# GAN: Rozpoznawanie gracza–bot'a

**Witold Kupś 127088**

**Z uwagi na coraz większy postęp w dziedzinie sztucznej inteligencji na podłożu gier komputerowych oraz zauważalną skuteczność takich rozwiązań postanowiono podjąć próbę odróżnienia sytuacji, w których grę prowadzi Sztuczna Inteligencja od tych, w których gra człowiek. Jako potencjalny środek do rozwiązania problemu wykorzystano *Generative Adversarial Network (GAN)*, natomiast jako przykład modelu gracza opartego na Sztucznej Inteligencji (AI) - modelu uczonego za pomocą *Q-learningu*, jednak inicjalizowanego próbkami rzeczywistej gry (*Imitation Learning*).**

**GAN | VizDoom | Q-Learning | Imitation learning | Deep Neural Network**
**Korespondencja:** ***witold.kups@student.put.poznan.pl***

## Wprowadzenie

Na przestrzeni ostatnich lat widać wzmożone wysiłki w obszarze rozwoju i wykorzystania sztucznej inteligencji. Objawiają się one również w przestrzeni gier. Początkowo wysiłki skupione były na grach turowych czy planszowych jak *szachy* czy *go*, jednak przeniosły się one na te pochodzenia czysto komputerowego, zarówno o mniejszej złożoności ze względu na w pełni obserwowalne środowisko, jak *Space Invaders*, *Pacman*, ale i te o częściowo obserwowalnym środowisku sprawiające dużo większe wyzwanie – zaledwie kilkoma przykładami mogą być *Starcraft II*(1) czy *Doom*(2). Powstałe silniki okazują się często skuteczniejsze nawet od najlepszych graczy, co tworzy potrzebę odróżnienia takich przypadków, aby zapobiec nierównej rywalizacji, a z czasem jej całkowitemu zaniku.
Do osiągnięcia celu można wykorzystać model *GAN*(3) bazujący na uczeniu sieci poprzez koordynowanie jej tworów tak, by wyglądały na rzeczywiste. Kluczowym tutaj składnikiem jest istnienie weryfikatora mówiącego o prawdziwości danego obrazu. Można to przenieść na płaszczyznę gier – w niej to generator produkowałby sekwencję ruchów podlegającą weryfikacji, czy jest wynikiem działań człowieka, czy też sieci.

## Prace powiązane

W dziedzinie uczenia ze wzmocnieniem przy wykorzystaniu *GAN* zostało opublikowanych niewiele prac – większość skupia się na bezpośrednim przetwarzaniu obrazu(4, 5). Najnowszą jest próba utworzenia sieci dążącej do samopoprawy rezultatów bazując właśnie na modelu GAN(6) – daje ona poprawę wyników względem powszechnych rozwiązań (DQN, A3C, REINFORCE), co stwarza nadzieje do uzyskania pozytywnych rezultatów.
Innymi przykładami może być utworzenie modelu mogącego w sposób autonomiczny prowadzić samochód(7) bazując na próbkach z gry czy też polepszenie istniejącego modelu algorytmu Q-learning(8). Niestety, kod źródłowy żadnego z nich nie został opublikowany. Co więcej, do ostatniego z nich została podjęta próba replikacji(9) w wyniku której uzyskano **znacznie gorsze** rezultaty, niż te prezentowane w artykule.
Pod znakiem zapytania staje więc efektywność takiego podejścia, gdyż nawet w pracy oryginalnej zadowalające rezultaty osiągane były znacznie szybciej z wykorzystaniem pierwotnego rozwiązania.

