la stringa hash della rappresentazione dello stato. Altri metodi messi a disposizione dalla classe sono il metodo "set_disc" il quale data una coordinata e un colore permette la modifica del disco presente in quella coordinata con il valore passato. Il metodo "is_empty" che verifica se la rappresentazione è vuota e infine, il metodo "is_final", il quale verifica se lo stato in esame è o meno uno stato finale e torna 4 possibili risultati:

- None: non è uno stato finale
- k: il vincitore è il giocatore le cui pedine sono di colore nero
- w: il vincitore è il giocatore le cui pedine sono di colore bianco
- e: si è verificata una situazione di parità tra i possessori delle pedine

Seguono i metodi che verificano le conseguenze di una determinata mossa, dove per mossa si intende voler mettere una pedina in una certa posizione. In un certo senso questi metodi non solo verificano le conseguenze di una mossa, ma rappresentano quelle che sono le regole del gioco.
Il primo metodo che incontriamo è “get_w_discs_change” (get wich discs change), il quale verifica quali dischi sarebbero influenzati dalla mossa m che metterebbe una pedina in posizione (a, b). In tale metodo vengono chiamate quattro funzioni che data la posizione dove si desidera mettere la pedina, verifica se intorno ad essa vi siano già pedine dello stesso colore. Queste funzioni sono:

- ”get_nearest_discs_row”: verifica qual è la pedina dello stesso colore di quella da inserire, più vicina alla posizione che voglio riempire. La ricerca viene effettuata sulla riga.
- “get_nearest_discs_col”: verifica qual è la pedina dello stesso colore di quella da inserire, più vicina alla posizione che voglio riempire. La ricerca viene effettuata sulla colonna.
- “get_nearest_discs_first_diag”: verifica qual è la pedina dello stesso colore di quella da inserire, più vicina alla posizione che voglio riempire. La ricerca viene effettuata sulla diagonale che va dalla coordinata (0,0) alla coordinata (7,7).
- “get_nearest_discs_second_diag”: verifica qual è la pedina dello stesso colore di quella da inserire, più vicina alla posizione che voglio riempire. La ricerca viene effettuata sulla diagonale che va dalla coordinata (0,7) alla coordinata (7,0).

Tutti e quattro questi metodi prendono in input una coordinata che rappresenta la cella in cui si vuole tentare di inserire la pedina e il colore di quest’ultima (bianca o nera). Inoltre, tornano le posizioni delle pedine (se ci sono) dello stesso colore più vicine.
Seguono una serie di if, i quali vanno a riempire le strutture dati che memorizzano quali celle dovranno modificare il loro contenuto. Tali verifiche vengono effettuate sulle coordinate ottenute dai metodi appena discussi, passati come argomenti alle seguenti funzioni:

- “get_affected_row”
- “get_affected_col”
- “get_affected_first_diag”
- “get_affected_second_diag”

Questi quattro metodi, date le coordinate delle pedine di egual colore, verificano se nelle celle comprese tra tali coordinate vi sono tutte pedine nemiche o eventualmente la cella è vuota. Nel primo caso vengono inserite tutte le coordinate delle celle sotto esame in un set che viene poi ritornato in output; nel secondo caso invece il set viene tornato vuoto, in quanto, l’aver trovato una casella vuota vuol dire che la mossa non è valida.

Troviamo infine la classe che rappresenta un'effettiva giocata di Othello: **OthelloGame**. Tale classe a come attributo uno stato del gioco e due metodi:

