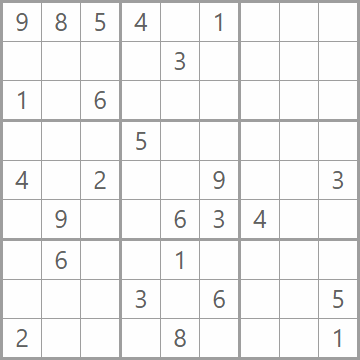
BUDOWA:

Plansza:

Przykład:



Kolumna:

Przykład: Kolumna 1



Wiersz:

Przykład: Wiersz 1



Sektor:

Przykład: Sektor 1/A

Obliczanie Sektora : (Int wiersz / 3) \* 3 +(Int kolumna /3)

Obraz zawierający zrzut ekranu, kwadrat, linia, Prostokąt

Opis wygenerowany automatycznie

Obraz zawierający tekst, numer, zrzut ekranu, kwadrat

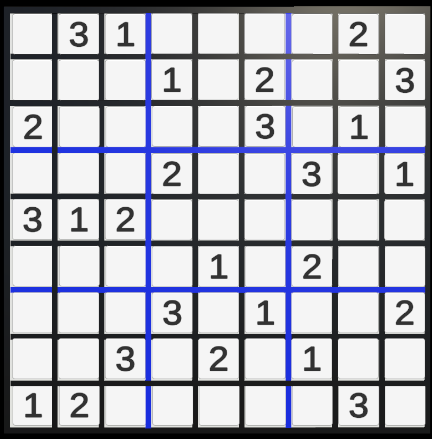
Opis wygenerowany automatycznie

~~-TABLICA NA START BĘDZIE WYPEŁNIONA ZERAMI, W KOLEJNYM ALGORYTMIE LOSOWO WYBIERA LICZBY I PRZYPISUJE WARTOŚĆ PILNUJĄC ZGODNOŚĆ Z KOLUMNĄ, WIERSZEM I SEKTOREM~~

Tablica generuje się strasznie długo i pozostawia mnóstwo pustych pól z uwagi na niezgodność z zasadami(10% tablicy jest puste)

~~-TABLICA NA START WYPEŁNIONA ZERAMI, KOLEJNY ALGORYTM WYPISUJE LICZBY OD 1 DO 9 W LOSOWYM WIERSZU I KOLUMNIE~~

Tablica generuje się krócej niż 5 sekund, ale dochodzi do ściany, gdy nie możemy zapełnić reszty z uwagi na niezgodność z zasadami gry



Backtracking?

~~Przez losowość często algorytm wybiera poprzednio wybrane komórki, co wydłuża czas.Backtracking gwarantuje rozwiązanie, kosztem czasu. Korutyna dodatkowo wydłuża czas pracy.~~

Backtracking+rekurencja

Sukces!

Tablica generuje się w kilka sekund, ale podane sudoku można rozwiązać na kilka możliwych sposobów.

Sudoku posiada nieoficjalny system „oceny” tzw. Grading, który nagradza plansze z tylko 1 rozwiązaniem łamigłówki, oraz odporność na sytuacje, gdzie gracz musi „strzelać” ze względu na brak jednoznacznej odpowiedzi.

~~Znalezienie konkretnej działającej planszy i mieszanie jej, co da 836,075,520 różnych wariantów?~~

<https://www.sudokuoftheday.com/creation>

PRZEBIEG GRY:

Wybieramy losowo elementy tablicy i wyświetlamy je:  
-łatwa plansza - 40-50 komórek odkryte

-średnia plansza - 30-40 komórek odkryte

* Przykładową planszę udało się rozwiązać za pomocą najprostszej metody i 1 „strzału”

-trudna plansza – 20-30 komórek odkryte

* Przez brak sprawdzeń co do możliwości wykonania, plansza generuje się praktycznie bez pewnych odpowiedzi, lub wyczerpujemy „pewniaki” po kilku ruchach

Można szukać algorytmów, które będą usuwać liczby pojedynczo z planszy i przy każdej iteracji sprawdzać czy dalej grę można rozwiązać korzystając z bazowych metod:

- ostatnia w rzędzie/kolumnie/sektorze (sytuacja, gdzie tylko 1 liczba pasuje, bo pozostałe pola są już zajęte innymi liczbami np.



