Warsztaty badawcze 2

Jakub Lange, Aleks Kapich, Michał Matejczuk, Paweł Świderski, Mateusz Kubita

6 czerwca 2024

1 Wstęp - Reinforcement Learning

Niniejszy raport skupia się na zastosowaniu algorytmów Reinforcement Learning do trenowania modeli na różnych poziomach gry Doom. Naszym celem jest eksploracja możliwości uczenia ze wzmocnieniem (RL) w dynamicznym środowisku gry komputerowej z użyciem biblioteki VizDoom [2] i pomocą środowiska OpenAI Gym. Przedstawimy podejście do doboru hiperparametrów i nagrody wyjaśniając nasze wybory na podstawie wcześniejszych badań i doświadczeń. Omówimy również napotkane trudności oraz przedstawimy rozwiązania, które zastosowaliśmy, aby je przezwyciężyć. Zaprezentujemy także wyniki naszych eksperymentów i podejście programistyczne, dokonując analizy skuteczności i efektywności każdego z badanych algorytmów w kontekście trenowania modelu do gry w Doom. Celem tego projektu było wytrenowanie agenta, który jest w stanie ukończyć dany poziom gry po wcześniejszej nauce.

2 Krótki opis gry - Doom

Doom jest grą science-fiction z kategorii strzelanek pierwszoosobowych wprowadzoną na rynek 10 grudnia 1993 roku [5]. Gracz wciela się w niej w rolę kosmicznego żołnierza przedzierającego się przez hordy wrogów w szeregu lokacji, różniących się od siebie konfiguracją oraz celami jakie postawione są przed graczem. Jest to jedna z wcześniejszych gier korzystających z grafiki 3D. Ze względu na swoją innowacyjność i wpływ na branżę gier komputerowych Doom zyskał status gry kultowej i jest uważany za jedną z najważniejszych gier w historii. Ze względu na swoją prostotę i łatwą dostępność gra stała się współcześnie również środowiskiem do trenowania modeli Reinforcement Learningu.

3 Tło teoretyczne algorytmów Reinforcement Learning

3.1 Proximal Policy Optimization

Proximal Policy Optimization (PPO) [4] to wynaleziony przez OpenAI w 2018 algorytm RL używany najczęściej do nauki modeli do grania w gry lub w robotyce, uważany niekiedy za rozwiązanie state-of-art. Cechuje się on prostotą, stabilnością oraz efektywnością próbkowania.

PPO jest metodą on-policy, ponieważ uczy się przez bezpośrednią interakcję ze środowiskiem przez wykonywanie akcji. Metody te polegają na użyciu dwóch sieci neuronowych: agenta zawierającego politykę podejmowania akcji

oraz sieci przewidującej wartość nagrody w danym czasie dla danego stanu. Uczenie polega na interakcji agenta ze środowiskiem, porównaniu otrzymanej nagrody do oczekiwanej i aktualizacji wag sieci w sposób faworyzujący działania, które przyniosły największą relatywną nagrodę. Cechą wyróżniającą PPO na tle podobnych algorytmów jak Trust Region Policy Optimization (TRPO) jest uproszczona funkcja celu w uczeniu agenta:

$$L^{\text{CLIP}}(\theta) = \hat{E}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{1}$$

gdzie:

- θ to parametr polityki,
- $\hat{E}_t$ oznacza empiryczną wartość oczekiwaną po krokach czasowych,
- r_t to stosunek prawdopodobieństwa między nową i starą polityką,
- $\hat{A}_t$ to oszacowana przewaga w czasie t (różnica między otrzymaną a przewidywaną nagrodą w tym czasie),
- ϵ to hiperparametr, zazwyczaj 0.1 lub 0.2.

Innowacją w uczeniu PPO jest zastosowanie obcięcia (clip) $r_t\hat{A}_t$, co niweluje problem niestabilnego, a co za tym idzie nieskutecznego uczenia. Dla wartości $r_t\hat{A}_t$ znacznie oddalonych od jedynki, czyli gdy dana akcja przynosząca przewagę $\hat{A}_t$ (dodatnią lub ujemną) była znacznie bardziej prawdopodobna w nowej polityce niż w starej, aktualizacja polityki za pomocą gradientu byłaby bardzo drastyczna. Jako, że model uczy się na podstawie egzekwowanej polityki, w przypadku gdy nowa polityka okazałaby się bardzo niepoprawna, model robiłby duży krok w tył który trudno byłoby mu naprawić. Obcięcie wartości $r_t\hat{A}_t$ do pewnego przedziału w okolicach 1 sprawia, że aktualizacja polityki nie będzie zbyt duża zwiększając tym samym stabilność uczenia.

Ostateczną funkcją celu dla calego modelu jest:

$$L_t^{PPO}(\theta) = \hat{E}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t) \right] \tag{2}$$

gdzie:

- $L_t^{VF}(\theta)$ jest funkcją straty w sieci szacującej nagrodę,
- $S[\pi_\theta](s_t)$ odpowiada za entropię,
- c_1, c_2 są hiperparametrami.

Maksymalizacja entropii sprawia, że zwłaszcza na początku uczenia akcje wykonywane przez agenta będą bardziej nieprzewidywalne, co skutkować będzie szeroką eksploracją.

