Reinforcement learning pentru jocul de sah

Cristian Ivan

1 Introducere

Aceasta lucrare de licenta are ca subiect aplicarea algoritmilor de reinforcement learning pentru diferite variante ale jocului de sah. Obiectivul general al lucrarii este dezvoltarea unui algoritm de reinforcement learning si apoi testarea acestuia cu ajutorul unui engine de sah - Fairy Stockfish, un engine derivat din Stockfish, cel mai cunoscut si performant engine de sah actual. In scopul acestei lucrari de licenta am studiat diferite capitole de retele neuronale, reinforcement learning, alogirtm de cautare minimax cu alpha-beta pruning, cat si alte metode care sunt folosite in mod uzual in inteligenta artificiala pentru sah. Motivatia alegerii acestui subiecte, cat si alte abordari existente vor fi acoperite in subcapitole viitoare.

In afara regulilor standard de sah, vom folosi inca trei variante ale jocului de sah:

1.1 Reguli alternative de sah

Antisah (Antichess)

Aceasta varianta a jocului are urmatoarele reguli:

- Jucătorul care își pierde toate piesele (inclusiv regele) sau rămâne fără mutări legale câștigă partida.
- Capturarea pieselor adversarului este obligatorie. Dacă un jucător poate captura o piesă adversă, este obligat să o facă.
- Dacă există mai multe opțiuni de captură, jucătorul poate alege care piesă să captureze, dar trebuie să captureze.

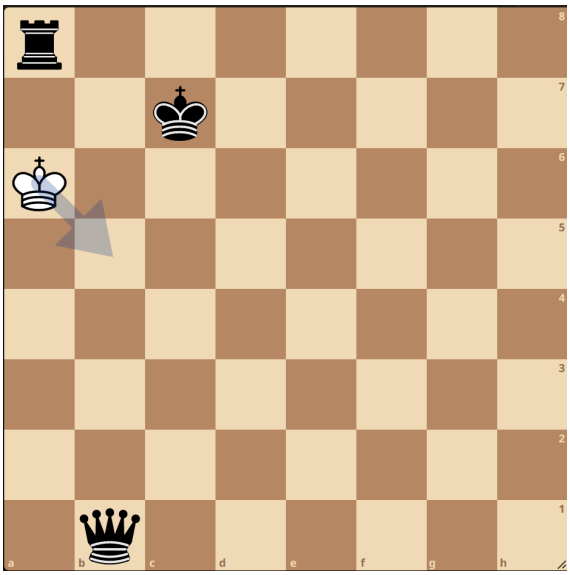

Figure 1: Imagine creata cu ajutorul lichess

In imaginea de mai sus, jucatorul cu piesele albe are o pozitie castigatoare: cand muta regele pe b5, jucatorul cu piesele negre este obligat sa ii captureze regele.
Secventa de mutari castigtaore : 1.Kb5 Qxb5#

King Of The Hill

Aceasta varianta adauga o singura regula noua : un jucător poate câștiga partida prin două metode: fie prin regulile standard de șah (șah-mat, abandon, sau remiză), fie prin aducerea regelui său în centrul tablei: jocul este câștigat imediat dacă regele unui jucător ajunge în unul dintre cele patru pătrate centrale (d4, d5, e4, e5) și nu se află sub atac (nu este în șah).

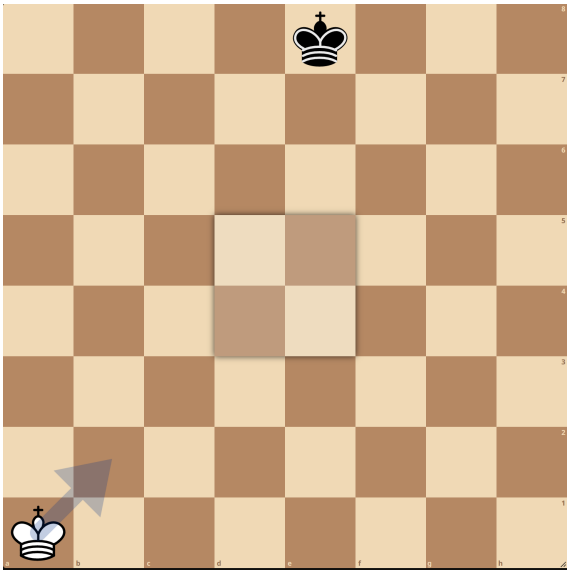

Figure 2: Imagine creata cu ajutorul lichess

In imaginea de mai sus, pentru ca muta primul, jucatorul cu piesele albe castiga, pentru ca regele sau ajunge primul pe unul dintre cele 4 patrate din centru.
Secventa de mutari castigatoare: 1.Kb2 Ke7 2.Kc3 Ke6 3.Kd4#

