

**Uniwersytet Jagielloński w Krakowie**

**Wydział Fizyki, Astronomii i Informatyki Stosowanej**

**Marcin Data**

Nr albumu: 11115385

**TetrisBot – zastosowanie uczenia  
maszynowego i algorytmów ewolucyjnych  
do budowy inteligentnego gracza w Tetris**

Praca magisterska  
na kierunku Produkcja Gier

Praca wykonana pod kierunkiem  
dr Katarzyny Grzesiak-Kopeć  
Zakład Technologii Informatycznych

Kraków 2019

## Streszczenie

Niniejsza praca ma na celu zbadanie i porównanie botów stworzonych przy pomocy uczenia maszynowego prowadzących rozgrywkę w grze Tetris. Boty zostały nauczone gry przy pomocy metod inteligencji obliczeniowej. Zaprezentowane są zagadnienia z dziedziny algorytmów ewolucyjnych oraz sieci neuronowych. Istotną częścią pracy są również wyniki przeprowadzonych eksperymentów.

Integralną częścią pracy jest aplikacja, w której zostały zaimplementowane techniki uczenia botów, a także służąca do prezentacji wyników.

Słowa kluczowe: tetris, sztuczna inteligencja, algorytm ewolucyjny, sieci neuronowe, bot, pso, optymalizacja rojem cząstek

---

---

Streszczenie po angielsku [...]

## Spis treści

Rozdział I :Wprowadzenie.....	5
1.1 Cel pracy.....	7
1.2 Zawartość merytoryczna.....	7
Rozdział II : Tetris.....	8
2.1 Wstęp.....	8
2.2 Opis Gry.....	8
2.3 Tetrimino.....	10
2.4 Lock Delay i Wall Kick.....	12
2.5 Wybór tetrimino.....	13
Rozdział III : Tetris i sztuczna inteligencja.....	16
3.1 Wstęp.....	16
3.2 Sposoby umieszczania tetrimino.....	16
3.3 Problem wielkości planszy.....	17
3.4 Heurystyka.....	18
3.5 Strategie reprezentacji planszy.....	18
3.5.1 Strategie reprezentacji planszy poprzez uproszczenie.....	18
3.5.2 Strategie reprezentacji planszy poprzez heurystyki.....	19
3.6 Dotychczasowe rezultaty.....	20
Rozdział IV : Zaimplementowane Heurystyki.....	28
4.1 Heurystyka: Wysokość.....	29
4.2 Heurystyka: Liczba zablokowanych pól.....	30
4.3 Heurystyka: Głębokość studni.....	32
4.4 Heurystyka: Pofałdowanie.....	33
4.5 Heurystyka: Ważona wysokość względna.....	34
4.6 Heurystyka: Pełne linie.....	36
4.7 Heurystyka: Pozycja tetrimino.....	37
4.8 Heurystyka: Przejścia pomiędzy polami planszy.....	38
4.9 Heurystyka: Wypełnienie planszy.....	39
Rozdział V : Zaimplementowani Agenci.....	40
5.1 Ocena ruchu.....	40
5.2 Kryterium wyboru parametrów.....	40
5.3 Dobór parametrów.....	41
5.3.1 Algorytm ewolucyjny.....	41
5.3.1.1 Opis algorytmu.....	42
5.3.1.3 Kodowanie, krzyżowanie i mutacja.....	44
5.3.1.4 Eksperymenty.....	44
5.3.1.4.1 Wpływ ilości rozegranych gier na wynik końcowy.....	45
5.3.1.4.2 Wpływ losowości scenariusza rozegranych gier na wynik końcowy.....	47
5.3.1.4.3 Wpływ spawnera tetrimino na wynik końcowy i proces uczenia.....	48
5.3.1.4.4 Wpływ zestawów heurystyk na wynik końcowy i proces uczenia.....	49

5.3.1.4.5 Wpływ limitu tetrimino na wynik końcowy i proces uczenia.....	51
{wip}.....	51
5.3.1.4.6 Wpływ sposobu selekcji osobników na wynik końcowy i proces uczenia.....	52
{wip}.....	52
5.3.1.4.7 Wpływ wielkości populacji na wynik końcowy i proces uczenia.....	53
{wip}.....	53
5.3.2 Particle Swarm Optimalization.....	54
5.3.2.1 Aktualizacja cząsteczki.....	54
5.2.2.2 Eksperyment.....	55
{wip}.....	55
5.3 Podejście wykorzystujące uczenie maszynowe.....	59
5.3.1 Rodzaje sieci neuronowych.....	59
5.4.3 Q-Learning.....	60
5.4.4 Zaimplementowana sieć neuronowa.....	61
5.4.5 Otrzymane wyniki.....	61
{wip}.....	61
Rozdział VI : Podsumowanie.....	62
{wip}.....	62
Bibliografia.....	63

# *Rozdział I :Wprowadzenie*

[WIP]

W ciągu ostatnich lat obserwowany jest szybki wzrost zainteresowania sztuczną inteligencją oraz sieciami neuronowymi. Jedną z przyczyn takiego stanu rzeczy jest stały wzrost mocy obliczeniowej naszych urządzeń osobistych i mobilnych. Rosnące zapotrzebowanie na rozwiązania, bazujące na sztucznej inteligencji, które zaspokajają potrzeby konsumenckie, a także badawcze powoduje, iż sztuczna inteligencja staje się narzędziem obecnym w wielu aplikacjach. Sztuczna inteligencja w aplikacjach służy też jako skuteczny chwyt marketingowy, ponieważ kto nie chciałby mieć sprzętu, który samodzielnie uczy się nawyków swojego właściciela i rozwiązuje za niego trudne problemy. Kto nie chciałby mieć sztucznego asystenta – osoby stworzonej przy pomocy kodu mającym być jednocześnie opiekunem i przyjacielem swojego właściciela. Taka koncepcja często przewija się w filmach sci-fi lub grach komputerowych. Innym znanym, utartym schematem przewijającym się często w grach jest motyw zbuntowanej SI, groźnego wroga wypranego z uczuć, który w wyniku błędu arytmetycznego, bądź ataku hakerskiego podjął decyzję o eksterminacji gatunku ludzkiego. Za przykład może tutaj posłużyć sztuczna inteligencja o nazwie SKYNET, należąca do filmowego uniwersum wykreowanego przez Jamesa Camerona[1]. Sztuczna inteligencja nie jest tylko elementem narracyjnym w grach.

W projektach typu AAA jakość sztucznej inteligencji odpowiedzialnej za zachowania przeciwników może zadecydować o sukcesie danego projektu. Jednym z najgłośniejszych projektów ostatnich lat, który zasłynął z algorytmu uczącego się przeciwnika jest gra „Alien : Isolation” stworzona przez Creative Assembly[2]. Główny antagonistą bot „Obey” dostaje od systemu zwanego „Reżyserem” częściowe informacje na temat kryjówki gracza, a następnie próbuje go odnaleźć. Bot uczy się sprawdzać kryjówki, w których gracz chowa się najczęściej zmuszając go do ciągłej zmiany strategii.

~~Stworzenie bota, który nie korzysta tylko z oskryptowanych zachowań wymaga doboru odpowiedniego algorytmu. Dobrze dobrany algorytm spowoduje u gracza wrażenie że jego przeciwnik jest inteligentny. Przy doborze algorytmu ważne jest nie tylko uzyskany efekt ale również czas w jakim ten rezultat został osiągnięty oraz skomplikowanie danego algorytmu. Bardziej złożone algorytmy wymagają więcej czasu w przypadku zastosowania ich ponownie w podobnym, kolejnym projekcie.~~

~~W tej pracy podjęta zostaje próba stworzenia botów do gry Tetris przy pomocy znanych algorytmów uczenia maszynowego i algorytmów ewolucyjnych. W tym celu napisano aplikację zawierającą grę Tetris oraz systemy potrzebne do wykonania odpowiednich obliczeń. Nauka sieci neuronowej odbywa się poprzez wykorzystanie frameworku Keras[3].~~

## 1.1 Cel pracy

Celem pracy jest zaimplementowanie trzech agentów do gry Tetris przy pomocy różnych technik inteligencji obliczeniowych. Każdy z agentów został wytrenowany przy pomocy innej techniki. Owe techniki to :

1. Algorytm Ewolucyjny
2. Particle Swarm Optimization
3. Fastforward Neural Network z częściową formułą Q-learning

Następnie wyniki botów zostaną ze sobą zestawione, poprzez porównanie zdobytych przez nie punktów na przestrzeni serii rozgrywek, oraz poprzez ilość czasu potrzebnego na ich wytrenowanie. Jak najlepsze spełnienie tych kryteriów, pozwoli na wskazanie bota, który w tym zestawieniu okazał się najskuteczniejszy, a także ukáže mocne i słabe strony tych rozwiązań.

## 1.2 Zawartość merytoryczna

[WIP]

~~Dokument składa się z sześciu rozdziałów. Rozdział pierwszy jest jednocześnie wstępem. W rozdziale drugim szczegółowo zostanie opisana gra Tetris, ze szczególnym zwróceniem uwagi na istnienie więcej, niż jednego sposobu losowości w grze. Następnie w rozdziale trzecim, zostaną wspomniane dotychczas stworzone boty, ze wskazaniem algorytmów sztucznej inteligencji z jakich korzystały oraz wynikami jakie osiągnęły. W kolejnym rozdziale zostaną opisane heurystyki jakie zostały zaimplementowane w aplikacji celem przeprowadzenia eksperymentów. Natomiast opis bardziej szczegółowo zaimplementowanych botów oraz zaprezentowane dane z procesu nauki, jak również porównanie ich skuteczność po zakończeniu nauki, będzie przedmiotem rozwa-~~

~~zan rozdziały piątego. Ostatni, szósty rozdział zawiera podsumowanie dokumentu. Na końcu pracy znajdują się również bibliografia oraz spisy ilustracji i tabel.~~

## ***Rozdział II : Tetris***

### **2.1 Wstęp**

Tetris to logiczna gra komputerowa, która pojawiła się na światowym rynku 6 czerwca 1984 r. Została ona zaprogramowana przez rosyjskiego matematyka Aleksieja Pajtnov'a [4], a jej nazwa jest wynikiem połączenia greckiego słowa „teta” oznaczającego liczbę cztery z ulubionym sportem Pajtnov'a, którym był tenis. Tetris jest łamigłówką, opartą o grę „Pentomino”( rys 1.), która polegająca na ułożeniu drewnianych klocków zbudowanych z pięciu identycznych kwadratów w prostokątnym pudełku[5]. Należy w tym miejscu zaznaczyć, iż na przestrzeni lat gra Tetris zyskała olbrzymią popularność oraz cieszy się zainteresowaniem szerokiego grona fanów, czego przykładem są m.in. Classic Tetris World Championship[6]-Klasyczne Mistrzostwa Świata Tetris organizowane z USA od 2010 r.



