PROGETTO FONDAMENTI DI INTELLIGENZA ARTIFICIALE M

Reti neurali applicate ai giochi da tavolo

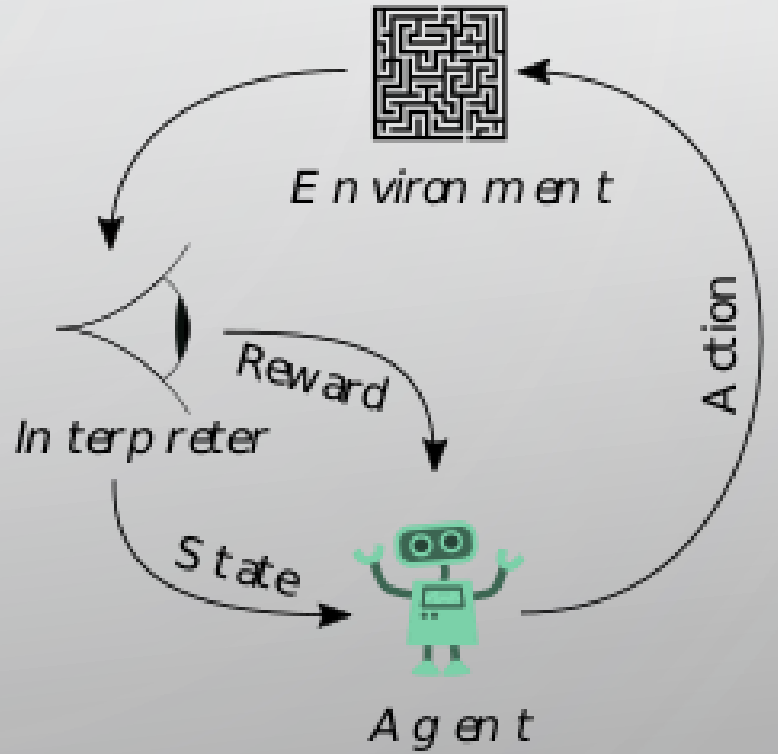
Realizzato da DALMONTE ALAN

OTHELLO

- **Due giocatori** con **conoscenza perfetta**
- **Scacchiera** di **8x8** con 64 caselle
- Una pedina **imprigiona** quelle avversarie in una o più direzioni
- Il giocatore **deve imprigionare almeno un disco avversario**
- **Vince chi**, quando è stata giocata l'ultima mossa, **ha più pedine dell'avversario**

REINFORCEMENT LEARNING

- **Benefici a lungo termine**
- **Q-Learning:**
  - Funzione Q(s,a)
  - Stato s e s'
  - Azione a
  - Ricompensa r
  - Discount factor γ, 0< γ <1
  - Bellman equation



$$Q(s,a) = r(s,a) + \gamma \max_{a} Q(s',a)$$

DOUBLE DEEP Q-NETWORK

- **Rete DDQN:**
  - Rete neurale profonda
  - Approssima funzione Q
  - **Rete destinazione Q~**, congelata nel tempo per un intervallo di aggiornamento

Maggiore stabilità, meno oscillazioni

Capacità di apprendere compiti molto complicati

Rallentamento fase di apprendimento

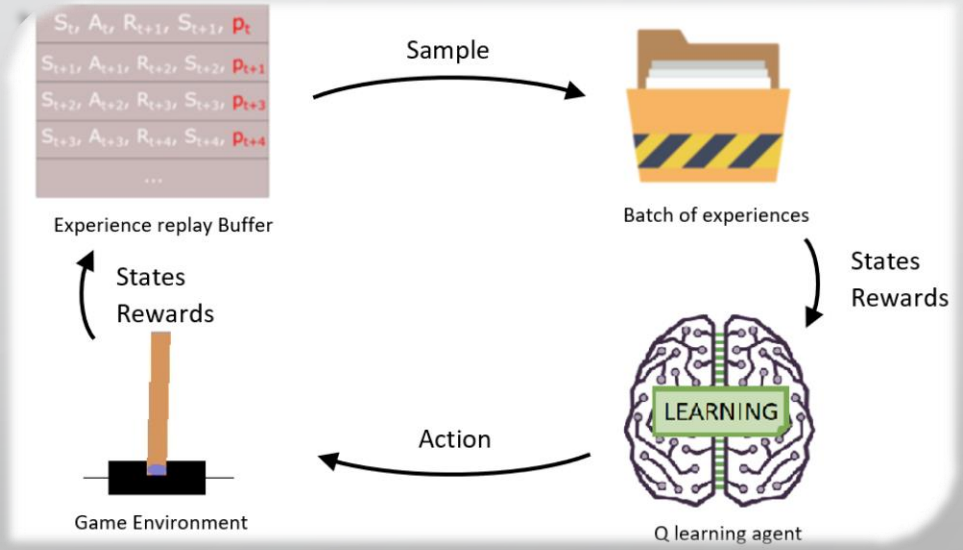
Una funzione viene utilizzata per determinare l'azione di massimizzazione e l'altra in secondo luogo per stimarne il valore