Three-Check Chess

În 3-Check, jucătorii pot câștiga partida în două moduri:

- Prin regulile tradiționale de șah (șah-mat, abandon, remiză).
- Prin darea de trei șahuri regelui adversarului. Primul jucător care reușește să dea trei șahuri câștigă automat partida.

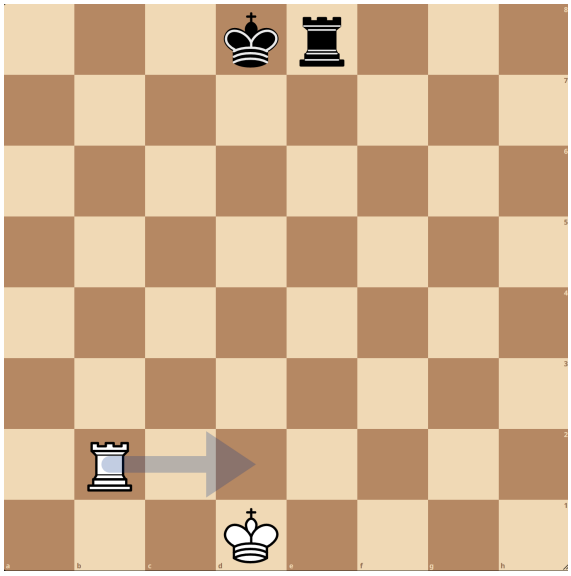

Figure 3: Imagine creata cu ajutorul lichess

In imaginea de mai sus, se observa o diferenta dintre sahul standard si varianta 3-check. Jucand cu regulile standard, aceasta pozitie este o remiza, deoarece niciun jucator nu poate progresa. Insa, in aceasta varianta, jucatorul alb castiga, deoarece jucatorul cu piesele negre nu isi poate apara regele de sah pentru ca tura sa este pe acelasi rand cu regele.

Secventa de mutari castigatoare: 1.Rd2+ Kc7 2.Rc2+ Kb8 3.Rb2# (aici exista mai multe posibilitati).

1.2 Motivatie

Domeniul inteligenței artificiale pentru jocul de sah a înregistrat progrese semnificative de-a lungul anilor, ducând la dezvoltarea unor engine-uri de șah puternice și a unor algoritmi avansați. Folosind metode actuale de a estima abilitatea unui jucator de sah - sistemul ELO - Stockfish sau orice alt engine modern are o probabilitate mai mare de 99% de a bate campionul mondial de sah. In realitate, inteligenta artificiala domina meciurile impotriva jucatorilor umani din anul 1997 (3).

Utilizarea reinforcement learning în șah este un subiect interesant deoarece nu a fost considerată o metodă tradițională în acest domeniu. Pentru o perioada lunga de timp, un engine de sah avea urmatoarea structura: algoritmul de baza era minimax cu alpha-beta pruning, unde, fiindca spatiul de cautare era prea mare, functia de evaluare era scrisa manual si se baza pe diferite euristici. La aceasta structura se adauga o baza de date cu diferite deschideri, care erau bazate pe meciurile celor mai buni jucatori de sah din lume. Aceasta abordare, imbunatatita de-a lungul timpului, a adus rezultate semnificative pentru o perioada destul de lunga de timp, dovada ca cel mai performant engine de sah, Stockfish, a utilizat retele neuronale pentru prima data abia in anul 2020.

Reinforcement learning a aparut pentru prima data in inteligenta artificiala pentru sah in anul 2018. Dintre cele cinci cele mai populare engine-uri de șah, doar AlphaZero și Leela Chess Zero utilizează reinforcement learning, în timp ce Stockfish, Komodo și Houdini se bazează exclusiv pe metode clasice, cum ar fi MCTS (Monte Carlo Tree Search), alpha-beta pruning și funcții de evaluare bazate pe euristici. Merita mentionat ca, dintre toate engine-urile numite, AlphaZero si LeelaChessZero sunt semnificativ mai noi (ambele in 2018), ceea ce sugereaza ca subdomeniul reinforcement learning in inteligenta artificiala pentru sah are potential de viitor.

