Wykorzystanie sztucznej inteligencji w symulowaniu zachowań biologicznych

(Utilizing artificial intelligence to model biological behaviours)

Tymoteusz Kaczorowski

Praca inżynierska

Promotor: dr Jakub Kowalski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

14 lutego 2018 r.

Tymoteusz Kaczorowski

.....

.....
(adres zameldowania)

.....

.....
(adres korespondencyjny)

PESEL: .....

e-mail: .....

Wydział Matematyki i Informatyki
stacjonarne studia I stopnia
kierunek: informatyka
nr albumu: 273882

Oświadczenie
o autorskim wykonaniu pracy dyplomowej

Niniejszym oświadczam, że złożoną do oceny pracę zatytułowaną *Wykorzystanie sztucznej inteligencji w symulowaniu zachowań biologicznych* wykonałem samodzielnie pod kierunkiem promotora, dr. Jakuba Kowalskiego. Oświadczam, że powyższe dane są zgodne ze stanem faktycznym i znane mi są przepisy ustawy z dn. 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (tekst jednolity: Dz. U. z 2006 r. nr 90, poz. 637, z późniejszymi zmianami) oraz że treść pracy dyplomowej przedstawionej do obrony, zawarta na przekazanym nośniku elektronicznym, jest identyczna z jej wersją drukowaną.

Wrocław, 14 lutego 2018 r.

(czytelny podpis)

Streszczenie

Metody sztucznej inteligencji znajdują zastosowanie w wielu dziedzinach, jak analizowanie zachowań użytkowników stron internetowych, kontrolowanie autonomicznych samochodów czy granie w gry (np. szachy). W tej pracy pokazuję wykorzystanie algorytmu Q-Learning do stworzenia symulacji modelującej w uproszczony sposób zachowanie roślinożercy w środowiskach pozbawionych drapieżników. W tym celu zaimplementowałem wizualizację i model środowiska oraz przetestowałem różne parametry algorytmu względem wielu ustawień świata, w którym uczy się on żyć.

Artificial intelligence has many applications, such as analyzing behaviours of website users, controlling self-driving cars or playing games (such as chess). In this thesis, I present a way of using the Q-Learning algorithm to create a simulation modelling the behaviour of an herbivore in a predator–free environment. I created a visualisation and the environment's model, and tested various parameters of the algorithm in the context of many example world settings.

Spis treści

Rozdział 1.

Wstęp

W ramach tej pracy stworzyłem grę przedstawiającą uproszczony model środowiska naturalnego, której przeznaczeniem jest testowanie metod sztucznej inteligencji w celu modelowania zachowań biologicznych. Aplikacja pozwala kontrolować stan symulacji i obserwować, jak AI uczy się grać optymalnie.

Agentem w tej grze jest osiołek, którego zadaniem jest przeżycie odżywiając się ostem. Poczynaniami osiołka kieruje kontroler oparty o algorytm Q-learning [1], który jest odmianą uczenia ze wzmocnieniem (ang. *reinforcement learning*) [6]. Metoda ta polega na uczeniu się optymalnych zachowań przez analizę nagród i kar płynących z podejmowanych decyzji i jest ważną gałęzią uczenia maszynowego [5].

Wartymi wspomnienia przykładami zastosowania uczenia ze wzmocnieniem jest na przykład TD-Gammon [7], który nauczył się grać w grę Backgammon w oparciu o jedynie wyniki gier rozgrywanych z samym sobą, czy AlphaGo, który w 2016 roku był najlepszym programem grającym w Go [3]

Ostatnia wersja AlphaGo, korzystająca z czystego RL (bez wiedzy ludzkich ekspertów wykorzystywanej przez wersję z 2016) wygrywa z poprzednią 100 na 100 gier [4]. Innym znanym zastosowaniem uczenia ze wzmocnieniem jest gra w szachy – przykładem jest program KnightCap wykorzystujący algorytm TDLeaf(λ) [2].

1.1. Motywacja

Założeniem projektu było stworzenie symulacji inspirowanej światem rzeczywistym, w której egzystują byty zdolne do nauczenia się strategii pozwalającej na optymalne funkcjonowanie. Stworzony model powinien nie tylko pozwolić sztucznej inteligencji na naukę, ale też, po nauczeniu, pozwalać na zaobserwowanie ciekawych zachowań.

1.2. Realizacja

Na potrzeby realizacji celu powstały trzy elementy:

1. Środowisko graficzne służące do wizualnego reprezentowania i kontrolowania symulacji
2. Silnik symulacji, będący grą dla jednego gracza, w której osiołek (gracz) ma za zadanie przeżyć jak najdłużej odżywiając się roślinami
3. Ucząca się sztuczna inteligencja kontrolująca osiołka

1.3. Narzędzia

Projekt w całości został zrealizowany w języku C# z wykorzystaniem silnika Windows Presentation Foundation do części wizualnej.

Rozdział 2.

Zasady symulacji

Symulacja stworzona na potrzeby tej pracy to prosta, turowa gra dla jednego gracza rozgrywająca się na ciągłej, dwuwymiarowej, kwadratowej planszy, na której żyje osiołek. Osiołek musi odżywiać się ostem by przeżyć, lecz poruszanie się po planszy go męczy, powodując dodatkowe zapotrzebowanie na pożywienie.