3.2 Advantage Actor Critic

Advantage Actor Critic (A2C) [3] jest algorytmem uczenia polegającym na połączeniu koncepcji uczenia bazującego na wartości i bazującego na polityce. W metodach actor-critic uczymy dwie sieci: aktora, czyli funkcję polityki

odpowiedzialną za wskazanie akcji oraz krytyka, funkcję wartości, który ocenia akcję wykonaną przez aktora. W trakcie jednej iteracji algorytmu ze stanu s_t najpierw aktor wskazuję akcję do wykonania a_t, następnie krytyk zwraca Q-wartość $q(s_t, a_t)$, potem akcja jest wykonywana, a więc otrzymujemy stan s_{t+1} oraz nagrodę R_{t+1}. Następnie aktualizowane są wagi aktora na podstawie otrzymanej Q-wartości $q(s_t, a_t)$, po czym aktor znów wskazuje akcję a_{t+1} i na podstawie $q(s_t, a_t)$, $q(s_{t+1}, a_{t+1})$ oraz tzw. błędu TD aktualizowane są wagi krytyka.

Metoda A2C charakteryzuje się funkcją przewagi zastępującą funkcję wartości Q:

$$A(s, a) = Q(s, a) - V(s) \tag{3}$$

gdzie $V(s)$ jest estymacją średniej nagrody dla ruchów wychodzących z s. Oznacza to, że podobnie jak w PPO bedziemy faworyzować trajektorie, które dały nam nagrodę wyższą niż oczekiwana. Unikamy również obliczania osobno $Q(s, a)$ i średniej nagrody dla danego stanu $V(s)$, zamiast tego estymując ich różnicę za pomocą błędu TD (Temporal Difference), otrzymując ostatecznie:

$$A(s, a) = R_{t+1} + \gamma V(s_{t+1}) - V(s) \tag{4}$$

gdzie γ jest parametrem odpowiadającym za zmniejszanie istotności nagrody otrzymanej później w czasie.

4 Scenariusz Basic

Scenariusz Basic jest podstawowym, bardzo okrojonym scenariuszem polegającym za zabiciu jednego potwora. Gracz znajduje się w pokoju i od razu widzi potwora pod ścianą (Rysunek 1).

Rysunek 1: Zrzut ekranu z gry scenariusza Basic

Potwór pojawia się w losowym miejscu na szerokości ściany, pozostaje nieruchomy przez cały czas rozgrywki i nie atakuje gracza. Agent ma do dyspozycji jedynie 3 akcje: move_left, move_right oraz attack, jego zadaniem jest więc przesunąć się w prawo lub w lewo tak, aby być naprzeciwko potwora a następnie do niego strzelić. Gra kończy się gdy gracz zabije potwora lub gdy minie 300 ticów, czyli najmniejszych odstępów czasu między akcjami w grze.

Do trenowania modelu użyliśmy metody PPO. Funkcja nagrody wyglądała następująco:

Funkcja nagrody

$$\begin{aligned}\textbf{Reward} = &\text{living\_reward}\\ &+ \text{kill\_reward}\\ &+ \text{ammo\_reward}\end{aligned}$$

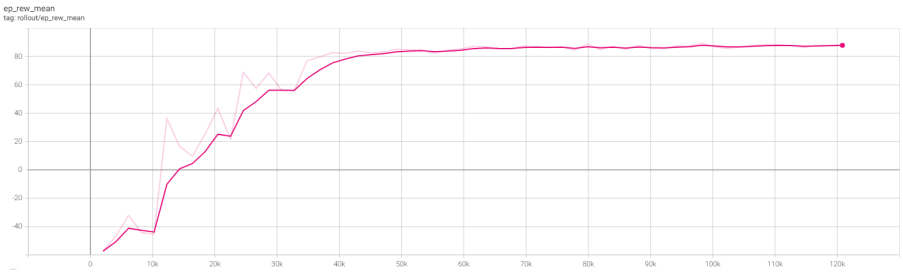
Gdzie:

- living_reward = kara wartości -1 naliczana co 1 tic, podczas gdy gracz żyje
- kill_reward = nagroda wartości 106 za zabicie potwora (ukończenie gry)
- ammo_reward = kara wartości -5 za wystrzelenie pocisku.

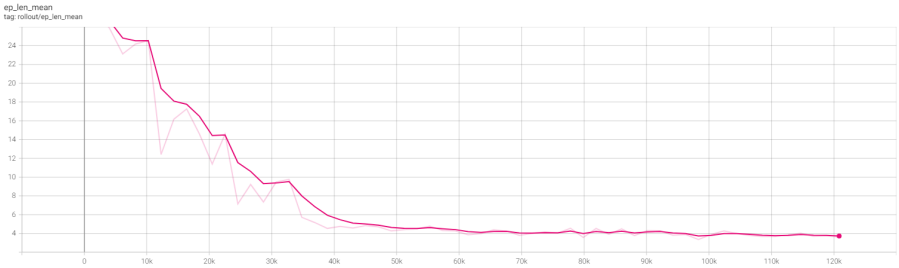
Kara za życie gracza (living_reward) nagradzała model za akcje prowadzące do jak najszybszego ukończenia poziomu, zaś kara za wystrzelenie pocisku motywowała model do niemarnowania amunicji, której gracz ma na początku 50.

Z pomocą biblioteki tensorboard byliśmy w stanie obserwować efekty uczenia. Zdecydowaliśmy się opisać niektóre z nich na przykładzie modelu wyuczonego na 120 tys. kroków.