- jedyny kandydat(sytuacja, gdzie poprzez eliminacje kolejnych liczb w komórce zostaje nam tylko 1 odpowiedź)

Obraz zawierający tekst, zrzut ekranu, numer, kwadrat

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

-ostatni w komórce(sytuacja, w której według zasad nie mamy innego miejsca na wstawienie konkretnej liczby)

Obraz zawierający zrzut ekranu, kwadrat, Prostokąt, Równolegle

Opis wygenerowany automatycznie

-LICZBY BĘDĄ WYBIERANE MYSZKĄ Z UI, CO PRZYPISUJE PLAYERPREFS, BY UNITY ROZRÓŻNIAŁ NASZE INTENCJE

-PLANSZA BĘDZIE 2 STRONNA – NUMERY SĄ ZNANE DLA GRY (Tablica tablic typu int), ALE ZAKRYTE DLA GRACZA, DO MOMENTU PRZYPISANIA PRZEZ NIEGO ODPOWIEDZI(Tablica guzików Button)

NUMER NA PLANSZY NA CZARNO, ZNACZY ŻE GRACZ MIAŁ RACJĘ, NUMER I PODŚWIETLENIE NA CZERWONO, ZNACZY ŻE GRACZ SIĘ POMYLIŁ PRZY WPISYWANIU LICZBY

-GRACZ MA PRAWO DO 10 POMYŁEK, PO CZYM GRA SIĘ RESETUJE, BY UNIEMOŻLIWIĆ „STRZELANIE NA ŚLEPO”

-AI BĘDZIE ZBIERAŁO PUNKTY ZA PODANIE DOBREJ ODPOWIEDZI I DUŻY BONUS ZA UKOŃCZENIE PLANSZY, GDY PODA BŁĘDNĄ ODPOWIEDŹ TRACI CZĘŚĆ PUNKTÓW

DODATKOWE FUNKCJONALNOŚCI:

-Timer- zlicza czas, może byś przydatny do porównań i statystyk oraz jako warunek przegranej, jak trening będzie przebiegał zbyt długo

-Licznik Błędów- obecnie nie robi nic poza zliczaniem błędów – w przyszłości będzie warunkiem resetu/porażki AI gdy osiągnie konkretną wartość.

CO DO SAMEGO BADANIA:

REINFORCEMENT LEARNING – przez doświadczenie agent uczy się podejmowania decyzji otrzymując kary i nagrody

Fajnie byłoby na końcu zrobić porównanie jak szybko AI rozwiąże zagadkę przy 1szej udanej próbie, 1000 udanej próbie i ostatecznej wytrenowanej wersji

Według mnie projekt jest w pełni możliwy do wykonania, chociaż metody bazujące bardziej na patternach i wykrywaniu konkretnych stanów typu sieci neuronowe byłyby znacznie lepsze, szczególnie na trudniejszych planszach, gdzie nie można polegać tylko na „wykluczaniu opcji”

Według projektów jakie wyszukałem w internecie, MLAgent radzi sobie dobrze z trudniejszymi łamigłówkami typu Mahjong.

Linki które mogą się przydać:

<https://norvig.com/sudoku.html>

<https://github.com/dimitri/sudoku>

<https://www.sudoku-solutions.com>

https://www.sudokuwiki.org/sudoku.html

PROBLEMY:

-LITERATURA

CIĘŻKO ZNALEŹĆ COŚ NAUKOWEGO OPARTEGO NA MATEMATYCE, A NIE NA WIEDZY POWSZECHNEJ, JEŚLI CHODZI O SAMO SUDOKU I JUŻ ODKRYTE ROZWIĄZANIA.

-ROZWIĄZANIE ŁAMIGŁÓWKI

JEST SZANSA, ŻE SUDOKU BĘDZIE NIEMOŻLIWE DLA ZWYKŁEGO UŻYTKOWNIKA UŻYWAJĄC BAZOWYCH METOD, A NAUCZENIE AI SKOMPLIKOWANYCH STRATEGII MOŻE POWODOWAĆ ZBĘDNE WYDŁUŻANIE TRENINGU ORAZ ZMIANĘ METOD UCZENIA.