2.1. Byty

Wszystkie byty posiadają następujące cechy:

1. Pozycja na planszy
2. Identyfikator

2.1.1. Osiołek

Osiołek jest bytem kontrolowanym przez gracza. W danej turze na planszy znajduje się dokładnie jeden osiołek. Osiołek posiada następujące cechy zdefiniowane w ustawieniach [sekcja 3.3.]:

1. **Mass** – masa, która opisuje stopień wypełnienia żołądka i umożliwiająca osiołkowi funkcjonowanie poprzez spalanie jej
2. **MovementSpeed** – maksymalny dystans, który może pokonać w ciągu tury
3. **InteractionDistance** – zasięg, na którym możliwe jest jedzenie ostu
4. **BiteSize** – maksymalna masa, którą może przyswoić z rośliny w ciągu tury
5. **StomachCapacity** – maksymalny rozmiar żołądka
6. **SightRange** – zasięg, w którym inne byty są dla osiołka widoczne

7. **PassiveWork** – ilość masy zużywanej co turę
8. **MovementWork** – ilość masy zużywanej na potrzeby ruchu

2.1.2. Oset

Osiołek żywi się ostem. Na planszy może znajdować się co najwyżej określona w ustawieniach symulacji liczba roślin. Każdy oset ma nastepujące cechy:

1. **Mass** – określa ilość dostępnego pożywienia w danej roślinie
2. **MaxMass** – maksymalna masa, do której roślina może urosnąć
3. **RegrowthRate** – stały przyrost masy na turę

Maksymalna masa i przyrost masy na turę danej rośliny są losowane z rozkładem jednostajnym na przedziałach zdefiniowanych w ustawieniach [sekcja 3.3.]. Początkowa masa ostu równa jest jego masie maksymalnej.

2.2. Początek gry

Przed pierwszą turą w losowych miejscach tworzone jest tyle roślin, ile wskazują ustawienia. Następnie w losowym miejscu umieszczany jest osiołek, którego początkowa masa wynosi połowę maksymalnej.

2.3. Przebieg gry

Każda tura przebiega następująco:

1. Wszystkie rośliny, których obecna masa jest większa niż zero zwiększają swoją masę w oparciu o **RegrowthRate**, nie przekraczając **MaxMass**.
2. Wszyskie byty, których masa nie przekracza zera umierają i znikają. Jeśli umiera osiołek symulacja jest resetowana.
3. Osiołek otrzymuje opis otoczenia i swojego stanu.
4. Kontroler wybiera akcję dla osiołka w oparciu o ten opis.
5. Osiołek wykonuje akcję i spala część swojej masy określoną wzorem: $\mathbf{PassiveWork} + \mathbf{MovementWork} \cdot \frac{distance}{\mathbf{MovementSpeed}}$, gdzie *distance* to dystans przebyty w danej turze.
6. Jeżeli liczba roślin na planszy nie przekracza maksymalnej, w losowym miejscu na planszy pojawia się nowa roślina, nie częściej niż raz na liczbę tur opisaną w ustawieniach.

2.4. Akcje

W każdej turze siołek może wykonać jedną z poniższych akcji:

1. Pominięcie tury.
2. Próba zjedzenia wybranej rośliny, która przekazuje część swojej masy osiołkowi jeśli ten znajduje się w odpowiednim zasięgu. Ilość zaabsorbowanej masy wynosi.

   $min($**BiteSize**, **plant.Mass**, **StomachCapacity - donkey.Mass**$)$
3. Poruszenie się w stronę zadanego punktu o wybrany dystans z zakresu $[0,$ **MovementSpeed**$]$.

Rozdział 3.