Rys 1: Gra pentomino. Źródło: <https://noskinoski.pl/products/bajo-drewniana-uklanka-pentomino>

### **2.2 Opis Gry**

Gra składa się z prostokątnej planszy podzielonej na mniejsze kwadratowe pola. Najbardziej podstawowa wersja ma postać 10 pól poziomo na 20 pól pionowo. Gracz może kontrolować obroty oraz poziomą pozycję obiektu- powszechnie zwanego



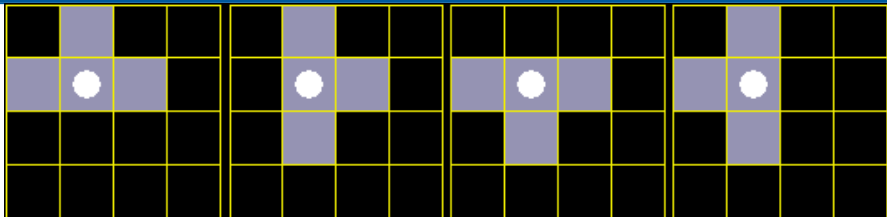
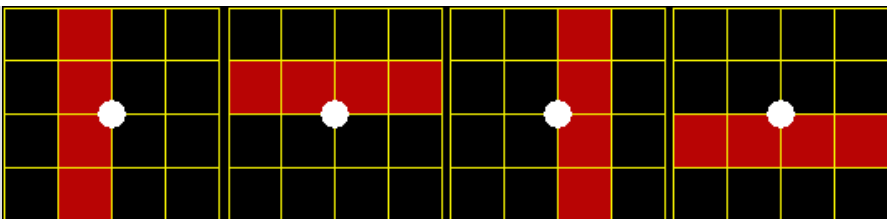
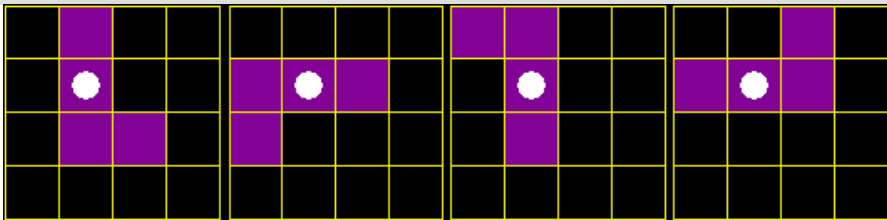
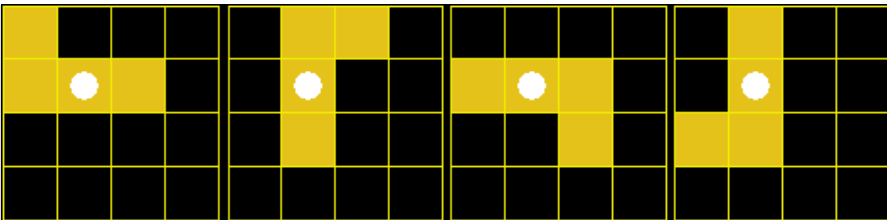
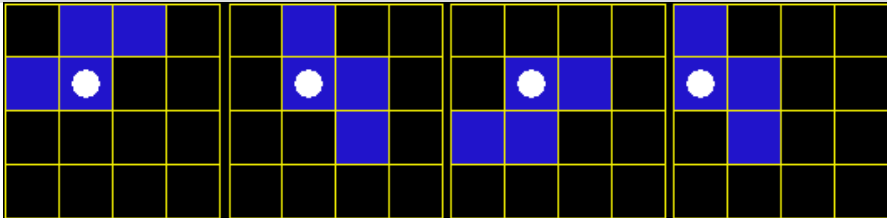
„klockiem”, a w nomenklaturze tetrisa znanego jako „tetrimin”[7], spadającego z góry planszy w taki sposób, żeby zdobywać punkty za każdą wypełnioną poziomą linię.

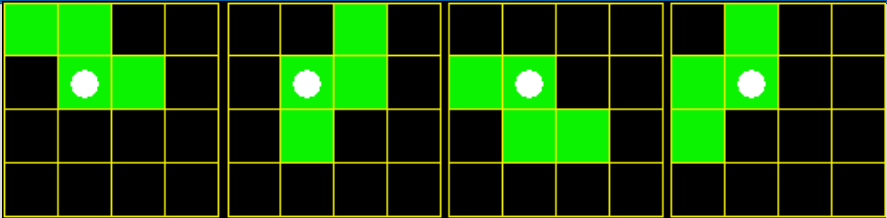
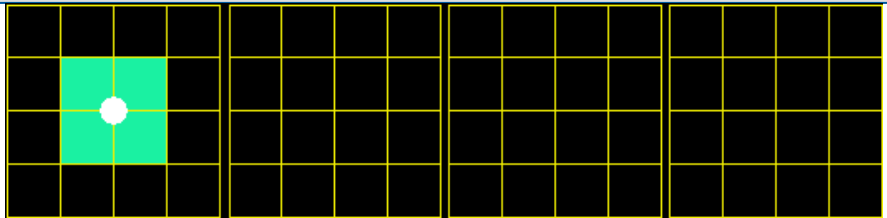
Pojedyncze Tetrimiony, cyklicznie spadają na dół planszy, jednak to do gracza należy decyzja, czy klocek będzie stopniowo opadał czy od razu znajdzie się, w wyznaczonym przez grającego, najniższym dostępnym polu. Celem gry jest ułożenie kolejnych tetrimionów, w taki sposób, by wypełniały poziomo planszę. Wypełnione w ten sposób poziomy, często są nazywane „liniami” i po ich wypełnieniu są automatycznie usuwane z planszy, co skutkuje wydłużeniem czasu rozgrywki. Nieprawidłowe umieszczenie klocka w obrębie danego poziomu, skutkuje pojawieniem się trudności związanych z wyczyszczeniem, nie w pełni zajętej linii na późniejszym etapie gry.

Gracz może zdobywać punkty na kilka sposobów. Niewielką liczbę punktów otrzymuje za samo umieszczenie klocka w dole planszy. Większy bonus przyznawany jest za wyczyszczenie pojedynczej linii, a najwyżej punktowane jest wyczyszczenie kilku linii naraz. W zależności od wersji gry, gracz może mieć podgląd na to, jaki następny klocek pojawi się na planszy po zablokowaniu/ustawieniu właśnie spadającego tetrimionu.

Dodatkowym utrudnieniem jest prędkość gry, która, w zależności od wersji, zwiększa się co określoną liczbę wyczyszczonych linii. Narastające tempo rozgrywki przejawia się w tym, że klocki spadają coraz szybciej, przez co gracz ma mniej czasu na reakcję. Gra kończy się w momencie, kiedy wysokość jednej z kolumn przekroczy bądź zrówna się z dwudziestym polem w pionie.

## 2.3 Tetrimino

Nazwa		Kształty			
T	Rys				
	2:				
Rotacje tetrimino T w SuperRotationSystem					
I					
Rys 3: Rotacje tetrimino I w SuperRotationSystem					
L					
Rys 4: Rotacje tetrimino L w SuperRotationSystem					
J					
Rys 5: Rotacje tetrimino J w SuperRotationSystem					
N / S					
Rys 6: Rotacje tetrimino N/S w SuperRotationSystem					

Nazwa	Kształty
Z	 <p>Rys 7: Rotacje tetrimino Z w SuperRotationSystem</p>
O	 <p>Rys 8: Rotacje tetrimino O w SuperRotationSystem</p>

Tab 1: Rotacje tetrimino w SuperRotationSystem

Słowa „tetromino”, oznacza geometryczny kształt złożony z czterech połączonych ortogonalnie kwadratów[8]. Nazwa ta powstała specjalnie na potrzeby gry.

W trakcie gry można obracać tetrimino w prawą albo lewą stronę. Każda figura ma tzw. punkt obrotu. Punkt ten, jest miejscem względem którego obraca się całe tetrimino o 90 stopni. Za obrót figury odpowiedzialna jest mechanika zwana Rotation Systemem. Natomiast od sposobu jej implementacji, zależy umiejscowienie punktu obrotu oraz ilość możliwych rotacji. [9]

Najwcześniejszy z nich- Original Rotation System został wydany przez Nintendo Entertainment System w 1989 r. [10][11]. Na przestrzeni lat pojawiły się również inne systemy obrotu tetrimino, takie jak:

- Arika Rotation System,
- DTET Rotation System,
- Nintendo Rotation System,
- Original Rotation System,

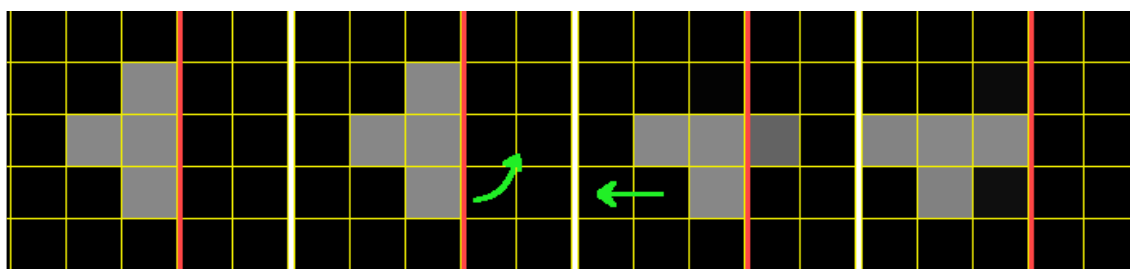
- Sega Rotation,
- Super Rotation System.

W Tab. 1 zaprezentowane zostały możliwe rotacje wszystkich tetriminów występujących w Super Rotation System. W pierwszej kolumnie została wypisana nazwa danego tetrimino, nazwa pochodzi od kształtu litery, którą przypomina dane tetrimino. W drugiej kolumnie ( Rys 2-8 ), zaprezentowane są możliwe obroty danego tetrimino. Białą kropką zaznaczono punkt obrotu względem, którego następuje obrót.

## 2.4 Lock Delay i Wall Kick

W nowszych wersjach gry Tetris, dodane zostały mechanizmy „Lock Delay” oraz „Wall Kick”.

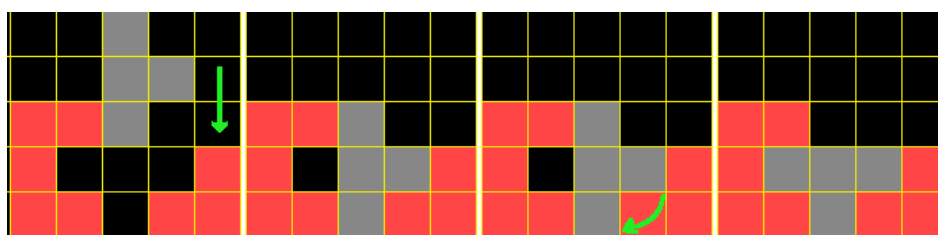
Wall Kick występuje w sytuacji, w której gracz obraca tetrimino blisko krawędzi planszy. W wyniku tego działania figura, nie będzie miała wystarczająco dużo miejsca, żeby pozostać w obecnej pozycji i gra będzie zmuszona przesunąć klocek horyzontalnie, w taki sposób, by dalej mieścił się na planszy. W Rys. 9 zostały przedstawione kolejne kroki umożliwiające wykonanie akcji Wall Kick przy pomocy tetrimino „T”. Obrócenie klocka po ustawieniu go w przedstawionej sytuacji, spowoduje automatyczne przesunięcie się tetrimino od ściany, gdyż w innym przypadku wystawało by ono poza planszę.



*Rys 9: Przykład akcji "Wall Kick" z wykorzystaniem tetrimino T*

Natomiast mechanizm „Lock Delay” pozwala na krótkie manipulacji pozycją oraz wykonanie rotacji, w czasie między zablokowaniem tetrimino w jego ostatecznej pozycji, a dotknięciem przez nie miejsca, w którym może zostać zablokowane. Wyko-

nanie takich kombinacji ruchów, jest możliwe dzięki zastosowaniu tzw. „operacji Twist”, która polega na „wkręceniu” tetrimino w miejsce, którego pozornie nie można wypełnić. Akcje te można łączyć, co pozwala graczom na wykonanie jeszcze ciekawszych manewrów, a także na kreatywną naprawę poprzednio popełnionych błędów. Przykładowa akcja „Twist” została pokazana na Rys. 10. Po opuszczeniu figury na dół planszy, zastosowując mechanizmu „Lock Delay” gracz obraca tetrimino T, w taki sposób, żeby wypełnić pole po lewej stronie, jednocześnie unikając jego zablokowania.



*Rys 10: Akcja "Twist" z wykorzystaniem tetrimino T*

## 2.5 Wybór tetrimino

Ostatnią ważną mechaniką Tetrisa, jest sposób w jaki następuje losowania kolejnej figury, mającej się pojawiać na planszy. W tym miejscu należy dodać, iż sposoby wyboru tetrimino zmieniały się od czasów stworzenia gry.

Pierwsza wersja stworzona w 1985 r. przez autora gry Aleksieja Pajitnova polegała na wybraniu jednego, spośród możliwych tetrimino, za każdym razem kiedy gracz ustawiał figurę na planszy. Bardzo często dochodziło wtedy do tzw. „Efektu powodzi” (ang. flood), w którym graczowi kilkakrotnie do ułożenia przydzielane było to samo tetrimino. Innym, często występującym skutkiem, był tzw. „Efekt suszy” (ang. drought), gdzie pożądanym przez gracza klocek, nie pojawiał się na planszy przez znaczącą część rozgrywki. W swojej publikacji pt. “How to Lose at Tetris” Heidi Burgiel udowodnił, że używając generatora Alekiewa, nie można grać bez końca, istnieje bowiem przynajmniej jedna taka sekwencja złożona z tetrimino “Z” i “S”, która gwarantuje zakończenie gry [12].