Majoritatea abordarilor curente - la fel ca toti algoritmii de inteligenta artificiala - se bazeaza pe regulile standard de sah. Cateva abordari existente sunt descrise in capitolul 2, insa aceste abordari nu sunt complete, in cazul CrazyAre, unde unele variante lipsesc cu totul, sau sunt bazate pe cod proprietar, care nu este disponibil public, cum este AlphaZero. Folosirea unor diferite variante de sah ofera mai multe avantaje, cum ar fi:

- Imbunatatirea strategiilor existente: regulile diferite si situatiile care apar datorita acestora pot crea o multime mai diversa de strategii pentru un engine.
- Imbunatatirea unui engine de sah: antrenarea unui engine cu reguli diferite poate fi vazuta ca o metoda de a imbunatati inteligenta artificiala: spre exemplu, unele situatii pot aparea mult mai des intr-o forma de sah decat in alta; aceste situatii diferite pot fi vazute ca un set de date de antrenare mai mare.
- Imbunatatirea abilitatilor jucatorilor umani: de-a lungul timpului, rezultatele obtinute de inteligenta artificiala in sah au reprezentat atat o sursa de inspiratie cat si o modalitate de a-si testa teoriile pentru jucatorii umani. Unele reguli pot pune accentul pe abilitatile unui jucator in partea de inceput a meciului (dechiderea), iar altele se axeaza pe sfarsitul jocului (endgame), deci cunostintle dobandite cu ajutorul inteligentei artificiale va creea jucatori de sah mai buni.

1.3 Abordari anterioare

CrazyAra

Crazy Ara este un engine de șah bazat pe rețele neuronale, conceput special pentru a juca varianta de șah numită Crazyhouse, varianta de sah care se bazeaza pe urmatoarea regula: atunci când un jucător capturează o piesă adversă, acea piesă este transformată într-o piesă de aceeași culoare ca jucătorul care a capturat-o. Motorul utilizează algoritmi de reinforcement learning pentru a evalua pozițiile pe tablă și a prezice cele mai bune mutări. Combină o rețea neuronală antrenată pe un set de date de 500.000 de jocuri jucate de oameni cu un algoritm de căutare Monte Carlo Tree Search (MCTS) pentru a selecta mutările în timpul jocului. Această abordare a fost inspirată de succesul lui AlphaZero de la Google, care utilizează, de asemenea, reinforcement learning și MCTS pentru a obtine rezultate importante. Dezvoltarea Crazy Ara include versiuni scrise atât în Python, cât și în C++, versiunea mai nouă în C++ oferind o performanță îmbunătățită. Este compatibil cu diverse interfețe GUI precum CuteChess și WinBoard pentru o utilizare convenabilă. Stilul de joc al lui Crazy Ara este adesea descris ca fiind mai intuitiv și dinamic în comparație cu motoarele tradiționale de șah, care se bazează foarte mult pe algoritmi de căutare extensivi. Crazy Ara tinde să prioritizeze pozițiile mai bune pe tablă în detrimentul valorii imediate a pieselor individuale, ceea ce duce la un joc mai agresiv și orientat spre inițiativă.

In acest articol (4) din 2020, autorii compara performanta CrazyAra cu alte engine-uri de sah ce pot juca varianta Crazyhouse. In toate meciurile jucate, CrazyAra a obtinut rezultate pozitive impotriva tuturor engine-urilor in afara de Stockfish.

Un dezavantaj semnificativ adus de CrazyAra este faptul ca se limiteaza doar la varianta Crazyhouse, deci nu poate juca multe alte reguli populare cum ar fi : Chess960 (Fisher Random Chess, o varianta unde jucatorii incep cu o pozitie aleatoare a pieselor, King of the Hill, Atomic Chess etc.