Aplikacja

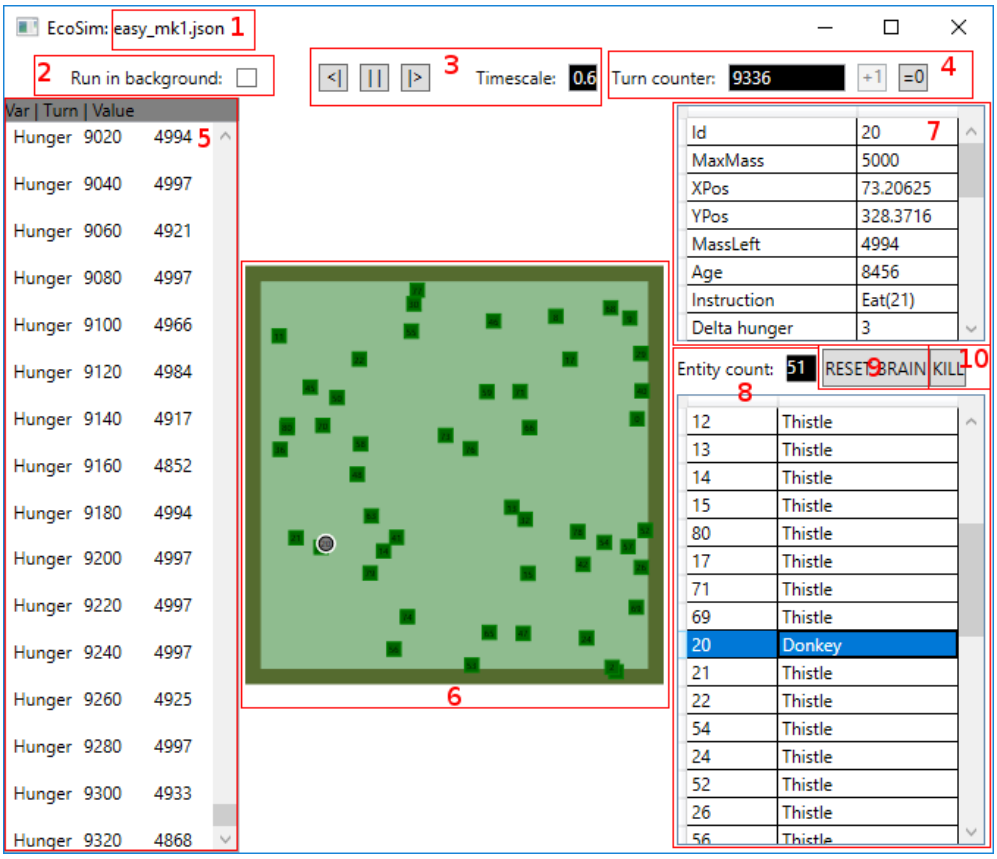
Aplikacja udostępnia prostą wizualizację oraz metody kontrolowania i podglądu stanu symulacji. Ten rozdział opisuje interfejs, obsługę i działanie aplikacji.

3.1. Działanie aplikacji

Po uruchomieniu użytkownik proszony jest o wybranie pliku `.json` z ustawieniami. Następnie tworzona jest instancja kontrolera osiołka i uruchamiane są zadania cykliczne: odświeżanie grafiki i obliczanie kolejnych tur symulacji. Program przyjmuje jeden opcjonalny argument wywołania: liczbę całkowitą określającą liczbę odświeżeń grafiki na sekundę. Wartość domyślna tego argumentu wynosi 20.

W momencie śmierci osiołka symulacja jest resetowana, ale licznik tur i kontroler zachowują swoje stany.

3.2. Opis interfejsu



1. Nazwa wybranego pliku z ustawieniami.
2. Checkbox pozwalający wyłączyć odświeżanie grafiki i obliczać kolejne tury symulacji tak szybko, jak jest to możliwe.
3. Przyciski kontrolujące częstotliwość obliczania kolejnych tur symulacji.
4. Licznik tur. Przycisk *+1* pozwala obliczyć kolejną turę symulacji gdy **Timescale** jest równe 0. Przycisk *=0* resetuje licznik tur.
5. Log, w którym pojawiają się wiadomości tworzone przez symulację. W przykładzie jest to bieżąca tura i stan żołądka osiołka wypisywane co 20 tur.
6. Plansza reprezentująca stan symulacji.
7. Szczegóły obecnie wybranego bytu. Byt można wybrać klikając go na planszy lub na liście bytów.
8. Lista bytów znajdujących się na planszy.
9. Przycisk pozwalający wywołać metodę *Reset()* na kontrolerze osiołka, która ponownie inicjalizuje jego stan wartościami początkowymi.
10. Przycisk pozwalający natychmiastowo zabić obecną instancję osiołka.

3.3. Ustawienia

Symulacja oraz kontroler są parametryzowane plikiem `.json`. Poniżej przedstawiam przykładowy plik z ustawieniami oraz ich opisami:

```
{
"Brain": "Mark1", //"Mark1" lub "Mark0", wybiera rodzaj kontrolera

"LogDeaths" : false, //wartość true sprawia, że po każdej śmierci
    osiołka zalogowana zostanie tura jego urodzin i wiek w momencie
    zgonu

"LogHungerEvery": 20, //loguje stan żołądka osiołka co każde
    LogHungerEvery tur. Wartość 0 wyłącza tę opcję.

"InitialPlantCount": 20, //liczba roślin pojawiająca się na początku
    symulacji

"MaxPlantCount": 50, //maksymalna liczba roślin na planszy

"NewPlantFrequency": 5, //liczba tur między pojawieniami się nowych
    roślin

"MinPlantMass": 100, //dolna granica przedziału, z którego losowana
    jest maksymalna masa rośliny
"MaxPlantMass": 500, //górna granica w/w przedziału

"MinGrowthRate": 30, //dolna granica przedziału, z którego losowany
    jest parametr RegrowthRate rośliny
"MaxGrowthRate": 50, //górna granica w/w przedziału

// opisane w rozdz. 2 pracy:
"MovementSpeed": 20,
"InteractionDistance": 22,
"BiteSize": 100,
"StomachCapacity": 5000,
"SightRange": 88,
"PassiveWork": 3,
"MovementWork": 5,

// opisane w rozdz. 4 pracy:
"BrainBaseActionScore": 0.0022,
"BrainDiscount": 0.6,
"BrainLearningRate": 0.148,
"BrainLearningRateDamping": 0.99999,
"BrainBaseActionScoreDamping" : 0.99952,
"BrainProbabilityExponent": 1.6
}
```

Rozdział 4.