Drugi znany sposób losowania figur, został stworzony przez Nintendo cztery lata po stworzeniu gry. W wersji NES Tetrisa został dodany mechanizm, umożliwiająca sprawdzenie czy właśnie wylosowane tetrimino, nie jest tym samym co poprzednio. Jeśli jest, algorytm wylosuje ponownie, ale za drugim razem, nie sprawdzi już, czy wylosowana figura jest inna od poprzedniej. Ten sposób pozwala na niewielkie zminimalizowanie szans powstania „efektu powodzi”, ale nie wyklucza on jednak możliwości zaistnienia „efektu suszy”.

Trzeci znany sposób wybierania figur, został stworzony na potrzeby gry *Tetris : Grand Master (TGM)* wydanej w 1998 r. Wypracowane rozwiązanie wzbogacone zostało o zapamiętywanie dłuższego ciągu poprzednich tetrimino. Zamiast jednego tetrimino algorytm sprawdzał historię, złożoną z czterech ostatnich tetrimino. Jeśli nowe, wylosowane tetrimino znajdowało się w historii, czterokrotnie próbował wylosować takie, którego tam nie ma. Wynik ostatniej próby był zapisywany w historii oraz dodawany na planszę. Ten sposób nie rozwiązywał problemu suszy i powodzi ale zmniejszał szanse na ich wystąpienie. [13]

Następny znany algorytm losowania, zwany jako Random Generator został zaprezentowany w 2001 r., wraz z pojawieniem się na rynku gry *Tetris World*. Algorytm ten, układa sekwencję, złożoną z możliwych, wszystkich siedmiu klocków w dowolnej permutacji. Następnie wszystkie figury pojawiają się w grze w takiej kolejności, w jakiej zostały wylosowane w sekwencji. Po pojawieniu się wszystkich siedmiu klocków, losowana jest kolejna permutacja. Taki system daje maksymalnie 7! możliwych sposobów ich ułożenia. Dzięki temu można łatwo zauważyć, iż po pojawieniu się maksymalnie 12 figur powtórzy się to samo tetrimino. Ogranicza to również możliwość wystąpienia „efektu powodzi” do maksymalnie dwóch takich samych klocków oraz możliwości wystąpienia sekwencji złożonych wyłącznie z tetrim „S” oraz „Z” do maksymalnie 4 tzw. sekwencja węża (ang. snake sequence). Użycie tego systemu jako pierwsze rozwiązało problem tzw. „zabójczej sekwencji” opisaną przez Heidi Burgiela. Natomiast równocześnie pojawił się problem, wynikający z faktu, iż mając tak ograniczoną ilość kombinacji figur, możliwym jest ułożenie strategii, która pozwoli praktycznie grać w nie-

skończoność[14]. Ponadto gra stała się przewidywalna a przez to nie stanowi dla gracza ciekawego wyzwania.

Ostatni system losowania figur, został zaprezentowany światu w 2005 roku w grze *Tetris: The Grand Master 3 – Terror-Instinct*. Rozwinął poprzednio wypracowane rozwiązania, poprzez połączenie losowań sekwencji i zapisywanie ich historii. W tym algorytmie, w jednej sekwencji znajduje się 35 tetrimin po 5 każdego rodzaju. Następnie losuje się z niej jedną figurę, która zostanie dodana na planszę, a na jej miejsce, do puli zostaje wstawione tetrimino, które pojawiło się na planszy najdawniej. System ten skutecznie minimalizuje problemy wystąpienia zarówno Zjawiska Powodzi jak i Zjawiska Suszy, oraz nie pozwala na wystąpienie Zabójczej Sekwencji, z jednoczesnym zwiększeniem trudności gry[15].

Wybór sposobu generowania sekwencji tetrimin ma wpływ na osiągniany przez bota wynik. Zmniejszenie szansy na wystąpienie, bądź całkowite wyeliminowanie możliwości pojawienia się Zabójczej Sekwencji wydłuża czas gry, co równocześnie stanowi okazję dla gracza, do zdobycia większej liczby punktów. Wykorzystując Random Generator, można stworzyć bota grającego niemalże w nieskończoność. Wariant losowania zaproponowany przez twórcę gry Aleksieja Pajitnov'a, ze względu na możliwość wielokrotnego wystąpienia tego samego klocka, uznaje się za wersję najtrudniejszą. Natomiast Random Generator, w której przewidywalność pojawienia się kolejnych figur jest bardzo wysoka skutkuje tym, iż powszechnie przyjęła została opinia, iż wariant tej gry jest prosty do rozwiązania.

## ***Rozdział III : Tetris i sztuczna inteligencja***

### **3.1 Wstęp**

Wybór gry Tetris jak środowiska testowego nie jest przypadkowy. W 1993 roku Lippman, Kukolich and Singer udowodnili, że Tetris jest problemem decyzyjnym Markova. [16] Oznacza to, że wynik działania bota nie zależy jedynie od właśnie podjętej decyzji, ale od sekwencji podjętych wyborów. Ponadto dopiero co rozpoczęta akcja, nie jest skutkiem tego w jakich sytuacjach była ona podejmowana do tej pory. W 2003 roku ukazała się publikacja opisująca możliwość znalezienia optymalnej strategii do Tetrisa jako problem NP-Zupełny[17], co zwiększa atrakcyjność tej gry, jako środowiska testowego dla algorytmów sztucznej inteligencji.

### **3.2 Sposoby umieszczania tetrimino**

W przypadku Tetrisa na wynik bota poza wybranym algorytmem, ma wpływ również: zastosowana strategia reprezentacji gry, użyty system losowania tetrimino oraz sposób jego przemieszczania się po planszy.

Można rozróżnić dwa sposoby przemieszczania się figury po planszy:

- Przesunięcie Tetrimino oraz jego obrót są traktowane jako osobne decyzje: oznacza to, że w danym momencie agent musi albo obrócić daną figurę, albo przesunąć ją po planszy.
- Tetrimino jest opuszczane kolejno na pozycji i w każdej rotacji, po czym dokonywana jest ewaluacja takiego ustawienia i obliczenie możliwego zysku. Tetrimino jest ustawiane w miejscu, w którym zysk był największy. W tym przypadku agent nie może wykonać akcji Twist, ponieważ nie przesuwa tetrimino po planszy, a jedynie wstawia je w odpowiednie miejsce i opuszcza



W pierwszej komercyjnej wersji gry Tetris, pojawiło się specjalne okno, w którym wyświetla się graficzna informacja jakie będzie następne tetrimino. Właściwe wykorzystanie tej informacji ma wpływ na ilość punktów zebranych przez gracza. Symulacja następnego posunięcia w wielu eksperymentach jest ograniczona do jednego ruchu- w przód, a podejście wykorzystujące wiele tetrimino w przód jest rzadko eksploatowane. Podejście wykorzystujące jedno tetrimino jest nazywane „One piece strategy”, natomiast wykorzystujące dwa „Two piece strategy”. W przypadku sprawdzenia większej ilości figur obserwowany jest znaczny wzrost ilości punktów niż, w stosunku do sprawdzania tylko jednego tetrimino. [18][19]

### 3.3 Problem wielkości planszy

Możemy opisać dowolny stan pola planszy do gry Tetris przy pomocy cyfr binarnych, przyjmując że:

- 0 – oznacza puste pole,
- 1 – oznacza pole wypełnione dowolną częścią tetrimino.

Wymiary planszy do tetrisa wynoszą 10 pól szerokości na 20 pól szerokości, z czego wynika że jeden stan planszy składa się z dwustu znaków. Przy pomocy wzoru na wariację z powtórzeniami ( $W_o^p = o^p$ ) można obliczyć na ile sposobów można wybrać znaki do jednego stanu gry. Przyjmując, za  $o$  liczbę stanów w jakich może się znajdować dane pole (2) i za  $p$  liczbę pól (  $10 * 20 = 200$  ) otrzymuje się że możliwa liczba stanów planszy w grze Tetris wynosi  $2^{200}$ . (Formuła 1)

$$\text{Liczba stanów planszy} = W_2^{200} = 2^{200}$$

*Formuła 1: Maksymalna ilość stanów planszy w grze Tetris*

Zakładając, że informacje na temat jednego pola planszy można zapisać na jednym bicie danych, ilość potrzebnego miejsca w pamięci na zapisanie wszystkich stanów, bardzo szybko przekroczy możliwości najlepszego z superkomputerów. Ilość pamięci operacyjnej, jaką posiada najlepszy amerykański superkomputer Summit na rok 2020 wynosi 250PB [20] co w przybliżeniu wynosi  $2^{58}$  bitów. Pomimo prostych zasad gry

Tetris przy obecnej technologii nie jest możliwe zapisanie w pamięci dowolnego urządzenia wszystkich stanów gry.

### **3.4 Heurystyka**

Heurystyką nazywamy sposób rozwiązywania problemów zarówno matematycznych, jak i logicznych, metodą prób i błędów, eksperymentu lub odwołania się do analogii. Metody heurystyczne znajdują zastosowanie tam, gdzie algorytm jest nieznan, albo nie jest on wystarczająco efektywny. Heurystyka umożliwia wyeliminowanie pewnej części przeszukiwanej przestrzeni, co skutkuje przyspieszeniem przeszukania i zmniejszeniem zużycia zasobów. O skuteczności metod heurystycznych świadczą przeprowadzone symulacje, jednak nie istnieją żadne formalne dowody na ich skuteczność [21].

### **3.5 Strategie reprezentacji planszy**

W związku z problemem z niemożliwą do zapisania w żadnym komputerze ilością możliwych stanów planszy do Tetrisa, można wyróżnić dwie strategie reprezentacji planszy, rozwiązujące ten problem:

1. Strategia uproszczająca planszę,
2. Strategia opisująca plansze poprzez heurystyki.

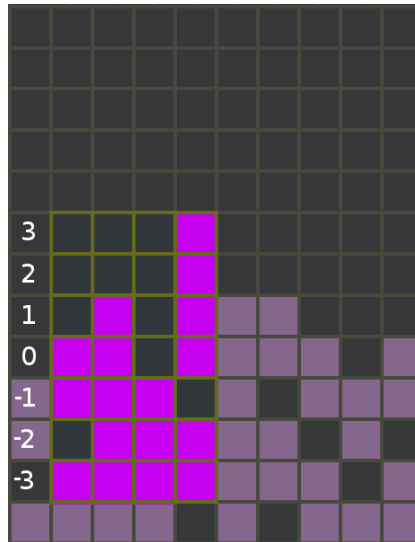
#### **3.5.1 Strategie reprezentacji planszy poprzez uproszczenie**

Rozwiązanie mające na celu uproszczenie planszy polega na zmniejszeniu jej wymiarów albo zakodowanie planszy w mniejsze uproszczone plansze. Na Rys 11 zaprezentowany jest sposób uproszczania planszy, polegający na zakodowaniu czterech kolejnych kolumn. Uproszczona plansza składa się ze zbioru czterech liczb. Każda z tych liczb zawiera się w przedziale  $< -3, 3 >$  i oznacza relatywną wysokość kolumny. W przykładzie zaznaczony stan zostanie opisany liczbami  $\{ 0, 1, -1, 3 \}$ . Takich stanów na

planszy jednocześnie znajduje się 9. Pozwala to obliczyć maksymalną liczbę uproszczonych stanów planszy przy pomocy Formuły 2.

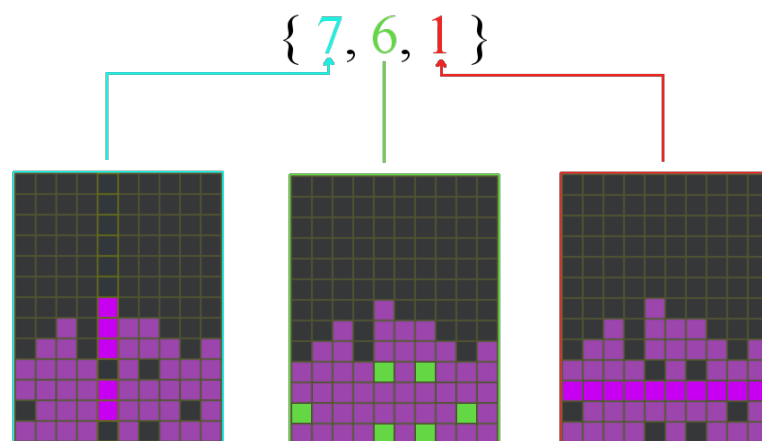
$$\text{Liczba uproszczonych stanów planszy} = 9 * W_7^4 = 9 * 7^4 = 21609 \approx 2^{15}$$

Formuła 2: Liczba uproszczonych stanów planszy w grze Tetris



Rys 11: Sposób uproszczania według pomysłu Samuel J. Sarjant.[22]

### 3.5.2 Strategie reprezentacji planszy poprzez heurystyki



Rys 12: Przykład opisywania planszy przy pomocy metod heurystycznych

Reprezentowanie planszy przy pomocy heurystyk sprowadza się do opisywania wybranych charakterystycznych cech planszy przy pomocy liczb. Ilość wykorzystywanych cech zależy od przyjętego modelu. Na Rys 12 zaprezentowano przykładowy model takiej reprezentacji, wykorzystujący trzy cechy planszy. Kolorem jasnoniebieskim zaznaczona jest heurystyka obliczająca najwyższą kolumnę, kolorem zielonym heurystyka obliczająca ilość zablokowanych pól, a kolorem czerwonym heurystyka obliczająca ilość pełnych linii. Dokładniejsza charakterystyka wymienionych heurystyk została opisana w Rozdziale IV.