## Możliwe problemy i zagrożenia

Głównym problemem w realizacji projektu jest odróżnienie następujących scenariuszy:

- Gracz początkujący a słaby model
- Bardzo dobry gracz a wyuczony model
- Gracz losowy a model bazujący na *random*
- Gracz grający bez zamiaru wygrania (*Troll*) a błędnie skonstruowana sieć

Problematyczne jest również nałożenie ograniczeń na model tak, by *fizycznie* mógł konkurować z człowiekiem. Z drugiej strony kluczowe jest również umożliwienie sieci zrozumienie środowiska oraz celu samej gry.
W projekcie zostanie zastosowany Tensorflow 2, którego struktura uległa diametralnej zmianie – implementacja może stworzyć dodatkowe problemy skutkujące w opóźnieniu oraz ewentualnych innych trudnościach.
Problematyczne jest również dostarczenie zasobów; rozwiązaniem jest wykorzystanie środowiska Colab, jednakże odbiór interaktywnych rezultatów w postaci wyników rozgrywek/filmów jest w nim utrudniony.

## Wymagania minimalne

- Jednym z założeń jest wykorzystanie próbek rzeczywistych – należy skolekcjonować reprezentatywne próbki rozgrywek w różnych scenariuszach
- Budowa projektu i jego integracja w środowisku Colab wraz z możliwością odczytu/zapisu próbek oraz modelu
- Budowa przykładowego modelu sieci potrafiącego grać w określonym scenariuszu (dalej – dowolnym) w Doom (platforma VizDoom).
- Stworzenie weryfikatora mogącego z określonym prawdopodobieństwem (większym od losowego) stwierdzić, czy gracz jest rzeczywisty, czy też jest on botem.

## Napotkane problemy

**Colab.** Jedne z największych trudności wynikały z zastosowania Colab'a, który pomimo przenośności, szybkości oraz darmowych zasobów, niestety niesie ze sobą:

- **Słabe i zawodne podpowiadanie składni** Środowisko oferuje lepsze możliwości, niż przeciętny notatnik poprzez podpowiedź składni czy dostęp do dokumentacji. Wciąż jednak daleko mu do PyCharm'a. Sam silnik jest jednak taki sam, jak ten zastosowany w Visual Studio Code – brakuje w nim jednak pluginów tam szeroko dostępnych.
- **Słabe typowanie oraz komórki notatnika** Zastosowana architektura składa się na późne wykrywanie błędów, często występujących późno w kodzie lub nawet na końcu przetwarzania. Schemat komórkowy teoretycznie mógłby złagodzić sytuację pozwalając na częste wykonywanie kodu, jednak nie sprawdza się on w przypadku większych bloków kodu, których podział mógłby przynieść jeszcze większą złożoność. Co więcej, takie rozwiązanie oraz jego dynamika praktycznie uniemożliwiają statyczną analizę kodu, której brak w Colab'ie.
- **Częste rozłączanie środowiska** Colab jest środowiskiem darmowym; ma to swoje konsekwencje w postaci maksymalnego czasu przetwarzania wynoszącego 12 godzin, a także możliwymi rozłączeniami, a tym samym możliwym utraceniu kontekstu i wyników przetwarzania. Rozwiązaniem ostatniego problemu jest podpięcie Google Drive, jednak rodzi to kolejny problem w postaci każdorazowego podania kodu autoryzującego do dysku.
- **Zasoby** Środowisko oferuje dostęp do wydajnych kart graficznych oraz jednostek TPU, jednak są one ograniczone. Co więcej, priorytet ich przydziału mają zadania krótkotrwałe; dla pozostałych oferowane jest CPU. Rozwiązanie takie, oraz długotrwałość prowadzonych badań skutkowało praktyczną koniecznością wykorzystania CPU na skutek odmowy przydziału GPU, co było bardzo zauważalne w prędkości obliczeń, których czas wzrósł około 5x.
- **Integracja** Zastosowane środowisko webowe wiąże się z ograniczonymi możliwościami interakcji – przykładowo brak dostępu do dysku czy wyświetlacza. Utrudnia to odbiór wyników; nie ma możliwości wykorzystania rozwiązań istniejących w GYM(10) czy VizDoom aby 'na żywo' podejrzeć grę agenta; należy nagrać film lub gif (takie rozwiązanie przyjęto) utworzony z przetwarzanych klatek.
  Często występuje również problem z buforowaniem wyników, które znikają, bądź przeciwnie – nakładają się. W celu wskazania postępu wykorzystano pasek z pakietu tqdm(11), który na skutek zatrzymania oraz ponownego uruchomieniu komórki 'tracił' możliwość nadpisania obecnej linii; Colab przy każdym restarcie jakby dodawał nową linię do nowego wyjścia.