Sztuczna inteligencja

Poczynaniami osiołka kieruje kontroler implementujący algorytm Q-learning. W tym rozdziale opiszę szczegóły implementacji kontrolera.

Wybrany algorytm w prosty sposob pozwala na uczenie się, jakie decyzje podejmować w danym kontekście utrzymując balans między wybieraniem natychmiastowej nagrody, a dążeniem do stanów o większym potencjale zdobycia nagrody, nawet jeśli oznacza to mniejszy zysk krótkofalowy.

4.1. Rodzaje kontrolerów

Zaimplementowane zostały dwa rodzaje kontrolerów, *Mark1* i *Mark0*. Różnią się jedynie liczbą stanów, które rozróżniają – *Mark0* jest „krótkowzroczny" i nie analizuje pożywienia poza zasięgiem interakcji. *Mark1* uwzględnia w swojej przestrzeni stanów wszystkie rośliny w zasięgu wzroku, największą uwagę zwracając na te, które znajdują się bliżej agenta.

4.2. Kwantyzacja stanu

Kontroler w każdej turze symulacji otrzymuje opis stanu osiołka:

1. **Hunger** – liczba z zakresu [0-1] równa $1 - \frac{\textbf{donkey.Mass}}{\textbf{donkey.MaxMass}}$.
2. Pozycję osiołka.
3. Pozycje i masy wszystkich roślin w zasięgu wzroku osiołka.

Na potrzeby działania algorytmu stan ten sprowadzany jest do liczby 19-bitowej (11-bitowej dla kontrolera *Mark0*). Składa się na nią kolejno:

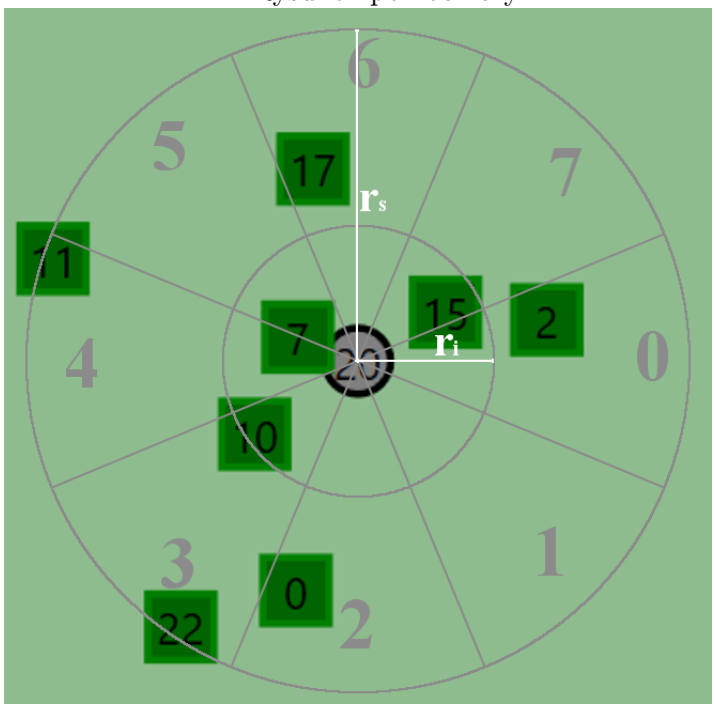
1. **Penalty** - liczba 3-bitowa równa $\lfloor \textbf{Hunger} \cdot 7 \rfloor$.

2. **FoodRichDirections** - liczba 8-bitowa wyznaczająca kierunki bogate w pożywienie (tylko w kontrolerze *Mark1*).

3. **CloseFoodDirections** - liczba 8-bitowa wyznaczająca kierunki, w których pożywienie znajduje się w zasięgu interakcji.

Kontroler rozróżnia osiem kierunków. Kolejne bity liczb **FoodRichDirections** i **CloseRichDirections** wyznaczają kolejne kierunki.

4.2. Rysunek pomocniczy



Niech:

- r_i oznacza zasięg interakcji
- $L(a,b) = \begin{cases} 1, & \text{gdy } a \leqslant b \\ 0, & \text{w p. p.} \end{cases}$
- $P_{k,0}...P_{k,n_k}$ reprezentuje wszystkie rośliny w zasięgu wzroku (r_s) w kierunku k, dla $k = 0...7$.
- $M(P)$ oznacza masę danej rośliny P
- $D(P)$ oznacza odległość rośliny P od osiołka
- $C_k = \sum_{i=0}^{n_k} M(P_{k,i}) \cdot L(D(P_{k,i}), r_i)$
- $R'_k = \sum_{i=0}^{n_k} \frac{M(P_{k,i})}{D(P_{k,i}) - r_i + 1} \cdot (1 - L(D(P_{k,i}), r_i))$
- $R_k = \frac{R'_{(k-1) \mod 8} + R'_{(k+1) \mod 8}}{2} + R'_k$

Wtedy:
CloseFoodDirections = $\sum_{d=0}^{7} 2^d \cdot L(\frac{1}{9} \cdot \sum_{c=0}^{7} C_c, C_d)$
FoodRichDirections = $\sum_{d=0}^{7} 2^d \cdot L(\frac{1}{9} \cdot \sum_{c=0}^{7} R_c, R_d)$

Tak więc bit **CloseFoodDirections** jest równy 1, jeśli w wyznaczanym przez niego kierunku znajduje się więcej jedzenia w zasięgu interakcji niż przeciętnie.