### 3.6 Dotychczasowe rezultaty

W Tab 3. zaprezentowane zostały dotychczasowe rezultaty eksperymentów nad stworzeniem bota w grze Tetris. Kolumny przedstawiają kolejno informacje na temat : autorów pracy w której opisano bota, rok publikacji pracy, wykorzystany algorytm, wielkość planszy na jakiej był trenowany dany bot, średni wynik w grze osiągnięty przez bota po procesie nauki oraz wykorzystane metody heurystyczne, jeśli zostały wykorzystane i opisane. W związku z dużą liczbą występujących metod heurystycznych zostały one wypisane w Tab 2. W dwóch ostatnich kolumnach Tab 2. znajdują się nazwy heurystyk, a także ich szczegółowe opisy, w przypadku w którym sama nazwa niewystarczająco tłumaczy sposób jej działania. Pierwsza kolumna w Tab 2 jest numerem porządkowym. Ostatnia kolumna Tab 3 zawiera odniesienie w postaci numeru porządkowego do Tab 2.

Indeks	Nazwa	Szczegółowy opis
1	Maksymalna wysokość kolumny	Wysokość najwyższej kolumny na planszy.
2	Minimalna wysokość kolumny	Wysokość najniższej kolumny na planszy.
3	Średnia wysokość kolumny	Średnia wysokości kolumn na planszy.

Indeks	Nazwa	Szczegółowy opis
4	Różnica między maksymalną i średnią wysokością kolumny	-
5	Różnica między minimalną i średnią wysokością kolumny	-
6	Liczba zablokowanych pól	Liczba wszystkich wolnych pól, które mają nad sobą przynajmniej jedno zajęte pole.
7	Liczba pionowo ze sobą połączonych zablokowanych pól	-
8	Średnia głębokość wszystkich zablokowanych pól	-
9	Wysokość każdej kolumny osobno	-
10	Liczba rzędów wyczyszczonych jednym ruchem	-
11	Różnica wysokości między najniższą i najwyższą kolumną	-
12	Liczba małych studni	Studnia definiowana jest jako puste pole poziomo otoczone zajętymi polami oraz posiadające puste pole bezpośrednio nad sobą.
13	Głębokość największej studni	-
14	Liczba studni	-

Indeks	Nazwa	Szczegółowy opis
15	Suma głębokości ścian wszystkich studni	-
16	Ważona głębokość studni	Suma wszystkich studni gdzie każda część studni liczy się tym mocniej im głębiej się znajduje.
17	Pozycja tetrimino	Wysokość miejsca, w którym tetrimino zostało usytuowane.
18	Sumaryczna wysokość kolumn	Suma wysokości wszystkich kolumn
19	Suma wszystkich zajętych pól na planszy.	-
20	Ważona liczba zajętych pól	Pole w n-tym rzędzie liczy się n razy
21	Liczba poziomych połączonych pól planszy	Jeśli dwie poziomo przyległe pola planszy są wolne to liczą się jako jedno połączone pole.
22	Liczba pionowo połączonych pól planszy	Jeśli dwie pionowo przyległe pola planszy są wolne to liczą się jako jedno połączone pole.
23	Najwyższe zablokowane pole na planszy.	-
24	Liczba zajętych pól nad najwyższym zablokowanym polem.	-
25	Potencjalne rzędy	Liczba rzędów nad najwyższym zablokowanym polem, które mają maksymalnie 2 puste pola.

Indeks	Nazwa	Szczegółowy opis
26	Płynność	Różnica wysokości między dwiema sąsiadującymi ze sobą kolumnami; w odniesieniu do wszystkich kolumn.
27	Różnica wysokości między dwiema sąsiadującymi ze sobą kolumnami.	-
28	Zniekształcone tetrimina	Liczba rzędów wyczyszczonych w ostatnim ruchu pomnożona przez liczbę usuniętych części ostatniego tetrimina, które pojawiło się na planszy.
29	Rzędy z zablokowanym polem	Liczba rzędów, które zawierają przynajmniej jedno zablokowane pole.
30	Głębokość zablokowanego pola	Suma zajętych pól nad każdym zablokowanym polem.
31	Różnorodność wzoru	Suma unikalnych rzędów na planszy.
32	Liczba wolnych pól na planszy.	-
33	Różnica kwadratów maksymalnej i minimalnej wysokości kolumn.	-
34	Suma wszystkich wyczyszczonych rzędów.	-
35	Suma wysokości wszystkich kolumn.	-

Tab 2: Lista opisanych w literaturze heurystyk

Autor	Rok	Algorytm	Rozmiar planszy	Wynik	Użyte cechy
Tsitsiklis & Van Roy [23]	1996	Feature Based Dynamic Programming	16 x 10	32	1, 6
Bertsekas & Tsitsiklis [24]	1996	Neuro-dynamic programming	19 x 10	$28 \cdot 10^2$	1, 6, 9, 27
M. G. Lagoudakis, R. Parr, and M. L. Littman [25]	2002	Least-Squares Pi	20 x 10	$\approx 2 \cdot 10^3$	1, 3, 6, 18, 26, 10, $\Delta 1$ , $\Delta 3$ , $\Delta 6$ , $\Delta 18$ , $\Delta 26$ ,
Kakade[26]	2002	Natural policy gradient	20 x 10	$\approx 5 \cdot 10^3$	1, 6, 9, 27
Dellacherie (Reported by Fahey)[27]	2003	Hand tuned	20 x 10	$6,6 \cdot 10^5$	6, 17, 21, 15, 10
Ramon & Driessens [28]	2004	Relational RL	20 x 10	$\approx 50$	N/A
N. Böhm, G. Kókai, and S. Mandl,[29]	2005	Genetic algorithm (Two piece strategy)	20 x 10	$4,8 \cdot 10^8$	1, 6, 7, 10, 11, 13, 14, 17, 20, 21, 22
Romdhane & Lamon-tagne[30]	2008	Case-based reasoning and RL	20 x 10	$\approx 50$	1, 6, 9, 27, 3, 15, 2, 4, 5, 8,
Farias & Van Roy[31]	2006	Linear programming	20 x 10	$4,274 \cdot 10$	1, 6, 9, 27
Szita & Lörincz[32]	2006	Cross Entropy	20 x 10	$\approx 3,5 \cdot 10^5$	6, 17, 21, 15, 10



Autor	Rok	Algorytm	Rozmiar planszy	Wynik	Użyte cechy
Samuel J. Sarjant[33]	2008	RL	20 x 10	$\approx 2550$	N/A
X. Chen, H. Wang, W. Wang, Y. Shi, and Y. Gao,[34]	2009	Ant Colony Optimization	20 x 10	17586	N/A
Boumaza [35]	2009	CMA-ES	20 x 10	$3,5 \cdot 10^7$	6, 10, 20, 15, 19, 17, 28, 21, 22, 16, 30, 29, 31
Thiery & Sherrier[36]	2009	Cross Entropy	20 x 10	$3,5 \cdot 10^7$	6, 17, 21, 15, 10
L. Langenhoven, W. S. van Heerden, and A. P. Engelbrecht [37]	2010	FFNN + PSO	20 x 10	1286705	10, 1, 6 7, 11, 13, 15, 17, 19, 20, 21, 22, 28
V. Romero, L. To-mes, and J. Yusiong [38]	2011	Harmony search	20 x 10	416928	1, 6, 7, 10, 11, 13, 12, 17, 19, 20, 21, 22, 23, 24, 25, 26, 28, 29, 30
Gabillon et all ...[39]	2013	Classification-based policy iteration	20 x 10	$5,1 \cdot 10^7$	6, 17, 21, 15, 10

Autor	Rok	Algorytm	Rozmiar planszy	Wynik	Użyte cechy
S. Phon-Amnu-aisuk[40]	2015	Genetic algorithm	20 x 10	N/A	1, 6, 7, 10, 11, 13, 14, 17, 20, 21, 22
Thawsitt Naing, Adrien Truong, Orien Zeng[41]	2016	Monte Carlo Tree Search	23 x 10	1748	1, 2, 3, 33, 6, 26, 34
O. Wang, V. Nguyen, M. Hritane, R. Marsart, A. George [42]	2018	PSO (Two piece strategy)	21 x 10	$1,4 \cdot 10^5$	1, 27, 13, 6, 18, 10, 21, 22

Tab 3: Dotychczasowe rezultaty znalezione w dostępnych materiałach[43][44]

Za pierwszego w historii bota do gry Tetris uważa się tego stworzonego w 1996 roku przez Tsitsiklis’a i Van Roy’a. W swojej pracy pt. ”Feature-based methods for large scale dynamic programming” autorzy publikacji, wykorzystali grę Tetris jako środowisko testowe dla swojego algorytmu dynamicznego programowania. Ich celem nie było stworzenie najlepszego na świecie bota z tego powodu też, wynik przez niego osiągnięty nie jest wysoki (średnio 32 punkty na grę). wykreowany przez nich agent, wykorzystywał jedynie dwie cechy do stworzenia reprezentacji planszy Tetris, którymi były: ilość zablokowanych pól oraz wysokość najwyższej kolumny. [45]

Niedługo później Bersekas i Tsitsiklis zwiększyli ilość cech o wysokość każdej kolumny oraz wysokości pomiędzy kolejnymi kolumnami. Przy wykorzystaniu  $\lambda$ -policy iteration osiągnęli wynik około 2800 linii. Podobnie uczynili Lagoudakis, Michail G, Parr, Ronald and Littman zwiększając ilość wykorzystanych cech. Przy pomocy Least-Squares Policy Iteration udało im się uzyskać wynik pomiędzy 1000, a 3000 wyczyszczonych linii na grę. Lepszy wynik w tym samym roku uzyskał Kakade przy pomocy algorytmu Gradient-policy wykorzystując ten sam zestaw cech co Bertsekas i Tsitsiklis.

Stworzony przez niego bot osiągnął wynik 6800 linii. Ten sam zestaw cech wykorzystali Farias i Van Roy osiągając wynik 4500 linii. Dużo niższe ale podobne wyniki osiągnęli Ramon & Driessens (2004) oraz Romdhane i Lamontage (2008) przy pomocy algorytmów uczenia ze wzmocnieniem. Ich wynik wynosi około 50 linii.

Do 2008 roku za jednego z najlepszych botów w grze Tetris uważało tego zreportowanego przez Fahey'a, a stworzonego przez przeciętnego gracza tetrisa, Pierre Dellacherier'a. Dellacherie zaobserwował 6 cech: zablokowane pola, suma studni, poziome i pionowe przejścia pomiędzy pustymi i pełnymi polami planszy, wysokość umiejscowienia ostatniego tetrimina oraz ilość pełnych linii powielona przez ilość części figur, które posłużyły do jego wyczyszczenia. Wagi tych cech dobrał ręcznie i są to kolejno -4, -1, -1, -1, -1 oraz 1. Pozwoliło mu to osiągnąć średni wynik 660 000 wyczyszczonych linii.