ZOSTAWIAJĄC KILKA UZUPEŁNIONYCH PÓL LOSOWO, NASZA PLANSZA SUDOKU BĘDZIE NARAŻONA NA 2 DUŻE NIEDOGODNOŚCI:

-kilka możliwych rozwiązań do 1 planszy(zaawansowane łamigłówki posiadają 1 rozwiązanie)

-brak jednoznacznej odpowiedzi(mogą pojawić się takie sytuacje, gdzie do 1 pola będą pasowały 2 lub więcej odpowiedzi na raz w danym momencie,bez natychmiastowej informacji czy ruch jest dobry czy zły, problem może wyjść pod sam koniec planszy)

25.11.24

Wdrożenie do mlAgent

1. Package Manager -> ML Agents
2. Python, koniecznie w wersji 3.10.2, nowsze wersje mają problemy z bibliotekami pytorch,protobuf,numpy,mlagents i zależnościami między nimi, a wersje <3.9.\* oficjalnie niewspierane
3. Do ustawienia środowiska:

-cmd -> cd ścieżka/projektu

-py -m venv venv (tworzymy wirtualne środowisko)

-venv\Scripts\activate (wchodzimy do venva)

-py -m pip install –upgrade pip(opcjonalnie upgrade pip’a)

-pip install mlagents

-mlagents-learn (jeśli wyskoczy logo unity i informacja o nasłuchiwaniu na porcie, znaczy że wszystko działa)

Behavior parameters jest skryptem z biblioteki mlagenta

\*KONIECZNIE MUSI MIEĆ ZMIENIONĄ NAZWĘ Z BAZOWEJ

Decision requester jest kolejnym dodatkiem z biblioteki i musi znaleźć się na obiekcie ze skryptem dziedziczącym z klasy Agent

Do włączenia treningu:

Cmd -> mlagents-learn config.yaml –run-id=nazwa\_modelu

By włączyć wytrenowany model: Obraz zawierający tekst, zrzut ekranu, Oprogramowanie multimedialne, oprogramowanie

Opis wygenerowany automatycznie

Zmieniamy Model na wcześniej stworzony i wytrenowany

27.11.24

Wstępny System nagród i kar:

+ za każdą dobrze wstawioną liczbę

+ duży bonus za ukończenie całej planszy

+ mały mnożnik punktów za czas przejścia

- za każdą źle wstawioną odpowiedź

ML Agents nie może wchodzić w interakcje z elementami UI typu guziki, dlatego będzie trzeba zasymulować klikanie guzików przez agenta(Invoke),

Trzeba zmienić koncepcję z reagowaniem na elementy UI, na interakcję z tablicą danych oraz „wizualizację” działań agenta w UI

W większości projektów „robota grającego w grę”, to spora wada, wydaję mi się że w przypadku łamigłówek logicznych, można ją przekuć w sporą zaletę.

W takim przypadku wystarczy przekazywać sztucznej inteligencji stan tablicy i symulować klikanie guzików za pomocą Invoke. To podejście powinno być znacznie łatwiejsze do zaimplementowania.

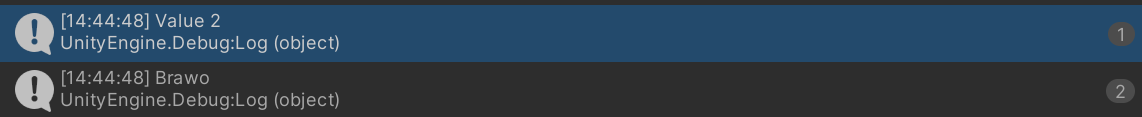
28.11.24

Nowa tablica przechowująca numery widoczne na planszy

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Inicjacja 1szego agenta:

Agent ciągle wybiera pola zaznaczone przez grę, 



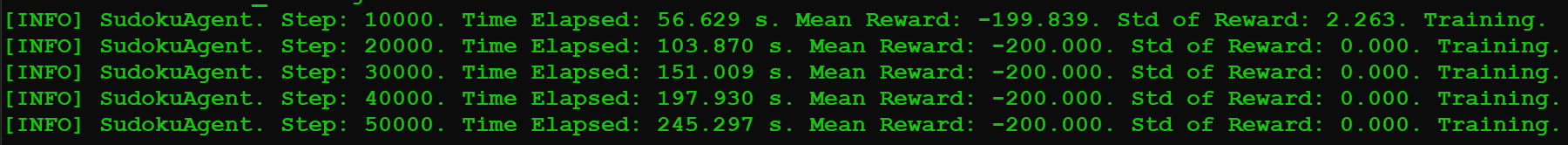
Ale zdarzyło mu się wstawienie dobrej i złej odpowiedzi

System kar i nagród wymaga sporych poprawek, gdyż wpędził ML Agenta w depresję – po kilku epizodach uznał za najbardziej opłacalne nie wykonywanie żadnego ruchu- w ten sposób nie ryzykuje utraty punktów.