Bit **FoodRichDirections** jest równy 1, jeśli w wyznaczanym przez niego kierunku, poza zasięgiem interakcji znajduje się (po przeskalowaniu mas względem odległości i uwzględnieniu kierunków sąsiednich) więcej jedzenia niż przeciętnie.

4.3. Akcje

Instrukcje, które kontroler może wydać osiołkowi są uproszczone, podobnie jak stany. Akcja w rozumieniu kontrolera składa się z dwóch elementów:

- Rodzaj - *pominięcie tury*, *ruch*, lub *jedzenie*
- Kierunek - liczba od 0 do 7

Akcja taka jest po wybraniu tłumaczona na instrukcję w rozumieniu silnika symulacji. Akcja *jedzenie* obiera na cel najbliższą roślinę w danym kierunku. Akcja *ruch* przemieszcza agenta o maksymalny możliwy dystans w kierunku odpowiadającym zadanemu, dla uproszczenia odchylonym w stronę najbliższej rośliny w tym kierunku.

4.4. Parametry kontrolera

Parametry kontrolera definiowane są w pliku .json z ustawieniami. Są to:

1. **ProbabilityExponent** – zwiększa różnicę w prawdopodobieństwie między najlepszymi a najgorszymi akcjami
2. **StartLearningRate** – początkowa wartość parametru α, który wpływa na skalę zmian wprowadzanych do stanu kontrolera po każdej decyzji
3. **LearningRateDamping** – zmniejsza z czasem wartość parametru α kontrolera poprzez mnożenie go przez tę liczbę w każdej turze
4. **StartBaseActionScore** – początkowy **BaseActionScore**, który zapewnia niezerowe prawdopodobieństwo wszystkim możliwym akcjom
5. **BaseActionScoreDamping** – zmniejsza faktyczny **BaseActionScore** kontrolera z czasem poprzez mnożenie go przez tę liczbę w każdej turze

6. **Discount** – wartość parametru γ, który wpływa na balans między rozważaniem zysku krótkofalowego i długofalowego

4.5. Nauka

Kontroler wykorzystuje prosty Q-learning oparty o tabelkę definiującą funkcję Q, która każdej parze $(stan, akcja)$ przyporządkowuje pewną wartość, która jest modyfikowana w czasie nauki. Na początku funkcja ta ma wartość 0 dla wszystkich argumentów.

Przyjmijmy, że w turze t osiołek znajdował się w stanie S_t, jego **Hunger** wynosił H_t i podjął akcję A_t. Doprowadziło go to do stanu S_{t+1}, w którym **Hunger** wynosi H_{t+1}. Niech $\Delta H = H_t - H_{t+1}$, $\alpha_t = \textbf{StartLearningRate} \cdot \textbf{LearningRateDamping}^t$ Wtedy:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) \cdot (1 - \alpha_t) + \alpha_t \cdot (\Delta H + \gamma \cdot max_a Q_t(s_{t+1}, a))$$

Dla wszystkich (s, a) różnych od (s_t, a_t), $Q_{t+1}(s, a) = Q_t(s, a)$

4.6. Podejmowanie decyzji

Po zaktualizowaniu funkcji Q wybierana jest akcja do podjęcia w danej turze. Każda akcja ma pewne pradopodobieństwo zostania wybraną, zależne od wartości funkcji Q. Niech $\beta_t = \textbf{StartBaseActionScore} \cdot \textbf{BaseActionScoreDamping}^t$. Wtedy prawdopodobieństwo akcji a w stanie s_t wynosi:

$P_t(s_t, a) = \frac{p_t(s_t, a)}{\sum_{a'} p_t(s_t, a')}$, gdzie

$p_t(s_t, a) = (Q_t(s_t, a) - min_{a'} Q_t(s_t, a') + \beta_t)^{\textbf{ProbabilityExponent}}$

Parametr β zapewnia niezerowe prawdopodobieństwo każdej możliwej akcji, dzięki czemu akcje uznane wcześnie za dobre nie dominują tych jeszcze niewypróbowanych. **ProbabilityExponent** zwiększa szanse wybrania dobrych akcji zmniejszając szanse wybrania złych.

Rozdział 5.