Najważniejszymi obserwacjami z uczenia modelu są średnia wartość nagrody (Rysunek 2) oraz średnia długość gry (Rysunek 3). Pierwszy z tych wykresów pokazuje, że już po około 50 tys. kroków model osiągał średnią nagrodę w okolicach 85, co jest bardzo dobrym wynikiem, biorąc pod uwagę, że maksymalna wartość tej nagrody wynosi 101 i to przy najbardziej sprzyjającej losowości (potwór pojawia się od razu na przeciwko gracza i gracz strzela do niego w pierwszym możliwym ruchu, co daje nagrodę 106 za zabicie potwora i -5 za wystrzał bez kary czasowej). Wykres średniego czasu potrzebnego do ukończenia poziomu również wskazuje, że model po 50 tys. krokach uczenia kończy poziom bardzo szybko, bo już po około 4 ticach. Z racji, że operujemy na maksymalnej możliwej częstotliwości wyświetlania klatek równej 35 na sekundę, taki model kończy poziom po około 114 ms. Z wykresu czasu również wywnioskować, że już po około 2 tysiącach kroków model regularnie wygrywał, osiągając czas znacznie poniżej ustalonego limitu 300 ticów.

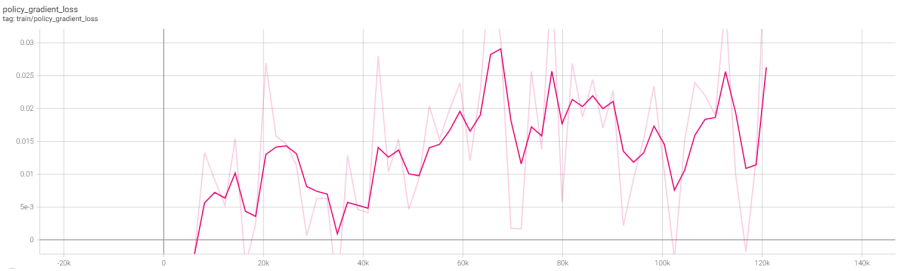


Rysunek 2: Średnia wartość nagrody podczas treningu scenariusza Basic w zależności od liczby kroków treningowych

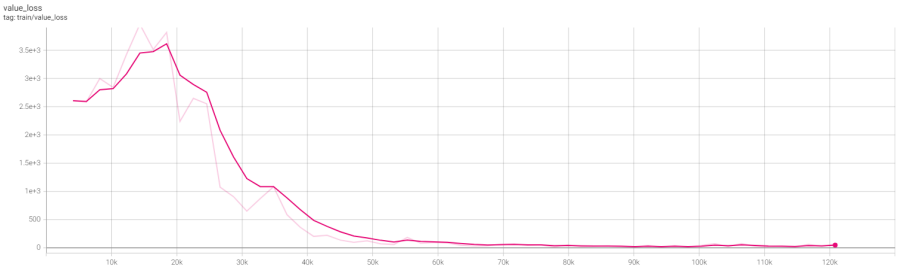


Rysunek 3: Średnia długość gry podczas treningu scenariusza Basic w zależności od liczby kroków treningowych

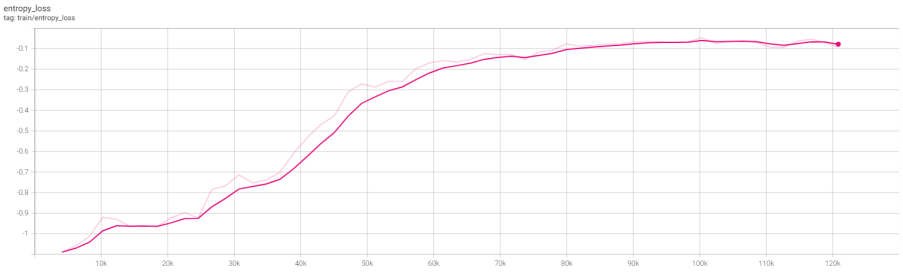
Powyższe wykresy wyraźnie wskazują 50 tys. kroków jako moment, w którym uczenie znacznie spowolniło. Możemy zbadać to zjawisko dokładniej dzięki wykresom poszczególnych funkcji straty w PPO (rysunki 4 - 7).



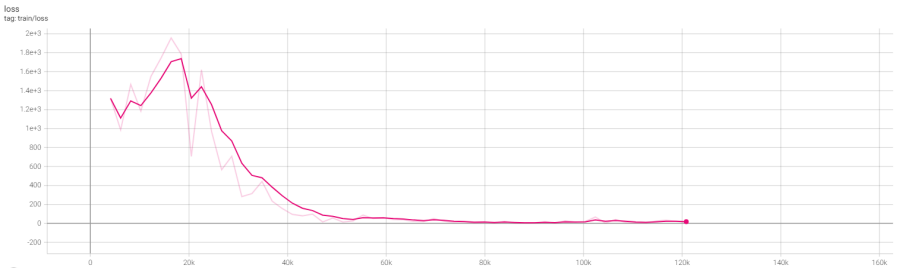
Rysunek 4: Wartość funkcji celu gradientu polityki (równanie 1) w zależności od liczby kroków treningowych



Rysunek 5: Wartość funkcji straty w sieci szacującej nagrodę w zależności od liczby kroków treningowych (SSE)