W 2005 N. Böhm, G. Kókai, i S. Mandl wykorzystali algorytm genetyczny do stworzenia bota. W swojej implementacji posłużyli się *Two Piece Strategy* oraz użyli nowych cech do opisu stanu gry takie jak: liczba połączonych zablokowanych pól, liczba zajętych pól planszy oraz ważona liczba zajętych pól planszy. Udało im się osiągnąć średni wynik tj. 480 milionów linii przy pomocy funkcji liniowej i 34 miliony przy pomocy funkcji wykładniczej. Wykorzystując ich pracę Thiery i Scherrer (2009) stworzyli bota BCTS, który z średnim wynikiem 35 milionów wygrał w 2008 roku Reinforcement Learning Competition. Inne rozwiązanie wykorzystujące algorytm ewolucyjny (CMA-ES) zostało stworzone przez Boumaza w 2009 r. Z jego pomocą wyczyszczono 35 milionów linii, jednocześnie dając wagi bardzo zbliżone do tych uzyskanych przez Dellacherie'go. W 2013 r. Gabillon znalazł wektor wag umożliwiający mu osiągnięcie 51 milionów linii przy wykorzystaniu algorytmu zainspirowanego przez Lagoudakis i Parr (2003). Był to pierwszy algorytm uczenia ze wzmocnieniem którego szybkość działania była podobna do algorytmu ewolucyjnego. [46]

## ***Rozdział IV : Zaimplementowane Heurystyki***

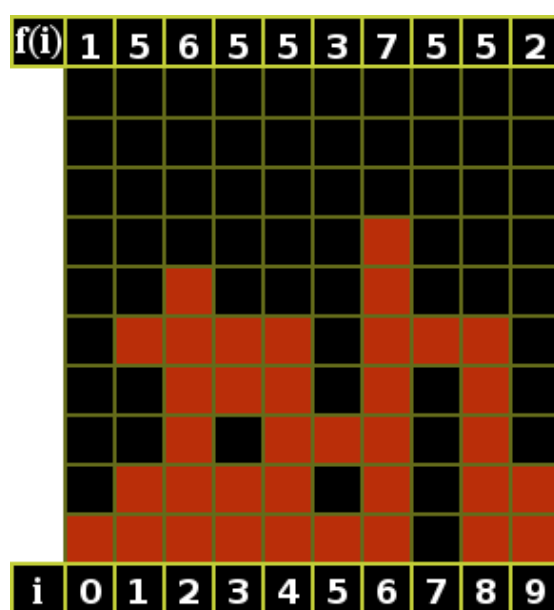
Zaimplementowane przeze mnie heurystyki, to tylko niektóre spośród wszystkich opisanych w tabeli. Zostały częściowo zaczerpnięte z modelu bota Pierre'a Dellacherie, modelu N. Böhm'a, G. Kókai'a, i S. Mandl'a z 2005 r, oraz bota Gabillon'a z 2013 r. Heurystyki w wymienionych modelach zostały intuicyjnie dobrane na podstawie obserwacji wyborów podejmowanych przez gracza w grze Tetris.

Heurystyki zaimplementowane na potrzeby eksperymentów to:

- Średnia wysokość kolumn,
- Suma wysokości kolumn,
- Wysokość najwyższej kolumny,
- Wysokość najniższej kolumny,
- Różnica wysokości najniższej i najwyższej kolumny,
- Liczba zablokowanych pól,
- Głębokość największej studni,
- Suma głębokości studni,
- Pofałdowanie,
- Ważona wysokość względna,
- Pełne linie,
- Pozycja tetrimino,
- Liczba pionowych przejść pomiędzy polami planszy,
- Liczba poziomych przejść pomiędzy polami planszy,
- Suma wszystkich zajętych pól na planszy.

## 4.1 Heurystyka: Wysokość

Na Rys 4. przedstawiony jest przykład planszy z obliczonymi wysokościami kolejnych rzędów. Kolejne kolumny oznaczono indeksem  $i$ , natomiast wysokość kolumny jako wynik funkcji  $f(i)$ . Funkcja  $f(i)$  zwraca liczbę całkowitą z przedziału  $(0,19)$  oznaczającą pozycję najwyżej wypełnionego pola w danej kolumnie  $i$ , licząc od dołu planszy planszy. Wartości funkcji  $f(i)$  można obliczyć w dowolnym układzie tetrimino na planszy.



Rys 13: Przykład planszy z obliczonymi wartościami wysokości kolumn.

W literaturze (Tab 2.) pojawiają się różne sposoby obliczania wysokości planszy. W Tab 4 przedstawiono zaimplementowane heurystyki, razem ze sposobem ich obliczania przy pomocy funkcji  $f(i)$  oraz obliczonymi wartościami dla przykładu z Rys 4. Najwyższą i najniższą kolumnę znajdują się poprzez wybranie spośród wszystkich wartości  $f(i)$  odpowiednio największe i najmniejsze.

Nazwa heurystyki	Sposób obliczania	Wartość dla przykładu
Suma wysokości wszystkich kolumn	$\sum_{i=0}^9 f(i)$	44

Nazwa heurystyki	Sposób obliczania	Wartość dla przykładu
Średnia wysokość wszystkich kolumn,	$\frac{1}{10} \sum_{i=0}^9 f(i)$	4.4
Wysokość najwyższej kolumny	$Max_{i=0}^9 f(i)^1$	7
Wysokość najniższej kolumny	$Min_{i=0}^9 f(i)$	1
Różnica pomiędzy najwyższą i najniższą kolumną.	$Max_{i=0}^9 f(i) - Min_{i=0}^9 f(i)$	6

Tab 4: Sposoby obliczania Heurystyk wysokości

## 4.2 Heurystyka: Liczba zablokowanych pól

Heurystyka oparta o liczbę zablokowanych pól wyznacza, ile pustych przestrzeni przykrytych co najmniej jednym wypełnionym polem, będzie na planszy po wykonaniu danego posunięcia. Każde z zablokowanych pól powoduje podniesienie wysokości, na której można wyczyścić linie. Powoduje to, że gra robi się trudniejsza i jednocześnie coraz szybciej zmierza ku końcowi. Wyczyszczenie wystarczającej liczby linii od góry powoduje pozbycie się już istniejących, zablokowanych pól. Oczekiwana wartość tej heurystyki powinna być jak najmniejsza.

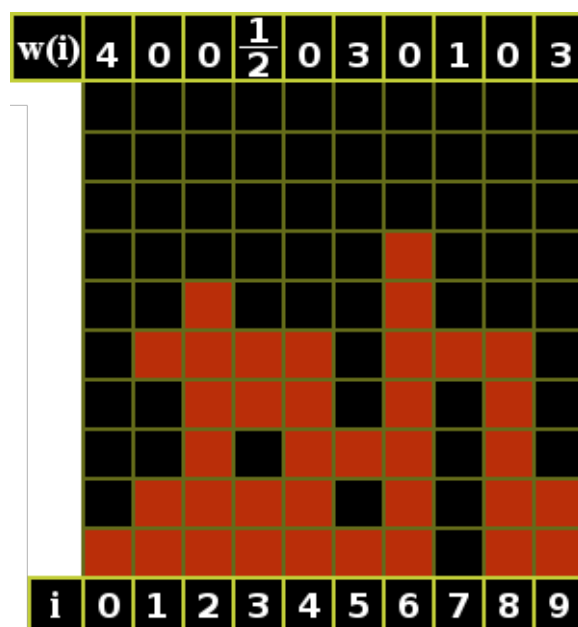
Funkcja  $g(i)$  oblicza ilość zablokowanych pól w postaci liczb całkowitych, w danej kolumnie  $i$ . Na Rys 6. zaprezentowano wartości funkcji  $g(i)$ . Wartość heurystyki zablokowanych pól oblicza się poprzez zsumowanie wartości funkcji  $g(i)$  (Formuła 5). W przykładzie podanym na Rys 6. wartość tej heurystyki wyniosła 5.

<sup>1</sup>  $Max_{i=0}^9 f(i)$  - Największa wartość spośród wartości funkcji  $f(i)$  dla  $i \in \mathbb{N} \wedge i \in \langle 0, 9 \rangle$



### 4.3 Heurystyka: Głębokość studni

Studnia w nomenklaturze gry Tetris oznacza specyficzny układ kolumn. Układ ten składa się z przynajmniej trzech kolumn będących obok siebie. Jeśli dwie skrajne kolumny są najwyższymi kolumnami w układzie to układ ten jest studnią. Takie zmiany mogą być ciężkie do uzupełnienia w sposób, który nie powoduje powstania zablokowania pól. Im większa jest studnia, tym większy jest spadek i tym trudniej będzie go zniwelować. Na Rys. 7 zaprezentowany jest przykład z obliczonymi wartościami heurystyki dla kolejnych kolumn.



Rys 15: Przykład planszy z obliczonymi wartościami studni.

Wartości heurystyk wykorzystujących studnie są obliczane przy pomocy funkcji  $w(i)$  zdefiniowanej w Formule 6. Funkcja  $w(i)$  oblicza wartość studni na podstawie wysokości( $f(i)$ ) trzech kolejnych kolumn ( $i-1, i, i+1$ ). W związku z możliwością wystąpienia studni w ostatniej i pierwszej kolumnie, wartość w tych miejscach oblicza się wykorzystując tylko wartości obecnej i sąsiadującej kolumny. W Tab 5 znajdują się dwie heurystyki, wykorzystujące studnie, a także sposób ich obliczania oraz wartość jaką przyjmują dla przykładu pokazanego na Rys 7.



Nazwa heurystyki	Sposób obliczania	Wartość dla przykładu
Najgłębsza studnia	$Max_{i=0}^9 w(i)$	4
Suma studni	$\sum_{i=0}^9 w(i)$	11.5

Tab 5: Sposoby obliczania Heurystyk wysokości

$$w(i) = \begin{cases} \text{dla } i=0: & Max(f(1)-f(0), 0) \\ \text{dla } i>0 \wedge i<9: & Max(\frac{1}{2}(f(i+1)+f(i-1))-2f(i), 0) \\ \text{dla } i=9: & Max(f(9)-f(8), 0) \end{cases}$$

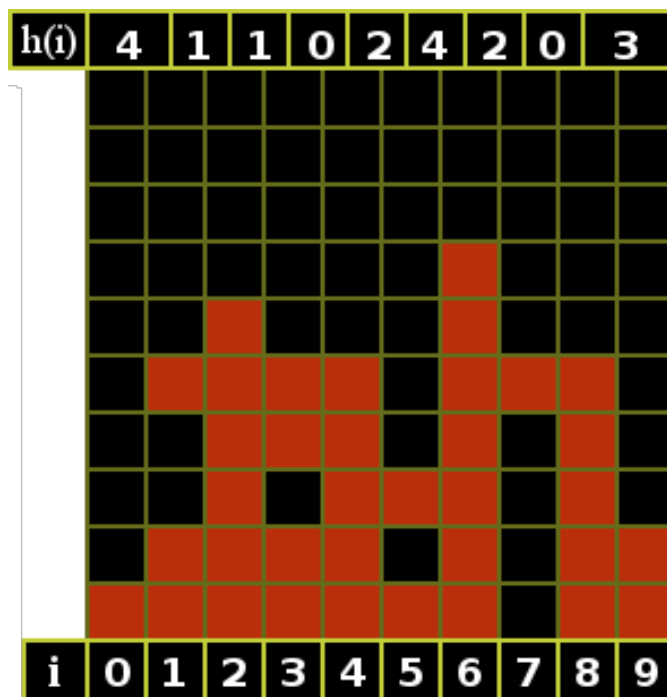
Formuła 6: Sposób obliczania wartości studni

#### 4.4 Heurystyka: Pofałdowanie

Heurystyka oparta o pofałdowanie określa jak bardzo nierównomiernie wypełnione są kolejne kolumny. Na Rys 8. przedstawiono przykładową planszę i obliczone różnice w wysokościach kolejnych kolumn przy pomocy funkcji  $h(i)$ . Wartość funkcji  $h(i)$  oblicza się jako wartość bezwzględna z różnicy dwóch kolejnych kolumn ( $i, i+1$ ). Formuła 7 prezentuje obliczenie wartości tej heurystyki dla przykładu z Rys. 8 Wartość tej heurystyki wynosi 17.

$$Pofa\lhd dowanie = \sum_{i=0}^8 h(i) = \sum_{i=0}^8 \|f(i) - f(i+1)\|$$

Formuła 7: Przykład obliczania wartości heurystyki pofałdowania



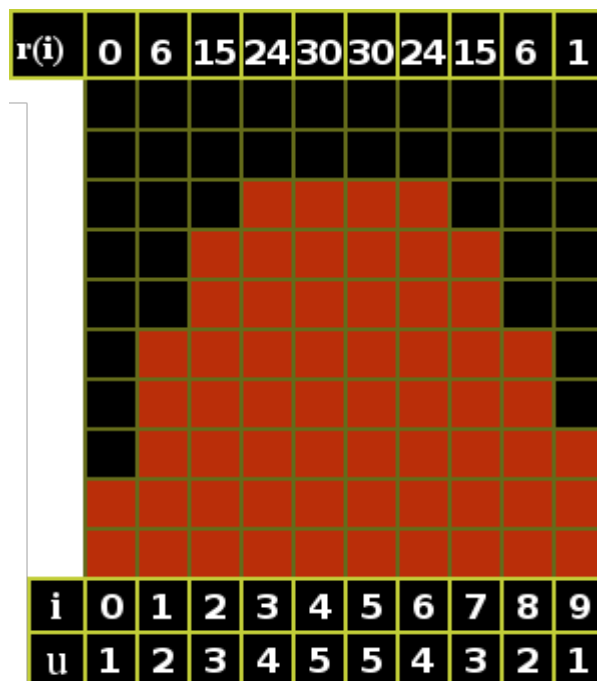
Rys 16: Przykład planszy z obliczonymi wartościami poświadczania.