Eksperymenty

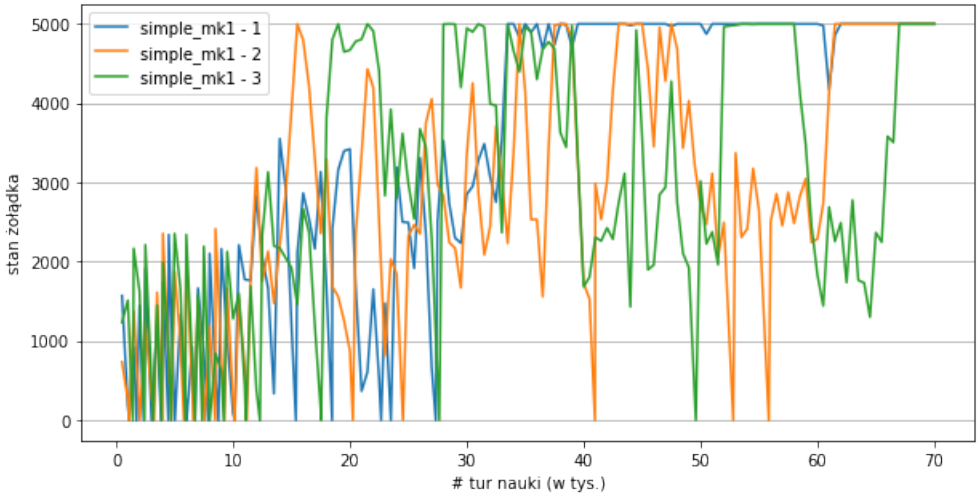
W ramach testów kontrolera przygotowałem kilka zestawów ustawień. W tym rozdziale opiszę, jak zaprojektowana przeze mnie sztuczna inteligencja radzi sobie w różnych środowiskach.

Parametry algorytmu uczącego były dobierane do wymyślonych ustawień świata w sposób eksperymentalny. Ostateczne pliki ustawień uwzględnione w tej pracy zawierają najlepsze z wypróbowanych parametrów.

5.1. Ustawienia *simple_mk1* i *simple_mk0*

Te ustawienia tworzą dość prosty świat – osiołek może zadomowić się w pobliżu jednego ostu i odżywiać się nim w nieskończoność, bo w żadnej turze nie zużywa więcej energii niż najmniejszy możliwy przyrost masy rośliny. I do takiej właśnie strategii dochodzi za każdym razem kontroler *Mark1*.

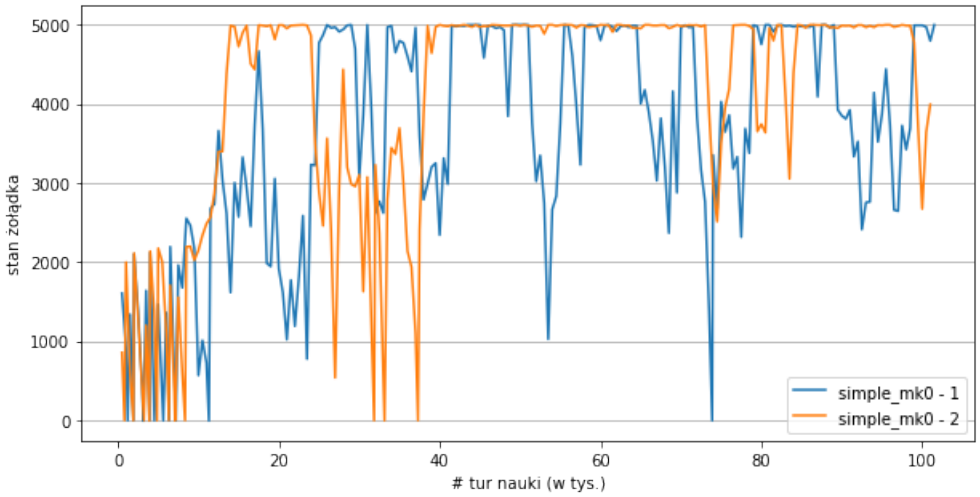
Stan żołądka osiołka względem liczby tur poświęconych na uczenie AI
(częstotliwość próbkowania: raz na 500 tur oraz przy każdym zgonie)



Na wykresie narysowane są trzy łamane odpowiadające trzem różnym uruchomieniom programu z tymi samymi ustawieniami. Każda z nich reprezentuje w istocie wiele instancji osiołka – wartość zero na wykresie oznacza zgon, po którym symulacja się resetuje, ale kontroler zachowuje wyniesione doświadczenia.

Z rysunku można wywnioskować, że różne uruchomienia dochodzą do strategii optymalnej w różnym czasie. Mimo to można dostrzec, że z upływem czasu zagęszczenie zgonów się zmniejsza, a wykres zaczyna oscylować w okolicach wyższych wartości. Podobne wnioski można odnieść przy wszystkich innych ustawieniach, więc dla czytelności w następnych wykresach przedstawiał będę tylko dwa uruchomienia.

Stan żołądka osiołka względem liczby tur poświęconych na uczenie AI
(częstotliwość próbkowania: raz na 500 tur oraz przy każdym zgonie)



Ten wykres przedstawia dwa uruchomienia z tymi samymi ustawieniami świata dla kontrolera *Mark0*. Ze względu na fakt, że kontroler ten nie analizuje ostów znajdujących się poza zasięgiem interakcji nie dochodzi on do dokładnie tego samego stanu co *Mark1*. Świat budowany przez te ustawienia nie ma dużego zagęszczenia roślin. Oznacza to, że częstokroć gdy błądzący osiołek w końcu natrafia na oset zjada go w całości.