**VizDoom.** Projekt, jak wspomniano wcześniej, został wykonany w środowisku *VizDoom*, oferującym możliwość uczenia agentów gry Doom. Można wskazać kilka zalet, m.in.

- **Proste API** Opisane, dostępne w kilku językach programistycznych, przeważnie pomiędzy nimi spójne
- **Dokumentacja** Środowisko oferuje dokumentację dla oferowanych funkcji, poradnik typu *Quick Start* oraz kilka implementacji pokazowych opartych na wiodących framework'ach uczenia maszynowego.
- **Łatwość wdrożenia** Aby zacząć, wystarczy postępować według krótkiego poradnika *Quick Start*.
- *Popularność* Jest ono wykorzystane w wielu pracach badawczych oraz poradnikach, co często upraszcza rozwiązywanie problemów.

Niestety, o wadach również należy wspomnieć

- **Długi czas instalacji** Środowisko jest dosyć złożone, co ma konsekwencje w czasie jego instalacji wynoszącym około 10 minut. Należy zauważyć, że Colab jest wyposażony w standardowe pakiety, jednak VizDoom się do nich nie zalicza, co skutkuje w każdorazowej konieczności instalacji.
- **Obsługa błędów** Występujące w kodzie błędy często są przyjmowane przez system bezkrytycznie, skutkując błędnymi wynikami lub zacięciem się środowiska. Takie sytuacje napotkano m.in. w przypadku dodawania akcji do scenariusza, gdzie ich kolejność była zależna od istniejących, czy później próbie odtworzenia scenariusza z takimi akcjami skutkującej zacięciem systemu.

**Brak wiedzy.** Dużo problemów wynikło bezpośrednio z braku ugruntowanej wiedzy z obszaru Q-learning, Uczenia Maszynowego czy Tensorflow (w szczególności jego drugiej wersji i modelu grafowym). Skutkowało to licznymi próbami doświadczalnymi sprowadzającymi się do weryfikacji poprawności algorytmu, co w wykorzystanym środowisku było czasochłonne oraz często niejednoznaczne.

## Przeprowadzone badania

Badania planowane były na 4 etapy:

- Przygotowanie rozgrywek przez gracza rzeczywistego
- Nauka sieci na kilku scenariuszach
- Próba gry na nieznanym scenariuszu bazując na wcześniej nauczonych modelach
- Zastosowanie GAN'a weryfikującego realność gracza, a jako efekt uboczny w miarę możliwości – ulepszenie sieci docelowej.