AlphaZero

AlphaZero este o inteligență artificială creată de DeepMind, capabilă să învețe și să joace la nivel inalt jocuri complexe precum șah, shogi și Go. AlphaZero a reprezentat prima dezvoltare serioasa a unor algoritmi de reinforcement learning pentru inteligenta artificiala in jocuri. Foloseste o combinatie de retele neuronale, MCTS si reinforcement learning:

- rețele neuronale profunde pentru a evalua pozițiile de pe tablă și a selecta mișcările. Rețeaua neuronală ia starea tabloului ca intrare și produce două informații cheie: politică- o distribuție a probabilităților asupra tuturor mișcărilor posibile, indicând probabilitatea ca fiecare mișcare să fie cea mai bună si valoare- o valoare scalară care estimează probabilitatea de a câștiga din poziția curentă.

- reinforcement learning: AlphaZero joacă un număr mare de jocuri împotriva sa, generând date despre pozițiile jocului, mișcări și rezultate. Datele generate din jocurile cu sine insusi sunt stocate într-un buffer, rețeaua neuronală fiind antrenată pe aceste date, învățând să prezică politica și valoarea.
- MCTS: Combină predicțiile rețelei neuronale cu explorarea bazată pe căutare. Începând de la nodul rădăcină (starea curentă a tabloului), algoritmul selectează noduri copil bazate pe un echilibru între exploatare (folosind politica rețelei neuronale) și explorare (încercând noduri mai puțin vizitate). Când se ajunge la un nod frunză (un nod nevizitat anterior), acesta este extins prin adăugarea unui nou nod corespunzător unei mișcări posibile. Rețeaua neuronală evaluează noul nod, furnizând o estimare a politicii și a valorii. Estimarea valorii este propagată înapoi în arbore, actualizând statisticile pentru fiecare nod de-a lungul traseului.

In aceasta lucrare (13) din 2020, AlphaZero este antrenat pe 9 variante de sah diferite pentru a trage concluzii despre efectul regulilor asupra :

- tendintei unui meci de sah de a ajunge la remiza
- avantajul pe care il confera prima mutare
- diversitatea deschiderilor
- cat de decisive sunt meciurile (in acest caz, un rezultat decisiv inseamna un meci fara remiza)
- dinamica jocului (numarul de mutari si cat de "incet" este un meci)
- utilizarea mutarilor speciale introduse de acea varianta.

Aceste rezultate, cat si alte rezultate obtinute cand AlphaZero a fost antrenat pe regulile standard de sah, demonstreaza ca AlphaZero este cea mai promitatoare abordare de reinforcement learning pentru sah. Cand vine vorba de performanta, AlphaZero a fost testat impotriva altor engine-uri de sah in mod direct doar cu reguli standard, unde a obtinut rezultate bune, avand un scor ELO asemanator cu Stockfish.

Singurul dezavantaj pe care il aduce AlphaZero este faptul ca nu este open-source. Spre deosebire de Stockfish, rezultatele unui engine nu pot fi comparate direct cu AlphaZero. Bazat pe o putere de calcul mare, el este antrenat doar intern de catre DeepMind, compania care l-a creat. Spre exemplu, algoritmul dezvoltat in aceasta lucrare de licenta, nu poate fi comparat cu AlphaZero. Totusi, merita mentionat ca DeepMind a publicat diferite articole in care detaliaza modul in care functioneaza AlphaZero si algoritmii care stau in spatele inteligentei artificiale, astfel ca pot exista mai multe implementari care seamana cu AlphaZero.

Fairy Stockfish

Fairy Stockfish este o extensie a engine-ului de șah Stockfish, proiectată pentru a suporta o gamă largă de variante de șah. Aceasta se bazează pe arhitectura robustă și optimizată a Stockfish, dar include funcții suplimentare pentru a gestiona regulile unice ale variantelor diferite de sah.

Fairy Stockfish moștenește arhitectura de bază a Stockfish, care include:

- Alpha-Beta Pruning: Un algoritm de căutare optimizat care evaluează mișcările posibile prin eliminarea ramurilor care nu influențează decizia finală.
- Reprezentarea Bitboard: O reprezentare eficientă a tablei de șah și a pieselor, permițând calculuri și manipulări rapide.
- Funcția de Evaluare: O funcție sofisticată care evaluează valoarea unei poziții pe tablă, ținând cont de echilibrul material, activitatea pieselor, siguranța regelui și alți factori.