Mimo ograniczeń kontrolera, w każdej próbie uruchomienia programu na tych ustawieniach osiąga on stan, w którym osiołek już więcej nie umiera.

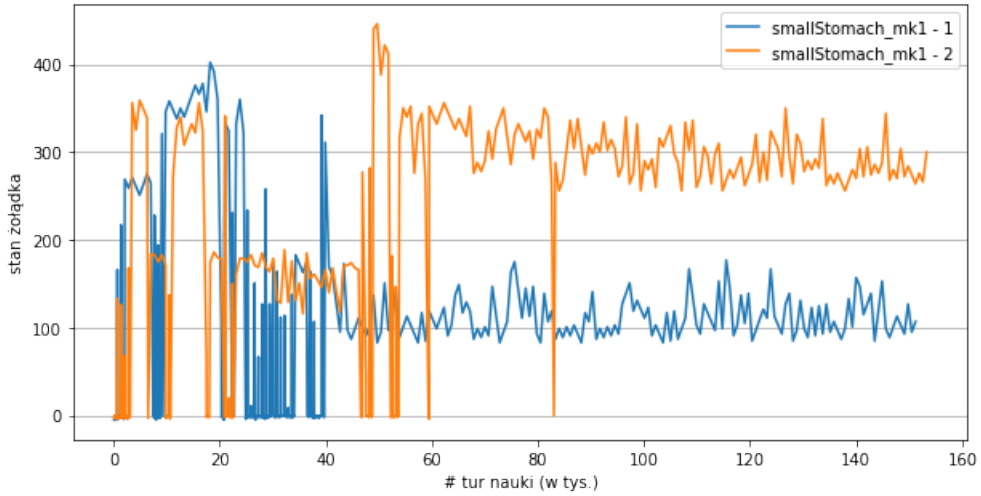
5.2. Ustawienia *smallStomach_mk1* i *smallStomach_mk0*

W przypadku tych ustawień osiołek ma mało czasu na znalezienie pożywienia ze względu na mały rozmiar żołądka i relatywnie duże zużycie energii na turę. Utrudnia to kontrolerowi uczenie się, jednak i tu osiągana jest w pewnym momencie strategia pozwalająca na przeżycie w nieskończoność.

Sprawę ułatwia fakt, że osiołek nigdy nie zjada żadnej rośliny w całości – każda

odrasta szybciej, niż ten jest w stanie absorbować jej masę, więc strategia z ustawień *simple* wciąż jest adekwatna.

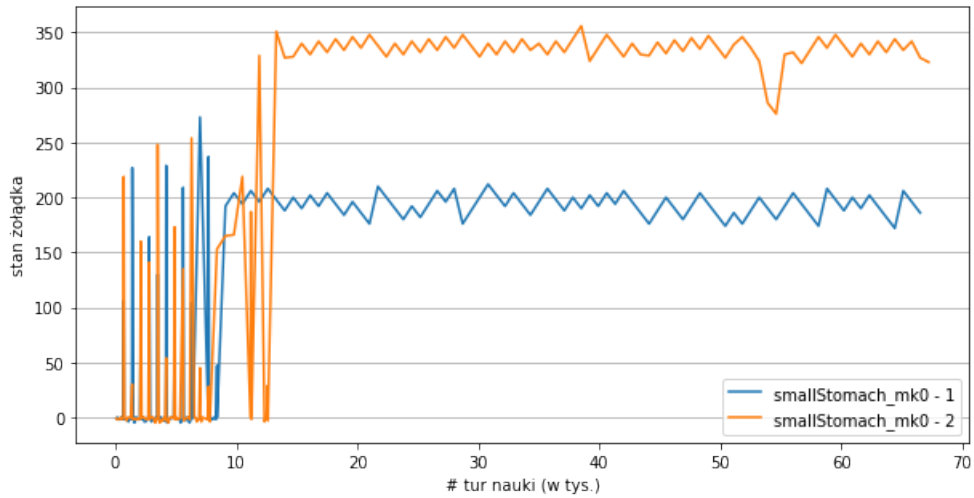
Stan żołądka osiołka względem liczby tur poświęconych na uczenie AI
(częstotliwość próbkowania: raz na 700 tur oraz przy każdym zgonie)



Choć na wykresie można dostrzec trochę mniejsze zagęszczenie zgonów wraz z postępem nauki, kontroler *Mark1* daje wiele podobnie mało satysfakcjonujących żywotów i ostatni, w którym stosuje już efektywną strategię.

Ciekawym jest, że w przeciwieństwie do ustawień *simple* stan żołądka po osiągnięciu stanu ostatecznego może oscylować wokół różnych wartości zależnie od uruchomienia.

Stan żołądka osiołka względem liczby tur poświęconych na uczenie AI
(częstotliwość próbkowania: raz na 700 tur oraz przy każdym zgonie)



Podobnie jak w przypadku *Mark1*, testy kontrolera *Mark0* nie wykazały dużej tendencji wzrostowej w jakości strategii względem czasu, aż do osiągnięcia strategii pozwalającej na nieśmiertelność. Także i on zależnie od uruchomienia osiąga stany,

w których stan żołądka oscyluje wokół różnych wartości.