29.11.24

Zmiana nagród na mniej restrykcyjne pomaga, ale model dalej preferuje się nie ruszać

Rozpoczęcie trenowania na losowych mapach okazało się porażką, powinienem zacząć od trenowania na 1 konkretnym seedzie

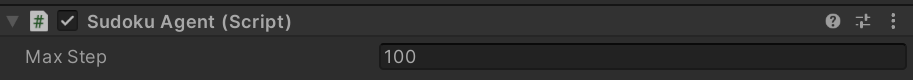


Po długiej sesji debugowania model zaczyna działac prawidłowoObraz zawierający zrzut ekranu, tekst

Opis wygenerowany automatycznie

~~Wymaga kilku poprawek i bardziej rygorystycznych kar, ale działa zgodnie z założeniem~~

Skrypt był obarczony błędami związanymi z definicją planszy



Max Step- odpowiada za ilość „kroków” jakie może wykonać AI przed zresetowaniem planszy, może również zdecydować by przedwcześnie przerwać epizod

Obraz zawierający zrzut ekranu, tekst

Opis wygenerowany automatycznie

Decision Period – jak często agent podejmuje decyzję o zmianie akcji,

Take Actions Between Decisions odpowiada za powielanie ruchu, do następnej decyzji

W przypadku gry planszowej, można potraktować każde uzupełnienie planszy jako turę, czyli co sekundę agent wykonuje 1 akcję na planszy

Obraz zawierający tekst, zrzut ekranu, Oprogramowanie multimedialne, oprogramowanie

Opis wygenerowany automatycznieContinuous Actions może się okazać dobre przy pracy nad modelem np. auta, postaci lub wszędzie gdzie musimy opisać dokładną pozycję na osi x,y ; dla sudoku wystarczą 2 dyskretne akcje – 1 przyjmująca rozmiar tablicy, 2- przyjmująca numery jaki chcemy wpisać

Stacked vectors- gdy chcemy,by agent rozpoznawał zmiany na planszy między ruchami

~~Epizod 3100- model stabilnie osiąga 4-6 trafień i powoli uczy się poprawiać błędy~~

~~Epizod 3500- coraz rzadziej wybiera oznaczone pola~~

Błędy w skrypcie

Udało się wyeksportować 1szy model

30.11.24

Zastosowałem bardziej restrykcyjne nagrody, od teraz model dostaje nieznaczny bonus(+0.1f) za podjęcie akcji, traci 1f za złą decyzję oraz zyskuje 2f za poprawny wybór i -2f za wstrzymanie się od wstawienia odpowiedz

Obraz zawierający zrzut ekranu, Wielobarwność, zieleń

Opis wygenerowany automatycznie

~~Wyrzucenie override Heuristic wpłynęło pozytywnie na trening, ale rezultat nie jest zadowalający, algorytm dalej popełnia błędy~~

~~Wydłużenie treningu do miliona kroków przynosi rezultaty, ale z uwagi na za mało elementów śledzonych również nie daje za wiele~~

Dane sprzed bugfixu

Pomysły:

Jeżeli po 500 000 kroków model dalej nie nauczył się zbyt wiele, może warto dodać jakieś nagrody za kamienie milowe:

Za każdy skończony rząd/kwadrat/kolumnę +5f

Za każdą dobrą odpowiedź Z RZĘDU +(2f+(seria\*0.5))  
  
Dorzucić więcej elementów do obserwacji przez agenta

1.12.24

Dokładny debug kodu agenta, w poprzednich podejściach błędnie definiował dobre i złe kroki

Trening powinien przebiegać w 3 etapach – łatwe plansze – 1, potem średnie plansze – 2, a koniec trudne plansze -3

Po milion lub 2 miliony kroków na daną trudność, start na trudniejszych planszach nie uczy zbyt wiele.

Kara za brak ruchu powinna być największa, by jak najszybciej zniechęcić agenta do wstrzymania, 2ga największa kara powinna być za wybieranie zajętych pól, a najniższa za błędną odpowiedź, by nie zniechęcać agenta do eksperymentowania i ryzyka.