Rysunek 6: Wartość entropii w zależności od liczby kroków treningowych



Rysunek 7: Wartość funkcji celu (równanie 2) w zależności od liczby kroków treningowych

Zauważone wcześniej spowolnienie uczenia po około 50 tys. krokach jest wyraźnie widoczne na rysunku 5, czyli przy funkcji straty sieci przewidującej nagrodę. Oznacza to, że po tym czasie sieć zaczyna dokładnie przewidywać nagrodę za wykonanie danej akcji. W efekcie zmniejsza się wartość przewagi, co wpływa na funkcję straty polityki gradientu. Warto też zauważyć, że na początku uczenia wartość tej funkcji rosła - jest to spowodowane przez początkowe losowe działanie agenta, które prowadzi do niskich nagród, co zmienia się w miarę ulepszania polityki, za czym funkcja wartości musi nadążyć. Na rysunkach 6 oraz 7 również widzimy osiągnięcie i utrzymanie się niskiej bezwzględnej wartości funkcji - w pierwszym przypadku jest to entropia, która odpowiada za eksploracje (a co za tym idzie - większe zmiany polityki); w drugim całościowa funkcja straty, a więc główna funkcja używana w uczeniu. W obu przypadkach spadek wartości bezwzględnej funkcji jest pożądany - w miarę osiągania coraz mniejszej wartości funkcji straty, w tym optymalnej polityki, chcemy mniej "eksperymentować". Jeśli chodzi o wykres 4 jest on jedynie estymacją gradientu polityki i nie jest właściwie miarą jakości uczenia modelu, a służy jedynie do aktualizacji polityki.

5 Scenariusz Defend The Center

Rysunek 8: Zrzut ekranu z gry scneariusza Defend The Center

W tym scenariuszu agent umieszczony jest wewnątrz koła (rysunek 8), gdzie z każdej strony zbliżają się do niego wrogowie. Stanowi to różnicę względem poprzedniego scenariusza Basic, gdzie występował wyłącznie statyczny przeciwnik. Podobnie jednak jak poprzednio, agent może przesuwać celownik w lewo, w prawo bądź oddawać strzał. Zadaniem agenta jest wyeliminowanie jak największej liczby przeciwników zanim sam zostanie wyeliminowany. Jest to nieuchronne, ponieważ podobnie jak wcześniej, agent dysponuje ograniczoną liczbą kul, zatem istotne jest by zarządzał nimi w sposób rozważny i strzelał wyłącznie, gdy z pewnością trafi przeciwnika - jedno trafienie eliminuje potwora, co przekłada się na +1 do funkcji nagrody.

Funkcja nagrody

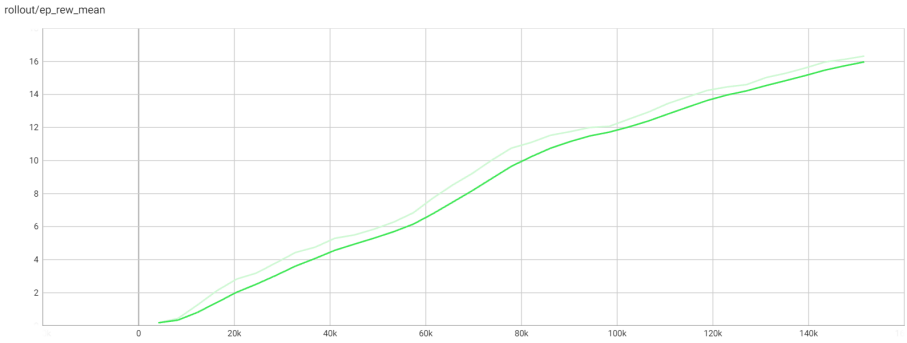
$$\textbf{Reward} = \text{kill\_reward} + \text{death\_penalty}$$

Jej postać nie jest skomplikowana

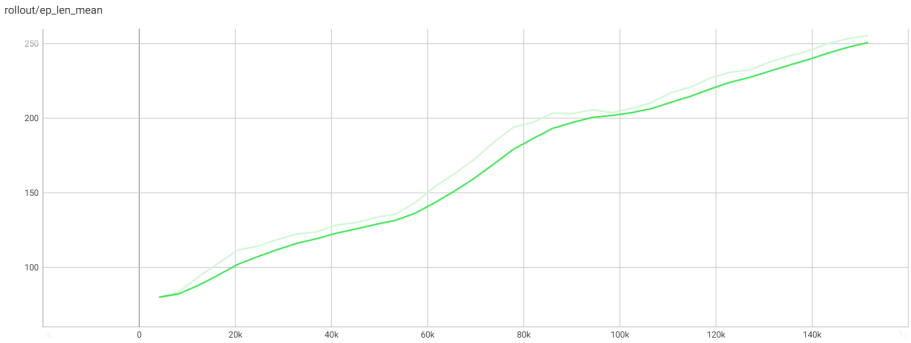
- kill_reward = nagroda równa 1 przyznawana za zabicie jednego przeciwnika
- death_penalty = kara równa -1 za przegraną rozgrywkę