Prowadzone były przez okres 2 miesięcy, głównie w obszarze przygotowania modelu sieci oraz uczenia na wybranych scenariuszach takich jak zbieranie apteczek (wersja basic), obrona środka, scenariusz podstawowy (jak najszybsze ustrzelenie celu) czy *deadly corridor* polegający na przejściu przez korytarz, w którym napotykamy wrogów.
Każdy scenariusz ma inna specyfikę, środowisko oraz cel, również inne zdefiniowane akcje. Do celów szkolenia konieczne było ich usystematyzowanie – dozwolone zostały ruchy w 4 kierunkach, obracanie oraz strzał.
Początkowe próby nauki spoczęły na niczym – konieczne było dostosowanie nagrody zarówno globalnie, jak i zależnie od scenariusza. Mimo to, nauczenie końcowego agenta w środowisku *deadly corridor* zajęło ponad dwa dni; łącznie ponad 100 epok po 500 epizodów, każda trwająca od 2 to 15 sekund zależnie od zaawansowania agenta. Ciekawym jednak może być obserwacja strategii przyjmowanych przez agenta w procesie kształcenia modelu jego nauki. Jednymi z przykładów jest próba biegu między wrogami, zabicie jednego a ominięcie drugiego, obracanie, czy wręcz wycofywanie się – nawet w wersji nagrody, która kara za takie zachowanie. Ostatecznie jednak agent potrafi trafić i pokonać wroga ledwo widocznego zza krawędzi.
Proces został zapoczątkowany próbami wykorzystania *imitation learning* bazując na nagranych rozgrywkach, jednak ze względu na niezdefiniowane na tamtym etapie błędy nie został one kontynuowany. Ostatecznie po ich wyeliminowaniu próbowano porównać efekty zastosowania wstępnego uczenia na 40 grach oraz uczenia zwykłego wykorzystując scenariusz "obrona linii"*(ang. defend the line)*; rezultat jednak nie przyniósł spodziewanych efektu, jakim było przyśpieszenie uczenia, czy zwiększenie nagrody na etapie pierwszych epok. Możliwe, że wynikało to z braku synchronizacji akcji podjętych, a tych odebranych przez sieć – ta podejmowała akcje co ustaloną liczbę klatek.
Przechodząc do samego schematu uczenia – wykorzystano agenta bazującego na *Double Q-learning* i zastosowaniu pamięci, natomiast sam stan reprezentowany był w postaci 4 następujących po sobie ramek (odstęp czasu zależny od liczby pomijanych ramek – parametr środowiska). Sam model sieci składał się z 3 sekwencji kombinacji warstw konwolucyjnych, batch norm., dropout oraz funkcji aktywacji Relu. Kolejno następuje spłaszczenie oraz dwie kombinacje warstwy gęstej i batch norma. ponownie zakończonych Relu. Sieć kończy się warstwą gęstą o liczbie wyjść równej liczbie akcji (7), symbolizującej również wartości Q. Do pamięci trafiała co druga para obserwacji i akcji, natomiast samo uczenie odbywało się co 4 w porcjach po 64 obserwacje na każdą iterację nauki. Sam rozmiar pamięci wynosił 50 tysięcy.

## Rezultaty i wnioski

Zarówno kod, jak i przykładowe rozgrywki agentów w postaci plików gif można znaleźć na repozytorium(12) a także na colabie(13). Jak wspomniano wcześniej, zarówno ze względu na czasochłonność procesu jak i ręczne dostosowanie parametrów pod każdy scenariusz, badania zakończono na nauce 3 agentów na scenariuszach *health gathering*(wykres 2), *deadly corridor*(wykres 1) oraz *basic*. Agent uczony był również na *defend the center*(wykres 3), jednak czas jednej sesji (12 godzin) nie był wystarczający do pełnej nauki, a wykorzystywanie istniejącego modelu w obecnej architekturze kodu powodowało praktyczne rozpoczynanie procesu uczenia od początku – rozwiązaniem mogłoby być zapamiętanie epoki, gdyż to od niej (w zależności od liczby wszystkich epok) zależy stopień eksploracji. Mimo to wyniki są sensowne, choć nie doskonałe(14);
Obecne rozwiązanie miało również swoje korzyści, stąd ze względu na brak czasu nie podjęto problemu, co w przypadku dalszych badań warto rozważyć. Kolejnym jest szersza eksploracja *imitation learning*; mogłoby to pozwolić na przyśpieszenie uczenia cięższych scenariuszach, takich jak *defend the line*, gdzie po 4 epokach (każda po 2 godziny nauki na CPU) agent potrafił jedynie stać i strzelać. Scenariusz jest o tyle ciężki, że wymaga zarówno eliminacji zbliżających się wrogów, dystansowych jak i omijanie pocisków. Można również wytworzyć w nim strategie, by pociski wroga niszczyły zbliżających się wrogów.
Ostatecznie jednak nie udało się doprowadzić projektu do zamierzonego celu, jakim była weryfikacja, czy gracz jest człowiekiem, czy też botem.