Algorytm działa znacznie lepiej niż poprzednio, lecz wymaga kilku poprawek,

-Dalej zdarza się wybór zajętych pól

-Agent potrafi stracić punkty za przypisanie złej liczby, potem poprawić ją inną złą liczbą i spowrotem przypisać złą liczbę

-agent nie najlepiej rozumie zasady i trening idzie stosunkowo wolno, 1 milion ruchów to zdecydowanie za mało przy obecnej konfiguracji

5.12.24

Ogromny sukces, zabawy z heurystyką pozwoliły skuteczniej wypatrywać błędy w kodzie. Wprowadzenie kamienia milowego w postaci ukończenia kolumny/sektora/wiersza skutecznie zachęciły agenta do podejmowania bardziej przemyślanych decyzji

Obraz zawierający tekst, kwadrat, zrzut ekranu, Gry

Opis wygenerowany automatycznie

Obraz zawierający zrzut ekranu, tekst

Opis wygenerowany automatycznie

Step: 2500000

Agent zaczyna rozumieć ideę sudoku, na przykładzie 9 widać, że powoli rozumie zasadę 1 w rzędzie, 1 w kolumnie 1 w sektorze

Agent stabilnie osiąga 8 poprawnych ruchów na 1 planszy

Bardzo stara się nagrodę za ukończenie rzędu/kolumny/sektora.

Jeszcze za wcześnie, by agent stabilnie maksymalizował nagrodę za serię poprawnych trafień

Obraz zawierający tekst, zrzut ekranu, kwadrat, zagadka

Opis wygenerowany automatycznie

Na zdjęciu niżej widać, że agent jeszcze nie rozumie oczywistych zagrań, najpierw spróbował wstawić 4 w sektorze H, potem dopiero wstawił oczywista 4 w sektorze G, oraz pominął kompletnie pewną 5 w sektorze D

Obraz zawierający tekst, zrzut ekranu, kwadrat

Opis wygenerowany automatycznie

Obraz zawierający zrzut ekranu, zieleń, tekst

Opis wygenerowany automatycznie

Po raz 1szy AI osiągnęło dodatni wynik nagród

Obraz zawierający zrzut ekranu, zieleń

Opis wygenerowany automatycznie

Po 5 milionach kroków algorytm dalej się rozwija, chociaż znacznie zwalnia

Obraz zawierający zrzut ekranu, Czcionka, tekst

Opis wygenerowany automatycznie

Po wznowieniu treningu agent wykazuje przeuczenie

Zmiany:   
batch size = 256

Buffer size = 4096

Wynik:

Obraz zawierający wzór, zieleń, zrzut ekranu, materiał

Opis wygenerowany automatycznie

Dzięki zwiększeniu tych parametrów agent wolniej zdobywa doświadczenie, ale jest w stanie poprawiać wyniki do ~10 milionów ruchów, w kilku poprzednich logach widać tymczasowe stagnacje i niestabilność wyników

Algorytm po tak długim treningu stabilnie zaznacza pół planszy w każdym epizodzie

8.12.24

Ostatnie aktualizacje nie przyniosły większego skutku, dlatego napiszę je zbiorczo

Wydłużenie treningu do 10 000 000 kroków jest znacznie lepsze dla agenta niż 5 000 000 kroków, chociaż wymaga wyłączenia debug logów i zostawienia komputera przynajmniej przez noc

Stacked vectors mogło być niesamowicie pomocne, by agent zaobserwował trywialność wielu ruchów, ale nie odnotowałem większych różnic po zastosowaniu, może przy dłuższym treningu da radę odczuć różnicę i rzadsze stagnacje agenta, ale do miliona kroków działa bez zmian.