Korzystając z algorytmu PPO po uczeniu trwającym 150 tysięcy kroków udało się osiągnąć satysfakcjonujące wyniki, bowiem agent nauczył się identyfikacji potworów oraz w większości przypadków nie marnował kul oddając celne strzały. Analizując wykresy średniej wartości nagrody oraz średniej długości rozgrywki na przestrzeni treningu (rysunki 9 i 10) widać między nimi bardzo silną zależność - agent zanim nauczył się rozpoznawać wrogów był

przez nich bardzo szybko eliminowany. Dopiero z czasem radził sobie z trzymaniem przeciwników na dystans przez dłuższy okres oraz oddawał mniej zmarnowanych strzałów. Dłuższy okres utrzymania przy życiu naturalnie umożliwiał eliminację większej liczby rywali i zwiększenie wartości nagrody. Sposób działania wytrenowanego agenta zdecydowanie nie przypomina rozgrywki prowadzonej przez człowieka, ponieważ bardzo szybko porusza on w kółko celownikiem i w większości wypadków oddaje strzał błyskawicznie w chwili natrafienia na wroga - uznaliśmy to za sukces, bowiem osiągi modelu są lepsze niż te Michała Jana Matejczuka. Niemniej porównując wyniki (tabela 1) warto mieć na uwadze, że trening agenta RL przebiegał znacznie dłużej.



Rysunek 9: Średnia wartość nagrody podczas treningu scenariusza Defend The Center w zależności od liczby kroków treningowych



Rysunek 10: Średnia długość gry podczas treningu scenariusza Defend The Center w zależności od liczby kroków treningowych

Gracz	**Średnia wartość nagrody**	**Liczba kroków treningowych**
Agent RL	16	150 000
Michał Jan Matejczuk	7	10

Tabela 1: Rezultaty dla poszczególnych graczy - scenariusz Defend The Center

6 Scenariusz Deadly Corridor

Kolejnym scenariuszem w grze Doom, na którym trenowaliśmy nasz model RL był scenariusz Deadly Corridor. W Deadly Corridor gracz znajduje się w wąskich korytarzach i ma na celu dotarcie do nagrody, unikając przy

tym wyeliminowania przez wrogów. Gracz spotyka po drodze 6 potworów, które może, ale nie musi zabić. Potwory jednak w przeciwieństwie do scenariusza Basic mogą zadawać obrażenia graczowi. Zrzut ekranu z Deadly Corridor przedstawia rysunek 11.

Rysunek 11: Zrzut ekranu ze scenariusza Deadly Corridor

Kluczową różnicą pomiędzy Deadly Corridor a poprzednim scenariuszem jest to, że gracz ma do wyboru aż 7 ruchów: move_left, move_right, attack, move_forward, move_backward, turn_left, turn_right.

Nowy scenariusz wymaga modyfikacji w funkcji nagrody dla agenta. Z tego względu eksperymentowaliśmy z różnymi metodami. Oczywiście celem gracza jest dostać się na koniec korytarza w jak najkrótszym czasie. Nie jest to jednak możliwe bez zabicia demonów, gdyż one zabiją gracza wcześniej. Dlatego też de facto celem jest: zabicie demonów tracąc jak najmniej życia oraz przejście przez korytarz do nagrody. Opracowana dla tego scenariusza funkcja nagrody wygląda następująco:

Funkcja nagrody

$$\begin{aligned}\textbf{Reward} = {} & \text{living\_reward} \\ & + \text{movement\_reward} \\ & + \text{damage\_taken\_delta} \times 10 \\ & + \text{hitcount\_delta} \times 210 \\ & + \text{ammo\_delta} \times 5 \\ & + \text{camera\_reward}\end{aligned}$$

Gdzie:

- living_reward = kara za czas gry
- movement_reward = nagroda za zbliżanie się do końca korytarza
- damage_taken_delta = (% życia w aktualnym kroku) - (% życia w poprzednim kroku)
- hitcount_delta = (Suma obrażeń zadanych w aktualnym kroku) - (Suma obrażeń zadanych w poprzednim kroku)
- ammo_delta = (Amunicja w aktualnym kroku) - (Amunicja w poprzednim kroku)
- camera_reward = kara za odwracanie się tyłem agenta (odwracanie skutkowało ustawieniem się plecami do potworów oraz końca korytarza)

Agent jest więc karany za czas w którym nie ukończył poziomu, otrzymywanie obrażeń, zużycie amunicji i odwracanie się o zbyt duży kąt, a nagradzany gdy porusza się w stronę końca korytarza oraz gdy udaje mu się zabijać demony.

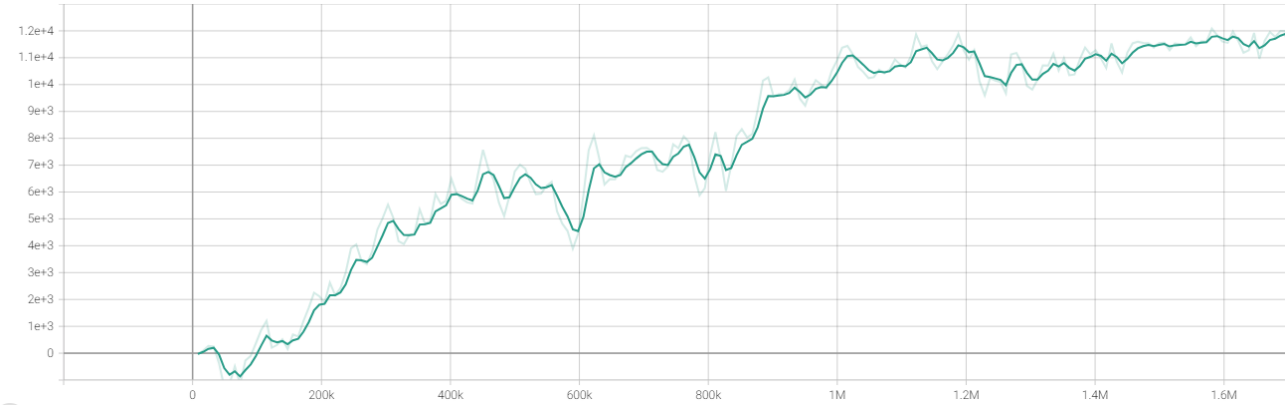
Zdecydowaliśmy się karać agenta za odwracanie się, gdyż zauważyliśmy że czasami odwraca się do tyłu co okazywało się być bardzo złą decyzją. Wagi nagród oraz kar zostały ustalone metodą heurystyczną.