$$Q(s,a) = r(s,a) + \gamma Q\sim(s', \underbrace{argmax_a Q(s',a)}_{\text{DQN Network choose action for next state}})$$

Target network calculates the Q value of taking that action at that state

PRIORITIZED EXPERIENCE REPLAY

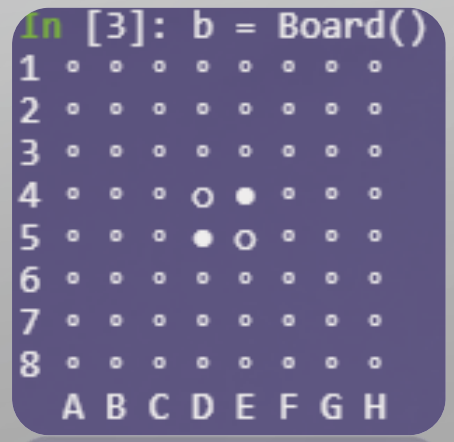
- Rappresentazione della **memoria**
- **L'errore** è trasformato in una **priorità**
- **Probabilità** di essere scelto per una **minibatch** di apprendimento



Minore overfitting **Maggiore efficienza**

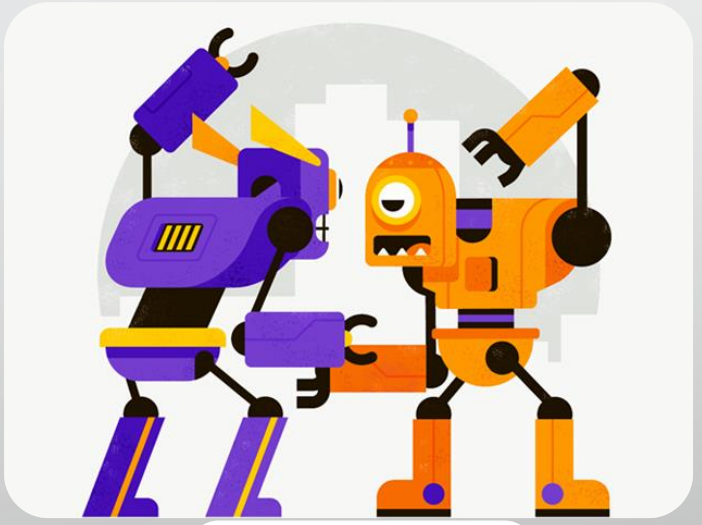
IMPLEMENTAZIONE

- Ambiente di gioco:
  - **Square**, casella della scacchiera
  - **Board**, scacchiera 8x8
  - **AI**, giocatore casuale



IMPLEMENTAZIONE

- **Memory**
- **SumTree**
- **DDQNAgent:**
  - Dense layers Keras
  - Huber loss function
  - Ricompense
  - Apprendimento rete neurale



```
1 ◦ ◦ ◦ ◦
2 ◦ o ● ◦
3 ◦ ● o ◦
4 ◦ ◦ ◦ ◦
  A B C D
Predicted move:
(1, 0)
1 ◦ ◦ ◦ ◦
2 ● ● ● ◦
3 ◦ ● o ◦
4 ◦ ◦ ◦ ◦
  A B C D
1 ◦ ◦ o ◦
2 ● ● o ◦
3 ◦ ● o ◦
4 ◦ ◦ ◦ ◦
  A B C D
```

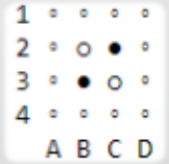
FASE DI APPRENDIMENTO

- **Episodio** di allenamento
- **Inizializzazione della memoria PER**, tramite sfide fra giocatori casuali
- Ogni turno dell'agente:
  - Selezione mossa **epsilon-greedy**
  - **Memorizzazione** della transazione
  - Fase di **apprendimento**