Fairy Stockfish introduce mai multe modificări și extensii: include un cadru flexibil pentru definirea regulilor diferitelor variante de șah. Acesta îi permite să ajusteze modelele de mișcare ale pieselor, condițiile de câștig și alte reguli specifice jocului. Definițiile Pieselor: Fairy Stockfish poate defini

noi tipuri de piese și mișcările lor. De exemplu, piesele din variante precum Crazyhouse sau Atomic Chess au reguli de mișcare și interacțiune unice. De asemenea, funcția de evaluare din Fairy Stockfish este ajustată pentru a ține cont de aspectele unice ale diferitelor variante de șah: funcția de evaluare include euristici adaptate caracteristicilor specifice fiecărei variante. De exemplu, în Crazyhouse, se ia în considerare valoarea pieselor aflate în mână pentru reintroducere. Se fac ajustări pentru a evalua corect mobilitatea și influența pieselor specifice variantelor. De asemenea, Fairy Stockfish evaluează controlul asupra zonelor critice ale tablei, care poate varia semnificativ între variante.

Fiind derivat din Stockfish, Fairy Stockfish este cel mai performant engine pentru variante non-standard de sah. Deoarece este performant si suporta un numar mare de seturi de reguli, este de obicei folosit pentru a testa alte engine-uri si algoritmi (va fi folosit pentru testare si in lucrarea de fata). In schimb, Fairy Stockfish nu foloseste reinforcement learning. Acest lucru nu este neaparat un dezavantaj, insa nu ne poate oferi informatii in plus despre cum poate fi folosit reinforcement learning pentru jocul de sah.

Leela Chess Zero (Lc0)

Leela Zero Chess este un proiect open-source care vizeaza regulile standard de sah; insa arhitectura sa flexibila ii permite sa fie folosit si pentru alte seturi de reugli. Lc0 foloseste reinforcement learning, anume o combinatie de retele neuronale si MCTS, metoda fiind inspirata de cea folosita de AlphaZero. Desi nu a fost folosit in mod oficial in acest sens, Lc0 poate fi reantrenat pe seturi de reguli diferite pentru a putea juca Antichess, Crazyhouse sau alte variante.

2 Descrierea solutiei

Algoritmul dezvoltat se va baza pe Deep Q-Learning, deci vom folosi retele neuronale adanci pentru a aproxima functia Q-valoare.

2.1 Tehnologii folosite

1. **TensorFlow**

   Pentru a crea reteaua neuronala adanca care sta la baza algoritmului de Deep Q-learning, vom folosi TensorFlow. Pentru a crea reteaua neuronala, putem alege mai multe moduri de a reprezenta tabla de sah, acesta de mai jos este doar unul dintre ele.

```
def create_chess_model():
    model = Sequential()
    model.add(Conv2D(64, (3, 3), activation='relu', input_shape=(8, 8, 12)))
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(Conv2D(256, (3, 3), activation='relu'))
    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['
    accuracy'])
    return model
")
```

Listing 1: Exemplu utilizare TensorFlow

   In acest exemplu, am folosit un tensor de forma (8,8,12), unde tabla este reprezentata ca o matrice patratica de dimensiune si 8 cu 12 nivele, unde fiecare nivel reprezinta prezenta unei piese.

2. **Gym, Stable Baseline 3**
   Gym este o librarie ce apartine ecosistemului OpenAI des folosita pentru dezvoltarea algoritmilor de reinforcement learning. Este folosita atat pentru ca este simplu de utlizat cat si pentru ca are un numar mare de "medii" (environment) ce pot fi folosite. In acest caz, putem folosi environmentul gym-chess.

   Stable Baseline 3 este o librarie care implementeaza algoritmi de reinforcement learning in Python. Este un fork al librariei Stable Baseline si este scris utilizand libraria PyTorch. Contine mai multi algoritmi de baza de reinforcement learning, cum ar fi: Deep Q-Learning (DQN), Advantage Actor-Critic (A2C), Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC) etc. Pentru simplitate, in exemplul de mai jos am folosit algoritmul DQN din libraria SB3. Acest lucru nu este obligatoriu; acest algoritm poate fi si implementat manual.

```

env = gym.make('Chess-v0')


env = make_vec_env(lambda: env, n_envs=1)


class CustomCNNPolicy(DQN):
    def __init__(self, *args, **kwargs):
        super(CustomCNNPolicy, self).__init__(*args, **kwargs)
        self.q_net = create_chess_model()

    def forward(self, obs):
        return self.q_net(obs)


model = DQN(CustomCNNPolicy, env, verbose=1)