Do trenowania stosowaliśmy algorytmy PPO oraz A2C, jednak to efekty osiągnięte pierwszym ze sposobów początkowo przynosiły lepsze efekty, więc w późniejszej fazie skupiliśmy się na dotrenowaniu pierwszego z modeli. Ze względu na ograniczoną moc obliczeniową nie byliśmy w stanie przetestować zbyt dużej liczby kombinacji hiperparametrów i wag funkcji nagrody. Niemniej jednak obserwując uczenie algorytmu i decyzje agenta dostrajaliśmy niektóre hiperparametry (np. uznaliśmy za bardzo ważne zabijanie demonów w pierwszej kolejności, gdyż skutkuje to mniejszą karą za otrzymywanie obrażeń i zużycie amunicji w przyszłości).

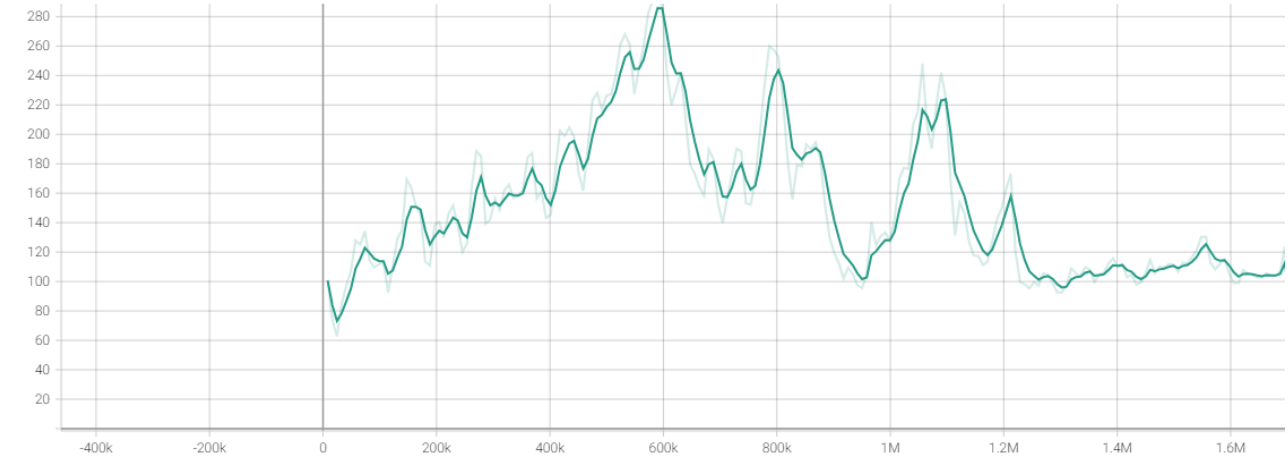
Model Reinforcement Learning w Doom: Deadly Corridor

Model został przeszkolony na scenariuszu Deadly Corridor w grze Doom. Po **1.7 milionach kroków treningowych**, osiągnął **średnią nagrodę na poziomie 12000**. Cały proces treningu trwał **27 godzin**.

Rysunki 12 i 13 przedstawiają średnią wartość nagrody oraz średni czas ukończenia poziomu w zależności od liczby kroków uczenia modelu.



Rysunek 12: Średnia wartość nagrody agenta podczas treningu scenariusza Deadly Corridor w zależności od liczby kroków treningowych



Rysunek 13: Średnia długość gry podczas treningu scenariusza Deadly Corridor w zależności od liczby kroków treningowych

Możemy zaobserwować że agent dość dobrze się uczy i osiąga średnią wartość nagrody wysokości około 12 tysięcy. Jest to oczywiście wartość relatywna, która nie mówi nam wiele o zachowaniu agenta. Rzeczywiste zachowanie agenta można zweryfikować empirycznie poprzez obserwacje zachowań modelu w środowisku gry na żywo. Film z takiej obserwacji jest przedstawiony poniżej.

Zapis obrazu podczas gry - Deadly Corridor

Zamieszczony tutaj materiał przedstawia, jak nauczony agent przechodzi scenariusz.

Na rysunku 12 widzimy, że agent najlepiej radził sobie po około 1.5 mln kroków. Można zauważyć, że wtedy rozgrywka trwała około 150 klatek, a więc niecałe 4,5 sekundy.