- "neighbors": Questo metodo prende in input un turno ed un eventuale stato e calcola, a partire dallo stato passato in input o da quello della giocata attuale, tutte le possibili mosse lecite che possono essere effettuate, sia che gli permettano di cambiare configurazione che no. Tale metodo, scorrendosi tutta l'othelliera verifica se in ogni cella che visita è possibile inserire o meno la pedina sfruttando il metodo "get_w_discs_change" (discusso sopra). Quando trova una cella in cui è lecito inserire una pedina simula la mossa e calcola un valore euristico da associare alla configurazione e, quindi, allo stato che si verrebbe a creare facendo quella particolare mossa. Ovviamente in ogni turno potrebbero esservi zero o più mosse possibili, quindi viene sfruttato un set che tenga traccia di tutti i possibili stati a cui si può arrivare partendo da quello in esame (il set non è mai vuoto, in quanto contiene anche lo stato in esame, il quale viene scelto nel caso in cui non vi sono mosse possibili da fare).
- "make_move": Questo metodo prende in output uno stato, una coordinata, il colore di turno e un set di coordinate ed esegue la mossa effettiva con i conseguenti cambi di colori che vanno a modificare lo stato passato in input. Questa è infatti la funzione che viene richiamata dal metodo "neighbors", il quale poi applica l'euristica sul nuovo stato tornato in output da "make_move".

## 2.2 Heuristic

Nella cartella **Heuristic** troviamo il file python dove è possibile visualizzare le euristiche utilizzate e la funzione minMax.
La prima classe principale del file è **OthelloHeuristic**, nella quale sono dichiarate le seguenti funzioni:

- "Avarage_Of_Heuristic": E' il metodo che preso uno stato e l'attuale turno fa una media pesata di tutti i valori euristici che risultano dall'applicazione dell'euristiche sulla configurazione passata. Per quanto riguarda i pesi assegnati, si è scelto di dare peso maggiore alle euristiche che portano a configurazioni favorevoli per i giocatori, cioè quelle laterali, dove, una volta messa la pedina si hanno meno probabilità che essa possa cambiare di colore.
- "Coin_Parity": E' l'implementazione della prima euristica, quella che ha il peso minore. Questa euristica si limita a contare il numero di pedine dello stesso colore di colui che sta giocando rispetto a quelle dell'avversario.

- “Sides_Captured”: è la seconda euristica implementata, quella con peso medio tra tutti e tre. Nell’Othello tra le migliori posizioni da poter raggiungere vi sono le caselle in Figura 2. Queste caselle sono delle posizioni strategiche in quanto una volta conquistata una di esse l’avversario avrà una probabilità molto inferiore di poter cambiare il colore alla pedina. Questa euristica semplicemente, calcola il numero di pedine del giocatore di turno che sono sui bordi, rispetto al numero di pedine dell’avversario.

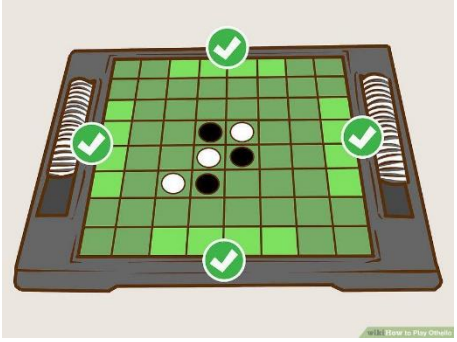

**Figura 2**

- “Corners_captured”: è la terza ed ultima euristica implementata. E’ quella che ha peso maggiore di tutte, in quanto verifica il numero di pedine del giocatore corrente che si trovano agli angoli dell’othelliera, rispetto a quelle dell’avversario. Infatti queste particolari posizioni come in Figura 3, sono le favorite tra tutte, in quanto una volta messa una pedina in un angolo non c’è modo alcuno di cambiare colore a quella pedina.