#### 4.5 Heurystyka: Ważona wysokość względna

Heurystyka ważonej wysokości względnej to sposób przeciwdziałania zjawisku tzw. „wieży”. Jest to układ figur charakteryzujący się tym, że większość tetrimin ułożona jest pośrodku planszy, a w skrajnie oddalonych kolumnach pojawiają się studnie. Sytuacją bardzo niepożądaną byłoby dokładanie tetrimino na szczyt wieży, ponieważ niebezpiecznie zwiększa ono ryzyko przegranej. Każda kolumna i posiada swoją wagę  $u$ . Heurystyka rośnie tym gwałtowniej im wyżej i bliżej środka ustawione są kolejne tetrimino. Wartość funkcji  $r(i)$  oblicza się przy wykorzystaniu najniższej kolumny i wagi kolumny  $i$  (Formuła 8).

$$r(i) = u_i * (f(i) - \text{Min}_{i=0}^9 f(i))$$

*Formuła 8: Obliczanie ważonej względnej wysokości dla kolumny i*



Rys 17: Przykład z wyliczonymi wartościami ważonej względnej wysokości.

Wartość heurystyki ważonej względnej wysokości oblicza się przy pomocy Formuły 9. Dla przykładu przedstawionego na Rys 9. wartość heurystyki wynosi 15,1.

$$\text{WażonaWzględnaWysokość} = \frac{1}{10} \sum_{i=0}^9 r(i)$$

*Formuła 9: Obliczanie wartości heurystyki ważonej względnej wysokości*

## 4.6 Heurystyka: Pełne linie

Heurystyka oparta o pełne linie określa, ile horyzontalnych linii wyczyści dany ruch. Im większa liczba pełnych linii, tym więcej punktów otrzyma bot za ich wyczyszczenie. Funkcja  $k(j)$  określa czy dany wiersz  $j$  jest pełen sprawdzając czy w każdej kolumnie na wysokości  $j$ , pole jest wypełnione. Na Rys X zaprezentowany jest przykład w którym wyliczone zostały wartości funkcji  $k(j)$ . Dla tego przykładu wartość heurystyki wynosi 2. Wartość tą można obliczyć poprzez zsumowanie wartości funkcji  $k(j)$  (Formuła 10)

j																			k(j)
18																			0
...																			0
7																			0
6																			0
5																			0
4																			1
3																			1
2																			0
1																			0
0																			0

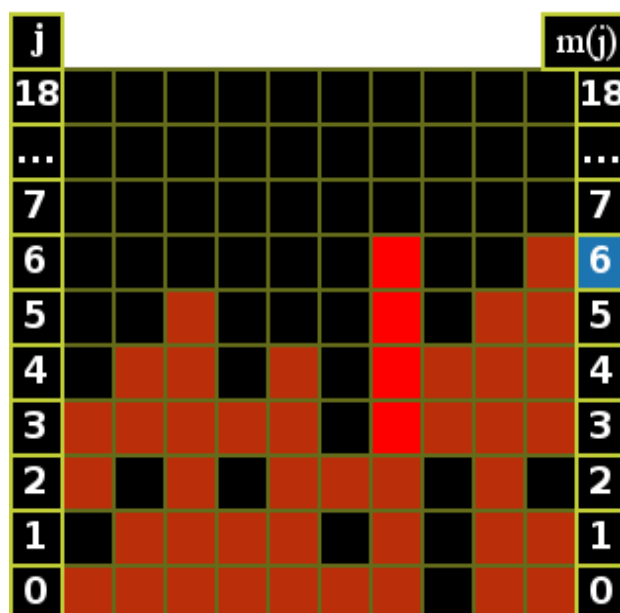
Rys 18: Przykład z wyliczonymi wartościami pełnych linii w każdym wierszu.

$$PełneLinii = \sum_{j=0}^{18} k(j)$$

Formuła 10: Przykład obliczania wartości heurystyki po fałdowaniu

## 4.7 Heurystyka: Pozycja tetrimino

Heurystyka oparta o pozycje tetrimino oznacza wysokość na której zostało upuszczone i zablokowane tetrimino. Na Rys 10. zaznaczono przykładową pozycję umiejscowienia tetrimino „I” oraz sposób oznaczania wysokości przy pomocy funkcji  $m(j)$ . Funkcja  $m(j)$  zwraca największy indeks ( $j$ ) rzędu w którym zatrzymała się część tetrimino. W zaprezentowanym przykładzie jest to górna część tetrimino „I” w wierszu 6.

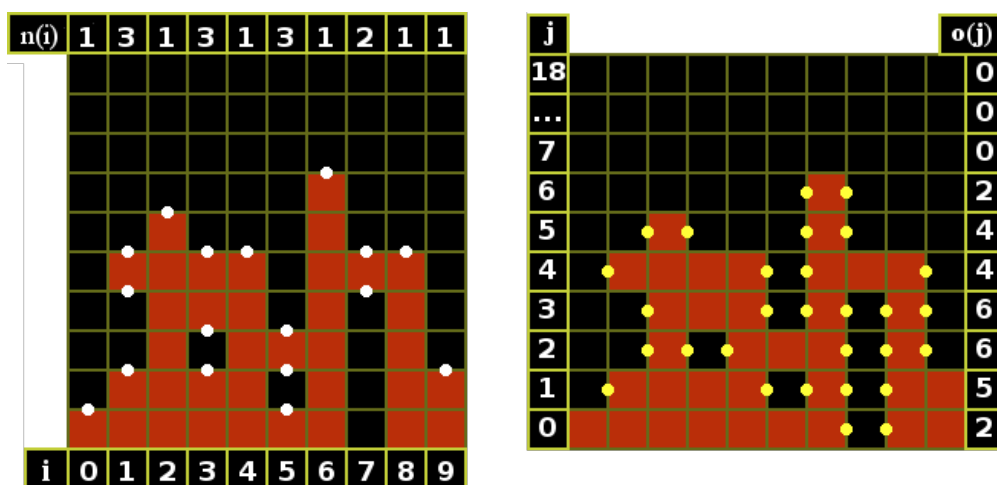


Rys 19: Przykład z zaznaczoną pozycją na jakiej zostanie zablokowane tetrimino „I”.

## 4.8 Heurystyka: Przejścia pomiędzy polami planszy

Heurystyka przejść pomiędzy polami planszy polega na zliczaniu ścian pól planszy, które graniczą jednocześnie z pustym i pełnym polem. Na Rys 11. kolorem białym zaznaczono wszystkie znalezione poziome przejścia pomiędzy polami, przy pomocy funkcji  $n(i)$ . Funkcja  $n(i)$  przeszukuje kolumnę  $i$  w celu znalezienia przejść pomiędzy polami, a następnie zwraca ich ilość. Analogicznie działa funkcja  $o(j)$ . Kolorem żółtym zaznaczono poziome przejścia pomiędzy polami, a ich sumę w wierszu zwraca funkcja  $o(j)$ .

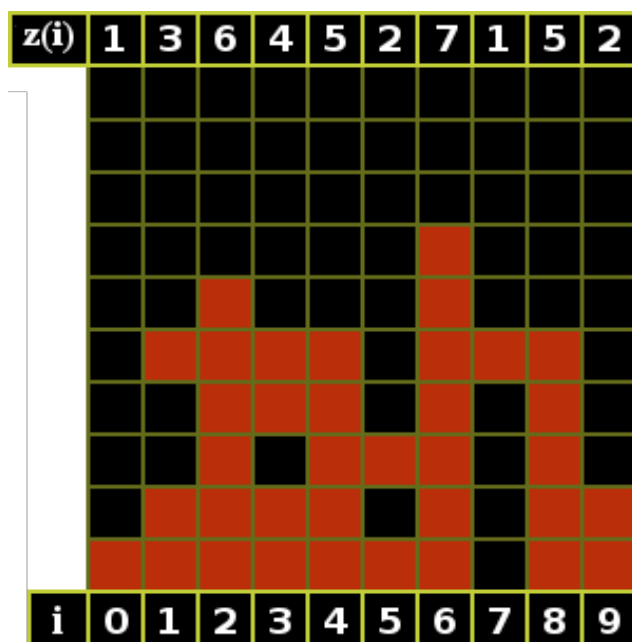
Wartość heurystyki poziomych przejść pomiędzy polami oblicza się poprzez zsumowanie wartości funkcji  $n(i)$  dla każdej kolumny, a wartość heurystyki pionowych przejść oblicza się poprzez zsumowanie wartości funkcji  $o(j)$  dla każdego wiersza. Dla przykładu z Rys 11. wartość heurystyki pionowych przejść wynosi 17, a wartość heurystyki poziomych przejść wynosi 29.



Rys 20: Przykład z zaznaczonymi przejściami pomiędzy polami.

## 4.9 Heurystyka: Wypełnienie planszy

Heurystyka oparta o wypełnienie planszy określa, ile pól planszy zostało wypełnionych. Im większa liczba pełnych pól, które nie tworzą pełnych linii tym szybciej nastąpi koniec gry. Funkcja  $z(i)$  określa ile pól jest pełnych w kolumnie  $i$ . Na Rys X zaprezentowany jest przykład w którym wyliczone zostały wartości funkcji  $z(i)$ . Wartość heurystyki wypełnienia planszy można obliczyć poprzez obliczenie ile % planszy jest zapełnione (Formuła 11). Wartość heurystyki na Rys X wynosi 13.



Rys 21: Przykład z wyliczonymi wartościami wypełnienia kolumn.

$$\text{Wypełnienie planszy} = \frac{1}{200} \sum_{j=0}^9 z(j) * 100\%$$

*Formuła 11: Przykład obliczania wartości heurystyki wypełnienia planszy*

## Rozdział V : Zaimplementowani Agenci

### 5.1 Ocena ruchu

Metodą wykorzystywaną do oceny ruchu jest obliczenie sumy wartości heurystyk. Każda heurystyka ma własny współczynnik określający jej udział w końcowym wyniku. Do obliczenia tej sumy zastosowano Formułę 9. Obliczenia wykonuje się po zasymulowaniu tetrimino na planszy. Dla każdego posunięcia  $i$  w rotacji  $k$  można obliczyć jego jakość  $w_{ik}$ .

$$w_{ik} = \vec{p} * \vec{t}$$

gdzie,

$$\vec{p} = \begin{pmatrix} a \\ b \\ \dots \\ N \end{pmatrix}, \vec{t}_i = \begin{pmatrix} \text{wartośćHeurezyA} \\ \text{wartośćHeurezyB} \\ \dots \\ \text{wartośćHeurezyN} \end{pmatrix}$$

$$a, b, \dots, z \in R,$$

*Formuła 9: Wzór na obliczenie jakości ruchu*

Wektor  $\vec{p}$  to wektor parametrów określających wpływ wartości danej heurystyki na jakość ruchu, natomiast wektor  $\vec{t}_i$  opisuje stan planszy po wykonaniu ruchu  $i$  przy pomocy wartości poszczególnych heurystyk. Wektor  $\vec{p}$  jest wektorem stałym przez całą rozgrywkę. W celu znalezienia najlepszego ruchu wystarczy wybrać ten ruch którego  $w_i$  jest największe.

### 5.2 Kryterium wyboru parametrów

W celu dobrania parametrów bot rozgrywa gry w Tetris, przy użyciu wektora  $\vec{p}$ . Jedna rozgrywka może trwać przez dużą ilość czasu dlatego gra kończy się również w momencie kiedy na planszy zostanie ustawiona określona ilość tetrimino. Za każde



ustawione tetrimino bot otrzymuje jeden punkt, natomiast za wyczyszczone linie otrzymuje punkty według Formuły 10.

$$IlośćPunktów = 2^{(ilość\ wyczyszczonych\ linii - 1)}$$

*Formuła 10: Wzór na obliczenie ilości punktów za wyczyszczone linie*

Ilość punktów za wyczyszczone linie jest dodatkowo mnożona przez  $10^6$  w celu łatwego rozróżnienia ich, od punktów zdobytych za umiejscowione tetrimino. Osiągnięty w takiej rozgrywce wynik świadczy o trafności dobranych parametrów.