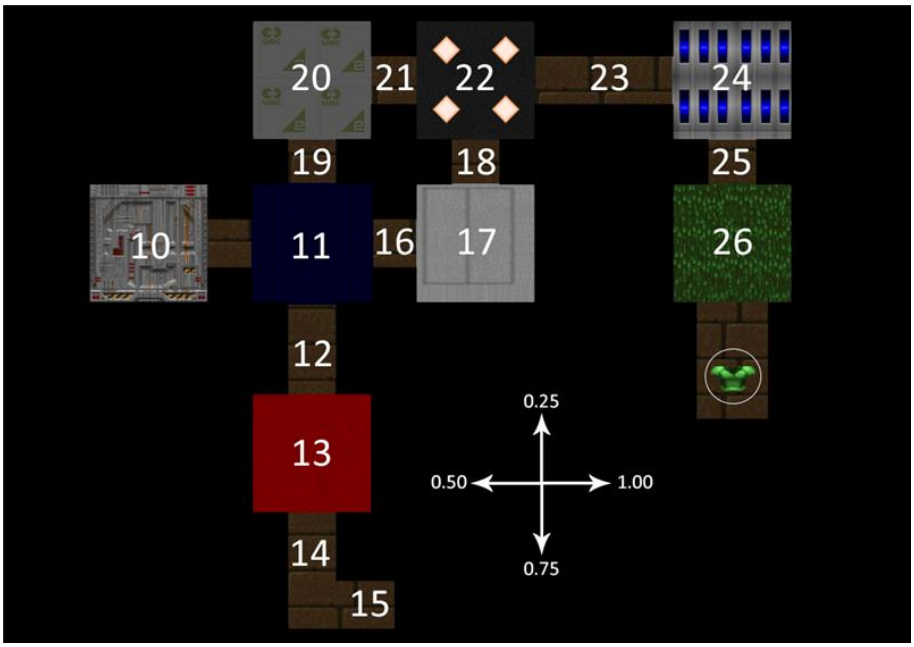
Podczas testowania tego modelu po 1.5 miliona kroków treningowych agent był w stanie zabić większość demonów i dojść do końca korytarza. Co ciekawe, zdarza się że demony upuszczają swoją broń gdy zginą. Nasz model nauczył się, że to jest dobra okazja na ulepszenie swojego uzbrojenia (gdyż broń podstawowa jest słabsza niż te od

demonów) więc często podnosił broń, co poprawiało jego wyniki. Dzięki modyfikacjom funkcji nagrody agent szedł prosto przed siebie, korzystając z obrotu kamery w celu zabijania demonów.

7 Scenariusz My Way Home

Scenariusz My Way Home jest wyjątkowy ze względu na to, że na mapie nie pojawiają się żadne potwory. Celem gracza jest odnalezienie w labiryncie konkretnego pomieszczenia, w którym znajduje się zielona kamizelka.

Jak widać, do docelowego pomieszczenia prowadzi korytarz o zielonych ścianach.



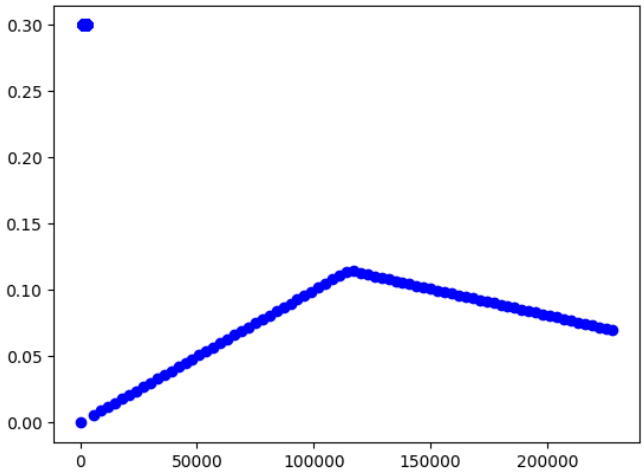
Rysunek 14: Schemat labiryntu - źródło

Zadanie jest utrudnione, ponieważ gracz za każdym razem spawnuje się w innym miejscu oraz jest obrócony w innym kierunku.

Najprostszym rozwiązaniem zadania wydaje się być nagradzanie modelu za zbliżanie postaci do określonych koordynatów, w których znajduje się pokój, ale takie rozwiązanie jest wadliwe ze względu na brak jego ogólności (gdyby docelowy pokój zmienił swoją pozycję, to model nie byłby w stanie go odnaleźć).

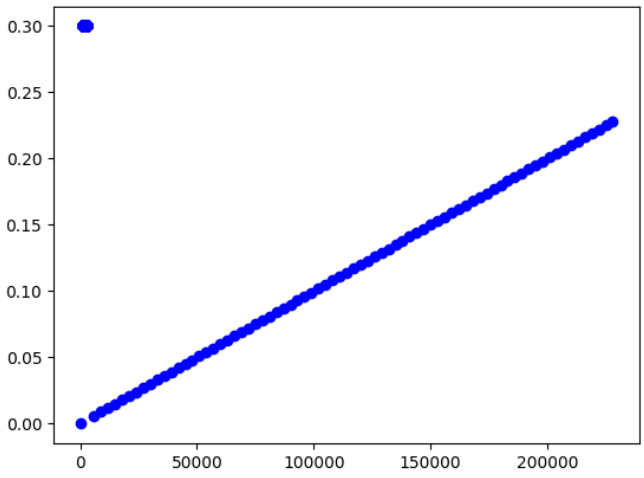
Aby zachęcić agenta do aktywnych poszukiwań, postanowiliśmy dać modelowi nagrodę za różnicę przemieszczenia. Model z funkcją nagrody zależną od delty przemieszczenia po wytrenowaniu faktycznie szybko się przemieszczał, ale niestety często ruch odbywał się w kółko po jednym / dwóch pomieszczeniach. Sama nagroda za ruch to zbyt mało. Premiowany powinien być przede wszystkim ruch w kierunku wyjścia. Dlatego kolejnym elementem funkcji nagrody była premia za utrzymywanie zielonego korytarza w polu widzenia - do nagrody była dodawana ilość zielonych pikseli na ekranie pomnożona przez stałą. Model z taką funkcją nagrody miał wysoką w stosunku do poprzedniego zdolność odszukiwania zielonego korytarza, ale niestety zamiast podążać nim w kierunku kamizelki, "przyklejał"się do ściany korytarza, żeby otrzymywać nagrodę za zielone piksele w polu widzenia. Aby rozwiązać ten problem, nagrodę za utrzymywanie korytarza w polu widzenia wyraziliśmy za pomocą funkcji:

$f : [0; 230000] -> [0, 0.1]$
której wykres wygląda następująco:



Rysunek 15: Pierwsza wersja funkcji nagrody za zielone piksele

Wyraża ona następującą heurystykę: jeśli ilość zielonych pikseli jest w przedziale [1000; 3000], to patrzymy bezpośrednio na kamizelkę - wystarczy iść do przodu i gra zakończy się. W pozostałych przypadkach chcielibyśmy, żeby zielone piksele zajmowały około połowę ekranu (jeśli zajmują więcej, najprawdopodobniej agent patrzy wprost na zieloną ścianę).
Niestety tak zdefiniowana funkcja nie sprawdziła się. Agent regularnie cofał się do miejsca oznaczonego na obrazku liczbą 25. Jest to spowodowane tym, że oprócz okolic korytarza finalnego, jest to drugie miejsce, z którego patrząc na wprost ilość zielonych pikseli na ekranie to około połowa.
Z tego powodu funkcję zmieniliśmy na następującą:



Rysunek 16: Druga wersja funkcji nagrody za zielone piksele

Ostatecznie nagroda jest wyrażona jako:

$$reward = 10^{-4} + 0.005 * \Delta_{dist} + greenReward + stuckPenalty + I$$

$$I = \begin{cases} 1 & znalezionio\ wyjscie \\ 0 & nie\ znaleziono\ wyjscia \end{cases}$$

7.1 My way home - wyniki

Model nawet po wielogodzinnym treningu nie był w stanie odnaleźć ścieżki do wyjścia w ogólnym przypadku. Radził sobie zadowalająco tylko wtedy, gdy gracz spawnował się relatywnie blisko wyjścia (w pomieszczeniu nr 11 lub bliżej).

8 Podsumowanie

Wytrenowaliśmy modele Reinforcement Learning na 4 różnych poziomach gry Doom: Basic, Defend the Center, Deadly Corridor oraz My Way Home. Każdy poziom posłużył nam do zbadania innego aspektu uczenia. Przy scenariuszu Basic obserwowaliśmy procesy i wartości funkcji ewaluacji w trakcie uczenia na przykładzie prostego poziomu o ograniczonej liczbie zmiennych i dostępnych akcji. W Defend The Center głównym celem było nauczenie agenta działania w trochę bardziej skomplikowanym środowisku i reagowania na zagrożenie ze strony przeciwników podejmujących aktywne próby przedwczesnego zakończenia rozgrywki. W Deadly Corridor znacznie rozszerzyliśmy model, umożliwiając agentowi więcej akcji oraz rozbudowując funkcję nagrody o szereg dostępnych informacji, trenując go przez ponad dobę w celu uzyskania zadowalających rezultatów. W My Way Home wykorzystaliśmy z kolei doświadczenie zdobyte w Deadly Corridor i połączyliśmy je z niestandardowym podejściem do funkcji nagrody i doskonaleniem jej na podstawie obserwacji zachowań modelu. Możliwą kontynuacją eksperymentów z

Doomem mogłyby być próby wytrenowania modelu, który wykazywałby dobry poziom skuteczności na więcej niż jednym poziomie w celu zbadania uniwersalności zachowań w grze, jednak czasochłonność treningu i złożoność tego zagadnienia nie pozwoliły nam zrealizować ich w ramach tego projektu. Repozytorium z implementacją projektu znajduje się tutaj.[1]

Bibliografia

[1] Aleks Kapich i in. *Reinforcement Learning Doom*. Dostęp: 2024-06-05. URL: https://github.com/AKapich/Reinforcement_Learning_Doom.

[2] Michał Kempka i in. „ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning". arXiv: 1605.02097. URL: https://arxiv.org/abs/1605.02097.

[3] Volodymyr Mnih i in. „Asynchronous Methods for Deep Reinforcement Learning". arXiv: 1602.01783. URL: https://arxiv.org/abs/1602.01783.

[4] John Schulman i in. „Proximal Policy Optimization Algorithms". arXiv: 1707.06347. URL: https://arxiv.org/abs/1707.06347.

[5] Andrew Webster. *'Doom' at 20: John Carmack's hellspawn changed gaming forever*. Dostęp: 2024-06-05. URL: https://www.theverge.com/2013/12/10/5195508/doom-20th-anniversary-retrospective.