```

```
checkpoint_callback = CheckpointCallback(save_freq=10000, save_path='./checkpoints/', name_prefix='dqn_chess')
model.learn(total_timesteps=1000000, callback=checkpoint_callback)


model.save("dqn_chess_model")

#Testare
obs = env.reset()
for i in range(1000):
    action, _states = model.predict(obs, deterministic=True)
    obs, rewards, dones, info = env.step(action)
    env.render()
    if dones:
        obs = env.reset()
```

Listing 2: Exemplu utilizare Gym

3. Fairy Stockfish

Dupa cum am descris in mod anterior, acesta este cel mai performant engine pentru variante non standard de sah. Il vom folosi pentru a testa performanta algoritmului dezvoltat. Presupunand ca avem algoritmul dezvoltat, vom testa performanta acestuia prin jocuri impotriva lui Fairy Stockfish, cu un script care arata asa:

```
def test_algorithm_vs_stockfish(stockfish_path, algorithm, num_games=100):
    env = AntichessEnv(stockfish_path)
    results = {"wins": 0, "losses": 0, "draws": 0}

    for game in range(num_games):
        state = env.reset()
        while not env.board.is_game_over():
            algorithm_move = algorithm.get_move(state)
            if algorithm_move not in [move.uci() for move in env.board.legal_moves]:
                raise ValueError(f"Invalid move {algorithm_move} by the algorithm.")

            state, done = env.step(algorithm_move)
            if done:
                break


            state, done = env.step(env._parse_best_move(env._get_response()))
            if done:
                break

        if env.board.is_checkmate():
            if env.board.turn:
                results["wins"] += 1
            else:
                results["losses"] += 1
        else:
            results["draws"] += 1

    env.close()
    return results
```

Listing 3: Exemplu Script Testare

4. **Python Chess, pygame**

   Pentru a dezvolta interfata grafica, am folosit librariile Python Chess si pygame. Python chess ofera suport pentru toate cele 3 variante de sah pe care le vom testa, cat si suport pentru diferite functionalitati necesare pentru interfata grafica. Spre exemplu, verificarea corectitudinii mutarilor, verificarea daca jocul s-a incheiat sau nu, lista cu toate mutarile facute etc. nu au mai trebuit implementate manual.

   Pygame este una dintre cele mai folosite librarii pentru implementarea unei interfete grafice si este potrivit si in acest caz, pentru ca ofera metode simple de a implementa functionalitatile necesare.

2.2 Alte resurse

In aceasta sectiune, vom mentiona implementari asemanatoare sau care pot oferi informatii despre cum ar trebui strucurat algoritmul, sau cum putem sa imbunatatim performanta algoritmului.

1. Implementare bazata pe AlphaZero

   Aceasta este o implementare de algoritmi de reinforcement learning care se bazeaza pe arhitectura AlphaZero. Dupa cum descrie autorul, algoritmul face o cautare MCTS modificata, unde selectia nu este aleatorie ci bazata pe reteaua neuroanala, iar simularea este inlocuita de valoarea primita de reteaua neuronala

2. Implementare bazata pe Q-Learning

   Aceasta este o implementare bazate pe Deep Q-Learning. Este un algoritm fara model, deci nu are o modalitate de a vedea mutarile inainte. Din acest motiv, nu este o implementare la fel de performanta

3 Bibliografie

1. Sistemul de rating ELO
2. Fairy Stockfish
3. Meciul dintre Garry Kasparov si Deep Blue
4. Chess Variants with Neural Networks
5. Variante de sah cu reinforcement learning
6. Gym
7. Stable Baseline 3
8. Python Chess
9. CrazyAra
10. CrazyAra, github
11. AlpahZero description
12. AlphaZero Variants Article
13. AlphaZero Variants Source
14. AlphaZero Based implementation
15. Antichess rules
16. Three-check rules
17. King of the hill rules
18. Crazyhouse rules
19. Editor pentru crearea pozitiilor
20. Houdini Chess Engine
21. Stockfish Chess Engine
22. Komodo Chess Engine
23. Leela Chess Zero Chess Engine
24. Monte Carlo Tree Search
25. Articol despre AlphaZero si Reinforcement Learning
26. AlphaZero si MCTS
27. Tutorial implementare AlphaZero
28. AlphaZero explicat