CONCLUSIONE

Creazione di una **piattaforma di gioco semplificata**

SmallBoard, scacchiera 4x4



Minore complessità

Miglior visione dei risultati

Episodio di **apprendimento** di **7000** partite

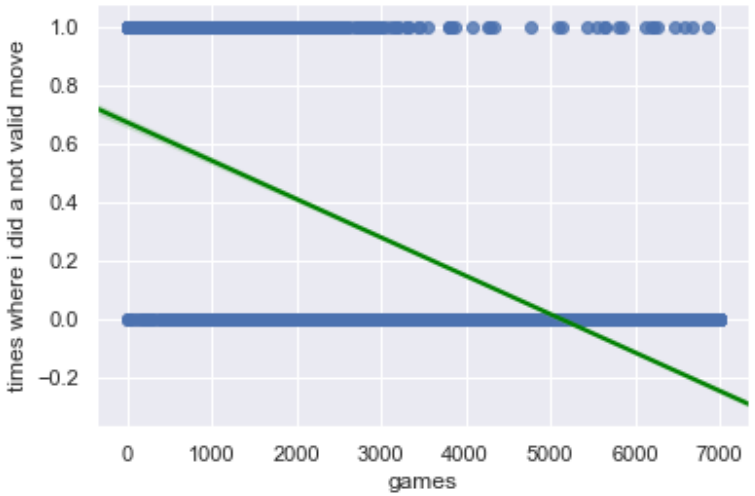


Grafico episodio di apprendimento dello small agent

```
1 ◦ ● ◦ ◦
2 ◦ ● ● ◦
3 ○ ● ○ ◦
4 ◦ ● ◦ ◦
  A B C D

Q-function of trained agent:
7.788171e-12     5.9560186e-16    3.6487373e-09    3.5780331e-10
3.3457832e-07    2.365053e-21     3.6767714e-21    2.3967648e-09
5.2521425e-11    1.5561957e-21    2.2788872e-21    5.2619168e-08
3.0647536e-06    0.99999034       5.3167e-07       5.677239e-06

Q-function of short trained agent:
0.039662585      0.064819194      0.016968185      0.076259896
0.21008962       0.009687764      0.0060026348     0.024160553
0.01781183       0.009611703      0.009583539      0.15350941
0.16079202       0.029436512      0.1444915        0.027113046
```

Confronto Q-function con agente con fase di apprendimento di 100 partite

Documents/universita/seconda | OthelloMain - Jupyter Notebook

localhost:8888/notebooks/Documents/universita/seconda%20magistrale/ProgettoAI/OthelloMain.ipynb

App YouTube Facebook Twitch Netflix Studenti Online - U... Fondamenti di Intel... Orario delle lezioni... Tiscali Mail :: Benve... Qwertee : Limited E... Twitter. Ciò che c'è... Mirco Musolesi's H... Insegnamenti OnLine Computer Graphics

jupyter OthelloMain Last Checkpoint: 17 minuti fa (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

In [10]:

```
#Games against random
def runGames():
    b=SmallBoard()
    agent = DQNSmallAgent()
    opp=AI(0)
    d=b.get_game_over()
    i=0
    counter_games=0
    win_plot=[]
    counter_plot=[]
    counter_wrong=0
    counter_pred=0
    wrong_move=False
    while counter_games < 1000:
        wrong_move=False
        i=0
        b=SmallBoard()
        d=b.get_game_over()
        while d==False:
            if(i==0):
                try:
                    move=agent.makemove(b)
                    print(move)
                    b.coord_move(move)
                    counter_pred+=1
                except:
                    print("Player did a not valid move")
                    #ending manually the loop
                    wrong_move=True
                    counter_wrong+=1
                    counter_pred-=1
                    d=True
                i=1
            else:
                b.coord_move(opp.move(b))
                i=0
            if(d==False):
                d=b.get_game_over()
        if b.get_black_score() >= b.get_white_score():
            if(wrong_move==False):
                win_plot.append(1)
            else:
                win_plot.append(0)
        else:
                win_plot.append(0)
        counter_games+=1
        counter_plot.append(counter_games)
    plot_seaborn_win(counter_plot,win_plot)
```

GRAZIE DELL'ATTENZIONE