Zwiększenie kar i nagród jest standardową praktyką i wymaga prób na bieżąco, by do danej aktualizacji dopasować strategię

Dodanie wielu elementów do obserwacji na raz wymaga przemyślanych zmian w configu agenta

Istnieje możliwość „maskowania” decyzji agenta, ale jeszcze nie udało mi się ich sensownie zastosować. Mogłyby posłużyć do maskowania pól, które już są wypełnione, co oszczędziło by agentowi sporo kroków, a mi kilka linijek kodu i uproszczenie decyzji agenta

<https://docs.unity3d.com/Packages/com.unity.ml-agents@2.0/api/Unity.MLAgents.Actuators.IDiscreteActionMask.html>

Istniejące „AI sudoku solver” opierają się głównie na backtrackingu, heurystyce, rekurencji, co odpowiada mojej metodzie do tworzenia plansz, w przypadku agenta ML ciężko jest je zastosować ze względu na specyfikę zadania oraz wizualizację zachowań agenta

Śledzenie poczynań agenta w czasie rzeczywistym jest praktycznie niemożliwe, agent wykonuje 100 ruchów w niecałe 3 sekundy, spowolnienie czasu unity poza treningiem może pomóc w śledzeniu ruchów, widziałem również paczkę gdzieś na githubie do prezentacji agenta, która również głównie opiera się na spowolnieniu wytrenowanego modelu.

Sprawdzone:

- obserwacje planszy – podstawa

- obserwacje ilości pustych pól- łatwiej przedstawić 1 sumę, niż 30-60 obserwacji

~~- obserwacja ilości pełnych pól – nie robi agentowi różnicy~~

~~- stacked vectors – raczej do płynniejszych gier z reakcją na otoczenie~~

- suma w kolumnach i wierszach – ciężko stwierdzić, wydaje się delikatnie pomocne

- zmiany w configu – trzeba non stop dostosowywać konfigurację, dalej nie do końca rozumiem konkretne configi

- zmiany w nagrodach – koniecznie dostosowywać, robić hierarchię kar dla różnych błędów i kamienie milowe przed główną nagrodą

- seria – poprawia wyniki agenta, nie pamiętam na ile agent sobie radził bez niej

-blokowanie agenta przed pominięciem decyzji – zdecydowanie potrzebne w grach turowych

-różne poziomy trudności – trening przebiega znacznie lepiej na prostszych planszach, ale wytrenowany osiąga minimalnie gorsze wyniki na trudniejszych planszach, to może być prawdziwa siła ml agenta w porównaniu do sprawdzonych przeze mnie modeli AI do sudoku, gdzie różnice były znaczne między prostymi planszami, a trudnymi

Do sprawdzenia :

-maski – tak jak wyżej, znaczna oszczędność i „błędoodporność”

-stacked vectors – do sprawdzenia na dłuższym treningu, bez dodatkowych zmiennych

-dodatkowy skrypt do gry, którym steruje człowiek i posiada kompletnie przepisany zestaw funkcji agenta, w celach dokładniejszej analizy i bardziej przewidywalnego bugfixu + rozwiązuje problem z czasem reakcji, na decyzje agenta.

-suma każdego sektora -jak w przypadku kolumn i wierszy, może się przydać jako informacja jakie liczby pasują – każdy wiersz,kolumna, sektor powinien mieć sumę 45

-drobna kara za każdy ruch (np. -0.1f), co może skłonić agenta, do jak najszybszego zakończenia planszy

-kara za oczywiste błędy – jeśli agent dalej nie rozumie podstaw sudoku, można wystosować większe kary, np. gdy wstawi liczbę, mimo że ta liczba jest w tym samym sektorze,wierszu lub kolumnie na jego planszy

-użycie TensorBoard by na bieżąco robił statystyki z treningu agenta

- metoda curriculum learning – w trakcie treningu agent stopniowo otrzymuje większy poziom trudności np. 2 miliony kroków na prawie rozwiązanej planszy – 2 miliony kroków na obecnej łatwej planszy – 2 miliony kroków na średniej planszy – 4 miliony kroków na najtrudniejszej planszy

- podejście hybrydowe – zastosowanie heurystyki jako „przewodnika” po przestrzeni eksploracji

\*nie jestem pewien na ile metoda jest zgodna z założeniami pracy inżynierskiej oraz samym uczeniem maszynowym, bo równie dobrze można to potraktować jako wytłumaczenie agentowi zasad jakie miał się wyuczyć w procesie reinforcement learningu. Wpadłem na to przypadkiem, jak zastanawiałem się, jakby można kogoś nauczyć sudoku(np. agenta) i doszedłem do wniosku, że nie słyszałem o sytuacji gdzie ktoś nie dostał planszy do uzupełnienia z instruktażem na zasadzie „musi być po 1 liczbie od 1 do 9 w każdym wierszu, kolumnie i kwadracie”. Trzymam ten pomysł jako ostatnią deskę ratunku, gdyby wszystko inne zawiodło