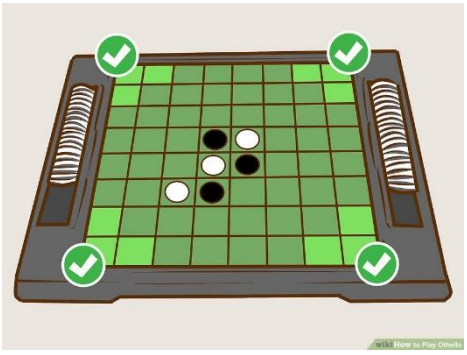

**Figura 3**

- “MinMax”: Questa è la funzione chiave di tutto il codice: preso in input lo stato attuale della giocata, il turno (se bianco o nero) e un livello **l**, calcola ricorsivamente tutte le possibili configurazioni raggiungibili dallo stato attuale per i successivi **l** turni, dove negli **l** turni sono compresi anche quelli dell’avversario. In particolare la funzione prevede le prossime mosse da fare favorendo il giocatore dell’attuale turno e sfavorendo quelle dell’avversario. Per favorire o meno una giocata, considera il valore euristico dello stato relativo alla particolare configurazione che rappresenta la giocata, calcolato con la funzione “Average_Of_Heuristic” precedentemente descritta.

Troviamo a questo punto, una serie di funzioni statiche di supporto alle Euristiche su descritte:

- “count_discs”: effettua il conteggio delle pedine “amiche” e di quelle “nemiche” presenti nell’othelliera.

- “get_enemy_color”: calcola chi è l’avversario rispetto al colore passato in input.
- “count_corner_discs”: effettua il conteggio delle pedine “amiche” e di quelle “nemiche” presenti negli angoli dell’othelliera. Si noti che per individuare tali posizioni si è sfruttato l’utilizzo di un set che contiene tutte le coordinate relative agli angoli.
- “count_side_discs”: effettua il conteggio delle pedine “amiche” e di quelle “nemiche” presenti nelle caselle ai bordi dell’othelliera, esclusi gli angoli. Come per il metodo precedente si è fatto uso di un set per salvare le coordinate che identificassero quelle particolari posizioni.

### 2.3 Main

Il file main è quello che fa effettivamente iniziare una sessione di gioco. Viene data all’utente la possibilità di scegliere il livello di profondità con cui ogni giocatore potrà “prevedere e valutare le mosse future”. Gli viene anche fatto scegliere il nome del file su cui verrà memorizzata l’intera giocata (si è infatti scelto di far memorizzare la giocata in un file, così che l’utente possa poi studiarsi ed analizzare in tutta calma l’andamento della giocata). Si noti che comunque ogni turno sarà ben visibile anche da terminale. Il passo successivo fa iniziare la sessione di gioco partendo dalla configurazione iniziale come descritta sopra. Vengono settate a valori iniziali le variabili di appoggio che conterranno rispettivamente il valore euristico (mx) e lo stato attuale (che nei passi successivi diventerà lo stato precedente, per questo gli viene assegnato il nome prev).
Partendo da un generico stato viene salvato in un set di nome states il neighboors di tale stato (quindi tutti i possibili stati raggiungibili da quello iniziale in una mossa) richiamando, per l’appunto, la funzione neighbors; A questo punto, per ogni stato in states viene calcolato il rispettivo valore euristico con il metodo MinMax, tranne il caso in cui lo stato in states risulti uguale a quello da cui si è partiti: questo caso viene gestito associando allo stato in esame un valore euristico fittizio che sia il più basso possibile (in questo modo si evita che venga scelta in ogni turno la configurazione in cui non viene fatta nessuna mossa). Dopo il calcolo del valore euristico, se questo è maggiore di qualsiasi altro trovato precedentemente allora viene salvato nella variabile mx, mentre il relativo stato viene salvato nella variabile ix. Inoltre, per ogni stato valutato viene fatta la verifica se sia o meno uno stato finale. Nel caso in cui non lo sia, il turno viene passato al relativo avversario, mentre in caso positivo, il gioco termina e viene stampato il vincitore e il tempo impiegato per completare la sessione di gioco.

# 3. Test effettuati

I test effettuati sono memorizzati in file di tipo “.txt” salvati nella cartella output.
All’aumentare del livello, aumentano anche i tempi di durata del gioco.
Per quanto riguarda il tempo impiegato, in media, una sessione di gioco, con livello pari a 3 impiega circa 170 secondi.