### 5.3 Dobór parametrów

Znalezienie wektora  $\vec{p}$  to problem optymalizacyjny do którego zostały zastosowane algorytm ewolucyjny[47] oraz algorytm PSO[48]. Oba algorytmy służą do przeszukiwania przestrzeni rozwiązań w celu znalezienia optymalnego rozwiązania.

#### 5.3.1 Algorytm ewolucyjny

Algorytm ewolucyjny jest algorytmem zainspirowanym procesami naturalnie zachodzącymi w przyrodzie. Naśladują głównie zjawiska występujące w trakcie ewolucji organizmów żywych. Algorytmy ewolucyjne znalazły zastosowanie w rozwiązywaniu problemów optymalizacyjnych czyli problemów w których poszukuje się najmniejszą lub największą wartość pewnego parametru spełniającą jak najlepiej pewną własność. Poszukuje się najlepszego, możliwego rozwiązania spośród pewnej grupy możliwych rozwiązań. Taką grupę nazywa się przestrzenią rozwiązań.

W algorytmie rozwiązanie problemu traktuje się jak osobnika, a zbiór wszystkich rozważanych rozwiązań jak populację osobników. Dla każdego osobnika oblicza się wartość funkcji dopasowania. Obliczona funkcja dopasowania służy jako miara jak dobrze dany osobnik spełnia zadaną własność; rozwiązuje dany problem. Wartości funkcji dopasowania które są bliższe oczekiwanej wartości zwiększają szanse osobnika na przetrwanie i przekazanie swojego genotypu następnemu pokoleniu.

Genotyp jest informacją genetyczną przekazywaną w nici DNA w sensie rozumienia biologicznego oraz sposobem kodowania pojedynczego osobnika w rozumieniu algorytmów genetycznych. Istnieją różne sposoby kodowania pojedynczego osobnika : Można kodować za pomocą liczb binarnych, liczb rzeczywistych, a także całkowitych. Kodowanie binarne stosuje się w sytuacjach gdzie pojedyncze geny można potraktować jako ciąg zer i jedynek. Kodowanie za pomocą liczb rzeczywistych stosuje się, kiedy potrzeba zakodować przy pomocy wartości ciągłych, natomiast kodowanie za pomocą liczb całkowitych, kiedy nie można użyć kodowania binarnego bo jest ono ograniczone tylko do dwóch wartości.[49][50]

### **5.3.1.1 Opis algorytmu**

Działanie algorytmu rozpoczyna się stworzeniem populacji startowej, przy pomocy losowego wygenerowania wybranej liczby osobników. Następnie dla każdego osobnika oblicza się jego wartość funkcji dopasowania. Im wyżej został oceniony dany osobnik tym większa szansa że weźmie on udział w etapie selekcji.

Etap selekcji osobników polega na wybór osobników które wezmą udział w procesie tworzenia nowej populacji. Umożliwia to ukierunkowanie działanie algorytmu w stronę rozwiązań dających lepsze rozwiązania. Istnieje wiele sposobów selekcji osobników w tym najpopularniejsze : wybieranie ruletkowe, wybieranie rankingowe oraz wybieranie turniejowe.

Wybieranie ruletkowe zwane jest również selekcją proporcjonalną. Metoda ta polega na obliczeniu prawdopodobieństwa wylosowania dla każdego osobnika na podstawie jego wartości dopasowania, a następnie wylosowania osobnika. Można to sobie wyobrazić jak koło ruletki gdzie wartości prawdopodobieństwa odpowiadają pewnym wycinkom koła i wycinki są różnej wielkości (zależy od prawdopodobieństwa wylosowania osobnika). Im lepiej przystosowany osobnik tym większą część koła zajmuje i ma większą szansę na zostanie wybranym.

Wybieranie rankingowe polega na przydzieleniu rang wybranej liczbie najlepiej dopasowanych osobników. Rangi są przydzielane w zależności od wartości funkcji

dopasowania. Im wyższa pozycja w rankingu tym większa wartość rangi, która przekłada się na większą szansę zostania wybranym.[51]

Ostatnim najpopularniejszym sposobem wybierania jest wybierani turniejowe. Wybieranie turniejowe składa się z dwóch etapów. W pierwszym etapie wybiera się losowo określoną liczbę osobników z populacji. Następnie w drugim etapie przeprowadza się „turniej” pomiędzy osobnikami. Turniej polega na wybraniu osobnika, który ma największą wartość funkcji dopasowania. Tak wybrany osobnik przechodzi do kolejnego etapu.

Pomiędzy wybranymi osobnikami zachodzi następny etap jakim jest krzyżowanie. Istnieją różne sposoby przeprowadzenia procesu krzyżowania. Można tutaj wymienić, krzyżowanie wymieniające bardzo podobne do procesu mejozy zachodzącego w ludzkich komórkach albo krzyżowanie uśredniające, które oddziałuje na wartości genów. W przypadku krzyżowania wymieniającego potomek otrzymuje pewną kombinację genów swoich rodziców. W przypadku krzyżowania uśredniającego wartości genów plasują się pomiędzy wartościami genomu rodziców. Ostatnim zabiegiem tego etapu jest mutacja.

Mutacja polega na losowej zmianie genów potomka. Nie jest to operacja, którą występuje u każdego nowego osobnika, a jedynie u losowej części z nich. Istnieje wiele sposobów na dokonywanie mutacji genów. Mogą one wpływać pozytywnie lub negatywnie na danego osobnika (poprawić lub pogorszyć jego wartość funkcji dopasowania). Celem mutacji jest zwiększenie różnorodności osobników.

Po wygenerowaniu nowych osobników, kroki algorytmu powtarzają się tak długo jak zostaną spełnione określone warunki stopu. Warunek stopu to kryterium na podstawie, którego następuje zatrzymanie algorytmu. Stosowane kryteria można podzielić na dwie grupy. Pierwsza grupa to kryteria bazujące na monitorowaniu rozwiązań. Wśród nich można znaleźć kryterium stopu związane z zatrzymaniem po osiągnięciu określonej generacji, bądź osiągnięciu zadowalającego poziomu przystosowania (określanego funkcją oceny). Druga grupa to kryteria związane z

ograniczeniem możliwości eksploracyjnych. W tej grupie zawierają się kryteria związane z zanikiem różnorodności populacji albo zanikiem mutacji ( jeśli jest ona samoczynnie adaptująca się). [52]

### 5.3.1.3 Kodowanie, krzyżowanie i mutacja

Zaimplementowany model algorytmu ewolucyjnego korzysta z osobników zakodowanych przy pomocy liczb rzeczywistych. Każdy osobnik jest wektorem o liczbie wymiarów równej liczbie używanych przez bota heurystyk i jest wykorzystywany jako wektor  $\vec{p}$  w procesie oceny przez określoną liczbę gier. Następnie z wyniku obliczana jest średnia która jest wartością dopasowania danego osobnika. Każda gra podczas procesu oceny jest dodatkowo ograniczona limitem tetrimino, w związku z tym że wraz ze wzrostem dopasowania osobników rośnie czas przetwarzania populacji.

Otrzymywanie nowych osobników odbywa się poprzez krzyżowanie uśrednione. Mając wybranych rodziców  $x_1$  i  $x_2$  nowy potomek zostanie stworzony zgodnie z Formułą 10. Dzięki temu wartości nowego osobnika  $\vec{x}'_1$  przyjmą wartości pomiędzy rodzicami z bliższym sąsiedztwem lepiej dostosowanego rodzica.

$$\vec{x}'_1 = \vec{x}_1 * \frac{\text{wynik}_{(\vec{x}_1)}}{\text{wynik}_{(\vec{x}_1)} + \text{wynik}_{(\vec{x}_2)}} + \vec{x}_2 * \frac{\text{wynik}_{(\vec{x}_2)}}{\text{wynik}_{(\vec{x}_1)} + \text{wynik}_{(\vec{x}_2)}}$$

*Formuła 10: Wzór na obliczenie jakości ruchu*

Każdy nowo stworzony osobnik ma procentową szansę na zmutowanie, w tym przypadku oznacza to, że do losowego wymiaru wektora potomnego zostanie dodana wartość +/- 0.2.

### 5.3.1.4 Eksperymenty

Esperymenty mające na celu dobranie wartości wektora  $\vec{p}$  zostały podzielone na dwa etapy. Pierwszy etap dotyczył wpływu poszczególnych parametrów funkcji oceny na etap uczenia i na końcowy wynik bota. Na tym etapie przeprowadzono eksperymenty

na temat losowości scenariusza testowego, ilości rozgrywanych gier, wpływu generatora, doboru heurystyk oraz limit tetrmino.

Drugi etap natomiast dotyczył wpływu parametrów algorytmu genetycznego na skuteczność bota. Eksperymenty te służyły znalezieniu odpowiedniego sposobu selekcji, a także wielkości populacji i szansy na zmutowanie.

#### 5.3.1.4.1 Wpływ ilości rozegranych gier na wynik końcowy

Eksperyment polegający na zbadaniu wpływu ilości rozegranych gier na wynik końcowy dotyczy sprawdzenia ile rozgrywek w grę Tetris powinien rozegrać osobnik w ramach funkcji oceny, aby uzyskany przez niego wynik był jak najwyższy i jak najbardziej stabilny. W tym przypadku stabilny wynik oznacza taki który na przestrzeni stu gier osiąga wyniki jak najbliższe średniej tych wyników. Warunkiem zakończenia procesu uczenia botów w tym eksperymencie było rozegranie przez nie średnio 2000 tetrmino w trakcie oceny.

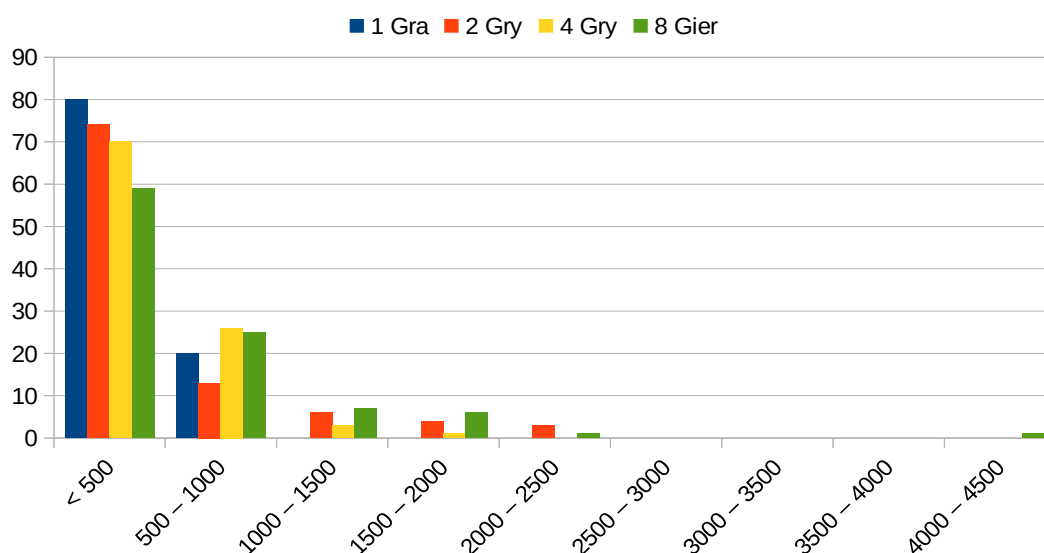
Ilość gier	1	2	4	8
Wynik	804	815	811	757
Generacja	14	5	28	81
Największy wynik	918	2308	1880	4030
Średni wynik	335.05	472.15	504	551.55
Średni czas jednej gry (s)	9.27	13.08	14.17	15.36
Średni czas oceny jednej generacji (s)	93.43	357	966.865	3228.53

Tab 6. Wpływ ilości rozegranych gier na wynik końcowy

Tab 6. prezentuje wyniki eksperymentu dla wartości 1, 2, 4 oraz 8. W tabeli podano dane takie jak wynik jaki osiągnął bot podczas treningu a także numer generacji w jakiej to zrobił. Wiersz „Największy wynik” zawiera wynik osiągnięty po etapie

treningowym, w trakcie stu rozgrywek testowych. W kolejnych wersach wpisane są dane o średnim wyniku osiągniętym w trakcie rozgrywek testowych, średni czas rozgrywki testowej oraz średni czas potrzebny na ocenę jednej generacji podczas treningu.