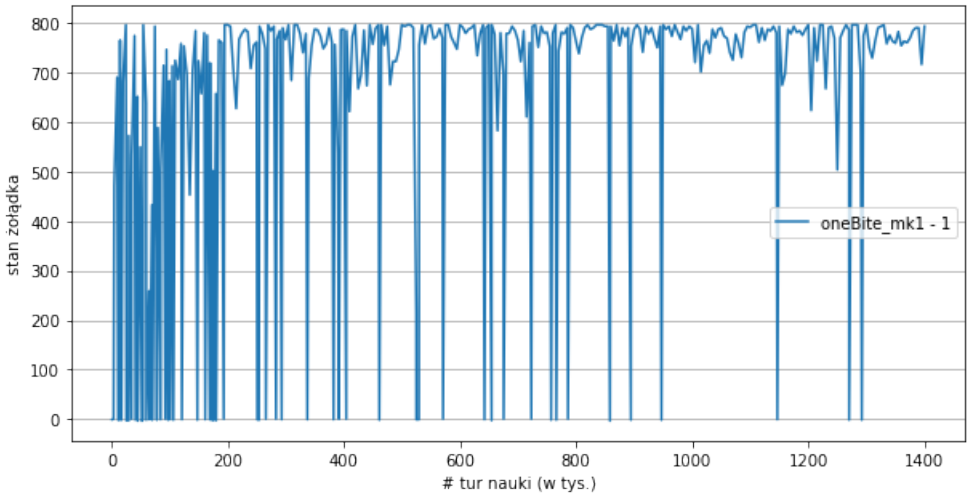
Mark0 osiąga jednak strategię efektywną znacznie szybciej - mniejsza liczba rozróżnianych stanów wydaje się być w przypadku tych ustawień zaletą. Analizowanie odległych roślin nie przynosi tu korzyści, za to zwiększa czas potrzebny na naukę.

5.3. Ustawienia *oneBite_mk1*

To najtrudniejsze z przygotowanych przeze mnie ustawień. Wszystkie rośliny mają jednakową masę, wystarczająco małą by osiołek mógł zjeść je w jednej turze i nie regenerują się. Oznacza to, że zjedzenie rośliny często prowadzi do mało korzystnego stanu, w którym trzeba się przemieścić, by znaleźć kolejną. Trudno zatem znaleźć odpowiednie parametry – mały **Discount** prowadzi do kontrolera, który preferuje się nie przemieszczać, zaś duży zmniejsza atrakcyjność jedzenia.

Spośród wszystkich sprawdzonych ustawień *oneBite* daje, po nauczeniu, najciekawszy do obserwowania kontroler, ponieważ zasady świata zmuszają go do nauczenia się, jak radzić sobie w zmiennym środowisku i systematycznie odnajdywać nowe źródła pożywienia, gdy dotychczas wykorzystywane się wyczerpią.

Stan żołądka osiołka względem liczby tur poświęconych na uczenie AI
(częstotliwość próbkowania: raz na 1000 tur oraz przy każdym zgonie)



Tym razem wszystkie uruchomienia dają bardzo podobne wykresy, więc dla czytelności przedstawione zostało tylko jedno. Można dostrzec mniejsze zagęszczenie zgonów wraz z czasem nauki, co sugeruje dążenie do efektywnej strategii. Mimo to w żadnym z wykonanych przeze mnie testów najdłuższy zarejestrowany żywot nie przekraczał 400 tysięcy tur.

Kontroler *Mark0* nie daje dla ustawień *oneBite* satysfakcjonujących wyników – żaden zarejestrowany żywot nie przekroczył 100 tysięcy tur, a zagęszczenie zgonów

nie maleje z czasem.

Rozdział 6.

Podsumowanie

Powstała w ramach tej pracy aplikacja pozwala na tworzenie i obserwowanie wielu symulacji, w których kontroler w różny sposób dąży do odmiennych strategii przeżycia. Przeprowadzone eksperymenty pokazują, że zaimplementowany algorytm istotnie uczy się efektywnych zachowań w przygotowanych przykładach środowisk.

Aplikację można w łatwy sposób rozszerzyć o bardziej rozbudowane zasady symulacji i kontrolery. Dużą część pracy potrzebnej do stworzenia ciekawej symulacji stanowią jednak szukanie odpowiednich ustawień oraz testy, czynność którą można i warto by zautomatyzować.

Bibliografia

[1] Q-learning — Wikipedia, The Free Encyclopedia. [online], 2018.

[2] J. Baxter, A. Tridgell, and L. Weaver. Learning to Play Chess Using Temporal Differences. *Machine Learning*, 40(3):243–263, 2000.

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[4] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[5] R. S. Sutton. Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, 3(1):9–44, 1988.

[6] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.

[7] G. Tesauro. TD-Gammon, a Self-teaching Backgammon Program, Achieves Master-level Play. *Neural Computation*, 6(2):215–219, 1994.