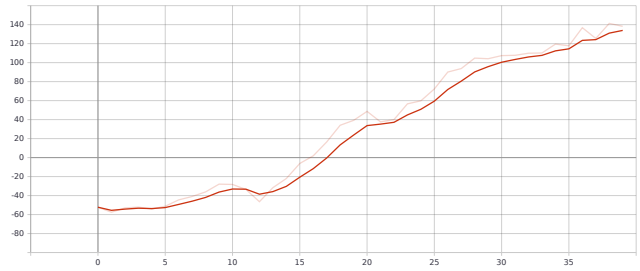


**Fig. 1.** Nagroda w procesie uczenia dla scenariusza *deadly corridor*

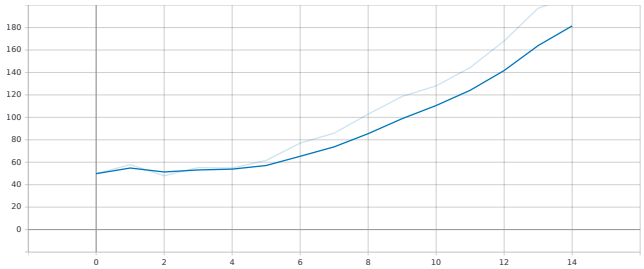


**Fig. 2.** Nagroda w procesie uczenia dla scenariusza *health gathering*

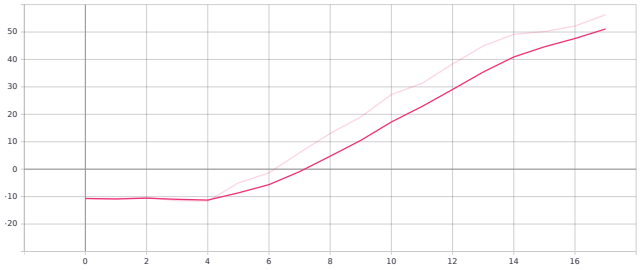


**Fig. 3.** Nagroda w procesie uczenia dla scenariusza *defend the center*

# Bibliography


1. Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojtek Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, Timo Ewalds, Dan Horgan, Manuel Kroiss, Ivo Danihelka, John Agapiou, Junhyuk Oh, Valentin Dalibard, David Choi, Laurent Sifre, Yury Sulsky, Sasha Vezhnevets, James Molloy, Trevor Cai, David Budden, Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Toby Pohlen, Dani Yogatama, Julia Cohen, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Chris Apps, Koray Kavukcuoglu, Demis Hassabis, and David Silver. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/, 2019.
2. Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. ViZDoom: A Doom-based AI research platform for visual reinforcement learning. In *IEEE Conference on Computational Intelligence and Games*, pages 341–348, Santorini, Greece, Sep 2016. IEEE. The best paper award.
3. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.
4. Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networkss. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.
5. Michal Uricár, Pavel Krízek, David Hurych, Ibrahim Sobh, Senthil Yogamani, and Patrick Denny. Yes, we gan: Applying adversarial techniques for autonomous driving. *ArXiv*, abs/1902.03442, 2019.
6. Yang Liu, Yifeng Zeng, Yingke Chen, Jing Tang, and Yinghui Pan. Self-improving generative adversarial reinforcement learning. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, pages 52–60, Richland, SC, 2019. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-1-4503-6309-9.
7. Arna Ghosh, Biswarup Bhattacharya, and Somnath Chowdhury. Sad-gan: Synthetic autonomous driving using generative adversarial networks. 11 2016.
8. Thang Doan, Bogdan Mazoure, and Clare Lyle. Gan q-learning, 2018.
9. Gan q-learning implementation attempt. https://github.com/daggertye/GAN-Q-Learning, 2018.
10. Środowisko gym. https://gym.openai.com/.
11. Pakiet tqdm. https://github.com/tqdm/tqdm.
12. Repozytorium z projektem oraz przykładowymi gifami reprezentującymi rozgrywkę. https://github.com/Azbesciak/VizDoomQPlayer.
13. Colab z projektem. https://drive.google.com/open?id=1FD5lKbp8XZRnc8JWjndFnA4ZqoaT-8ey.
14. Przykład rozgrywki w scenariuszu *defend the center*. https://github.com/Azbesciak/VizDoomQPlayer/blob/master/defend_the_center_optimized.gif.