Halite Project

- **Instrucțiuni de compilare:**

python3 TheEmpire.py

- **Detalii despre structura proiectului:**

Proiectul contine doar fisierul TheEmpire unde este toata implementarea.

- **Detalii despre abordarea algoritmică: ce algoritmi ați folosit, cum i-ați combinat, de ce, complexități, etc:**

Am mers pe o abordare de tip matematica, pe o euristica creata in urma unor discutii, grafice, teste, astfel:

- Trecem prin fiecare patratel detinut de Bot-ul nostru si calculam pentru fiecare directie posibila (Nord, Sud, Est, Vest sau Diagonala) o valoare pe care am denumit-o "oportunitate".
- Oportunitatea reprezinta o valoare intre 0 si 1 care marcheaza cat de "util" ar fi sa mergem intr-o anumita directie.
- Utilitatea este data de mai multi factori pe care i-am considerat importanti:
  - un factor care este calculat in functie de cati boti inamici se afla la momentul respectiv pe harta de joc (cand sunt mai multi inamici pe harta, nu ne dorim sa ne luptam direct cu toti, ci sa ii lasam sa se lupte intre ei pentru a irosi resurse).
  Factorul este dat de functia get_opportunity_factor
  - un alt factor este dat de dificultatea de a captura un patratel, raportat la cat de mult produce acel patratel. Pentru calculul lui, am folosit functia max pentru a evita ca acesta sa fie nul
  - ultimul factor este invers proportional cu anteriorul, folosim functia e^{-x} pentru a amortiza efectele produse de rezultatul anterior si a mentine factorii calculati intr-un anumit range pentru a mentine caracterul determinist. Daca nu am aplica o functie de activare, multe valori s-ar strange in jurul unui interval, iar la o mica schimbare, am putea obtine outliers care ar determina un comportament nedorit
- Scopul normalizarii este acela de a aduce toate valorile in intervalul [0,1] si de a introduce un caracter determinist proiectului.
- Folosim un **algoritm Greedy**, sortand descrescator miscarile cumulate dupa strength, apoi incercam sa pastram doar cele mai bune miscari (care consuma cel mai putin strength). Astfel, se evita overkill-ul.

- Procesul de alegere a miscarilor optime se face in mai multi pasi:

Initial, se stabileste cea mai buna directie in care sa se faca extinderea, acest lucru realizandu-se astfel: directia este aleasa pe baza oportunitatilor pe care le va avea in viitor.

Oportunitatea totala e definita ca fiind suma oportunitatilor pentru fiecare square (cu cat un target este mai departe, cu atat ponderea lui este mai mica, deoarece lucrurile se vor schimba pe viitor, asa ca este incert daca va mai reprezenta un punct de opotunitate). Pentru a acomoda acest aspect, este folosit un factor de degradare, care consta intr-o constanta exponentiala.

Alte strategii incluse in proiect, sunt:

- Daca un bot inamic ar face overkill pe un square disputat, atunci vom face respectiva miscare chiar daca este si in dezavantajul nostru, deoarece, pe termen lung, ne va avantaja.
- Pentru square-urile care se afla in teritoriul nostru, preferam sa le pastram pentru productie, decat sa le miscam.

- Pentru fiecare frame al jocului, avem o complexitate **$O(n * m^2)$,** unde n = numarul de linii si m = numarul de coloane.

  Descriere:

  O(n * m) = calculul celei mai bune miscari pentru fiecare square din harta

  O(m/2) = dimensiunea maxima pana la care cautam cea mai buna mutare

- ## Surse de inspirație:

  1. **https://2016.halite.io/basics_improve_random.html**
  2. https://github.com/Sydriax/HaliteBot/blob/master/MyBot.py
  3. https://github.com/GaudyZircon/infestron
  4. https://github.com/kindanoob/halite

- ## Responsabilitatea fiecărui membru al echipei:

Taiatu Iulian - responsabil cu idei matematice, calcul exponential, normalizarea factorilor, idei extrase din Machine Learning.

Mihailescu Eduard, Dinu Sabina - implementare + documentatie, comentarii cod