Na podstawie danych (Tab 6) można wysnuć wniosek że im więcej gier rozegranych w trakcie oceny osobnika tym lepszy wynik osiąga w trakcie rozgrywek testowych. Zarówno stabilność jak i długość jednej rozgrywki rosną wraz z ilością rozegranych gier. Problem jaki wynika ze wzrostu ilości rozgrywek to czas potrzebny na ocenę kolejnych pokoleń. Przy 8 rozgrywkach czas potrzebny na uzyskanie stabilnego wyniku rozegrania 2000 tetrimino wyniósł ponad 200 razy więcej niż w przypadku 1 rozgrywki i ponad 145 razy więcej w przypadku 2 rozgrywek. Wynik osiągnięty podczas treningu poprzez bota rozgrywającego 8 rozgrywek jest niższy niż w przypadku pozostałych botów. Jest to podyktowane faktem że bot ten nie ukończył wymagań treningowych. Na Wyk 1. przedstawiono wyniki botów wykorzystanych w tym eksperymencie podczas testowych 100 rozgrywek. Na osi X zaznaczone przedziały punktowe w jakich kończyły się gry, natomiast na osi Y zaznaczono ilość gier jaka zakończyła się wynikiem mieszczącym się w zadanym przedziale punktowym.



Wyk 1. Rozegrane gry testowe po szkoleniu przy pomocy różnych ilości gier.

#### 5.3.1.4.2 Wpływ losowości scenariusza rozegranych gier na wynik końcowy

Podczas eksperymentu na temat losowości scenariusza rozegranych gier w trakcie treningu i jego wpływu na wynik końcowy, próbowano odpowiedzieć na pytanie : Czy ocena przy pomocy stałych, jednorazowo wygenerowanych scenariuszy zawierających kolejność pojawiania się tetrmino wpływa korzystnie na osiągnięty wynik ?. Na potrzeby tego eksperymentu przygotowano stały scenariusz, zawierający zestaw 2000 tetrmino wygenerowany przy pomocy TGM3's Randomizer'a (Tab 8). Scenariusz losowy był generowany w trakcie rozrywki przy pomocy tego samego randomizera.

Typ scenariusza	Stały	Losowy
Wynik	800	804
Generacja	9	14
Największy wynik	813	918
Średni wynik	270.38	335.05
Średni czas jednej gry (s)	7.45	9.27
Średni czas oceny jednej generacji (s)	17.78	93.43

Tab 7. Wpływ losowości scenariusza rozegranych gier na wynik końcowy

Tab 7. prezentuje wyniki eksperymentu. W tabeli podano dane takie jak wynik który osiągnął bot podczas treningu a także numer generacji w jakiej to zrobił. Wiersz „Największy wynik” zawiera wynik osiągnięty po etapie treningowym, w trakcie stu rozgrywek testowych. W kolejnych wersach wpisane są dane o średnim wyniku osiągniętym w trakcie rozgrywek testowych, średni czas rozgrywki testowej oraz średni czas potrzebny na ocenę jednej generacji podczas treningu.

Na podstawie Tab 7. można odpowiedzieć na pytanie, że ocena przy użyciu stałego scenariusza negatywnie wpływa na wynik, natomiast przyspiesza ona proces treningu.

### 5.3.1.4.3 Wpływ spawnera tetrimino na wynik końcowy i proces uczenia

W historii gry w tetrisa można wyróżnić kilka różnych sposobów generowania kolejności tetrimino. Sposoby te nazywane są potocznie spawnerami albo generatorami. W Tab 8 przedstawiono trzy najistotniejsze spawnera. Celem tego eksperymentu było znalezienie i wybranie spawnera, który pozwoli na uzyskanie najwyższego wyniku.

Indeks	Nazwa	Opis
1	Random Spawner Tetrimino	Spawner uchodzący za najprostszy. Losuje sekwencje siedmiu tetrimino, następnie kolejno je rozgrywa.
2	Oryginalny Spawner	Spawner będący najtrudniejszym, stworzony przez Alexia. Za każdym razem losuje jedno z 7 możliwych tetrimino.
3	TGM3's Randomizer	Spawner stworzony na potrzeby gry <i>Tetris: The Grand Master 3 - Terror-Instinct</i> (2005). Sposób losowania jest kompromisem pomiędzy powtarzalnością kolejnych tetrimino, a wyższą trudnością rozgrywki.

Tab 8. Wpływ losowości scenariusza rozegranych gier na wynik końcowy

Spawner treningowy	1	2	3
Wynik	820	835	849
Iteracja	45	1	3
Średni czas gry jednej generacji (s)	2648.58	880	2472



Spawner testowy	1	2	3	1	2	3	1	2	3
Średni czas jednej gry (s)	16,4	15,2	17,7	43	44	36,8	49,7	58,2	53,5
Najwyższy Wynik	2920	2289	4106	6533	7971	8951	7142	10943	10676
Średni Wynik	635	590	678	1702	1745.2	1457.4	1962	2135	2034

Tab 9. Wpływ generatora na wynik końcowy

W Tab 9. Zaprezentowano wyniki procesu uczenia tego samego bota w różnych generatorach. Numery w wierszach opisanych jako Spawner treningowy i Spawner testowy odnoszą się do indeksów z Tab 8. Bot wytrenowany w jednym generatorze był testowany zarówno na generatorze w którym został wytrenowany jak i w pozostałych. Z Tab 8. wynika, że najlepsze rezultaty daje zastosowanie generatora TGM3's Randomizer. Bot wytrenowany w tym generatorze osiągnął wysokie wyniki w każdym generatorze. Wadą jego zastosowania jest zdecydowanie większy czas potrzebny na ocenę kolejnego pokolenia algorytmu ewolucyjnego.

#### 5.3.1.4.4 Wpływ zestawów heurystyk na wynik końcowy i proces uczenia

Na potrzeby tego eksperymentu przygotowano 7 różnych wektorów  $\vec{t}$ . Każdy z nich składa się z różnych heurystyk, opisanych w Rozdziale IV. Heurystyki zostały dobrane doświadczalnie. Poszczególne wektory, a także wartości odpowiadających im wartości wektora  $\vec{p}$  zostały zaprezentowane w tabeli 10.

Ilość heurystyk	4	5	6	7	8	10	14
Pozycja tetrimino	-0.196	-0.822	-1.831	-2.284	-2.15	-2.47	-0.812
Suma wysokości	-0.666	-4.654	-	-	-	-	-3.627
Średnia wysokość	-	-	-1.36	-1.304	-2.636	-1.472	3.215

Suma zablokowanych pól	-1.3	-2.45	-4.168	-3.237	-	-	-2.68
Poziome przejścia pomiędzy polami	-	-	-	-2.824	-0.43	0.358	-1.033
Pionowe przejścia pomiędzy polami	-	-	-4.054	-2.662	-2.824	-4.013	-3.992
Pofałdowanie	-0.334	-1.381	-2.111	-1.461	-0.633	-2.09	-1.373
Ważona wysokość pofałdowania	-	-	-0.081	-	-0.178	1.06	2.086
Najwyższa kolumna	-	-	-	-	-	-1.226	-3.093
Najniższa kolumna	-	-	-	-	-	-0.757	4.073
Różnica w wysokości najniższej i najwyższej kolumny	-	-	-	1.597	1.05	-	-
Wyczyszczone linie	-	-1.184	-	-	-	1.815	-0.043
Największa studnia	-	-	-	-	-0.846	-	-3.101
Suma wszystkich studni	-	-	-	-	-	-1.09	4.263
Wypełnienie planszy	-	-	-	-	-	-	1.858

Tab 10. Wartości heurystyk dla przygotowanych zestawów

W Tab 11. znajdują się wyniki (wiersz „Wynik”, oraz „Średni czas oceny jednej gry”) poszczególnych botów podczas treningu jak również wyniki osiągnięte podczas rozgrywek testowych („Największy wynik”, „Średni wynik”, „Średni czas jednej gry”). Wiersz „Ilość heurystyk” odnosi się do tak samo nazwanego wiersza Tab 10.

Ilość heurystyk	4	5	6	7	8	10	14
Wynik	757	712	849	860	837	769	761

Generacja	81	19	3	4	93	36	55
Największy wynik	4030	2794	10676	12647	9386	2722	1504
Średni wynik	551.55	553.84	2034	2311	2371	711	503
Średni czas jednej gry (s)	15.36	15.02	53.5	60.05	69	18.94	15.31
Średni czas oceny jednej generacji (s)	3228.53	3158.2	2472	1358.5	1408	3130.34	3230.34

Tab 11. Wyniki przygotowanych zestawów heurystyk

Na podstawie Tab 11. można wyznaczyć 3 wektory  $\vec{t}$ , które osiągnęły wysokie wyniki. Są to wektory o ilości heurystyk 6,7 i 8. Wektor, którego liczba heurystyk wynosiła 8, osiągnął najlepszy wynik średni podczas rozgrywek testowych ale jednocześnie zajęło mu to najwięcej czasu. Trening bota korzystającego z tego wektora trwał ponad półtora dnia, gdzie trening pozostałych dwóch zajął do dwóch godzin. Najlepszym spośród zaprezentowanych wektorów okazał się wektor korzystający z 7 heurystyk. Bot wykorzystujący ten wektor osiągnął najszybciej, najwyższy wynik oraz bardzo zbliżony średni wynik do najlepszego pod tym względem bota wykorzystującego wektor o 8 heurystykach.

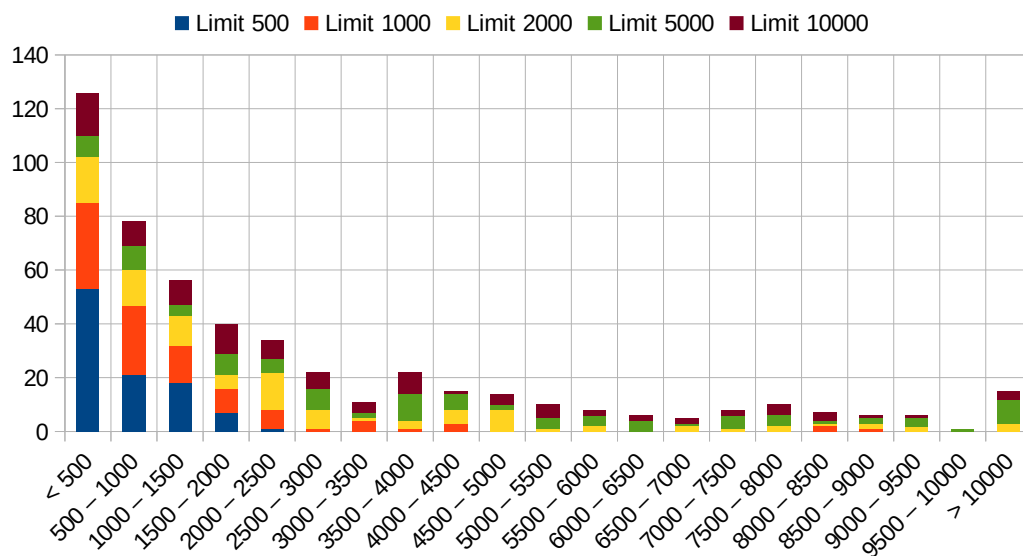
#### 5.3.1.4.5 Wpływ limitu tetrimino na wynik końcowy i proces uczenia

{wip}

Limit tetrimino	500	1000	2000	5000	10000
Wynik	221	421	845	2177	4341
Generacja	4	5	8	6	23
Największy wynik	2003	8750	12577	24222	25615
Średni wynik	666.5	1341.28	2844.31	4653.3	3340.83
Średni czas jednej gry (s)	18.64	39.04	85.23	135.67	88.51

Średni czas oceny jednej generacji (s)	678.456	982.829	1896.5	2669.76	11414.57
----------------------------------------	---------	---------	--------	---------	----------

Tab 12. Wyniki przygotowanych zestawów heurystyk



Wyk 2. Rozegrane gry testowe po szkoleniu przy pomocy różnych limitów w trakcie szkolenia.

#### 5.3.1.4.6 Wpływ sposobu selekcji osobników na wynik końcowy i proces uczenia

{wip}

Sposób selekcji osobników	Wybieranie ruletkowe	Wybieranie rankingowe	Wybieranie turniejowe
Wynik		2168	2117
Generacja		33	6
Największy wynik		15134	24222
Średni wynik		2876.91	4653.31
Średni czas jednej gry (s)		80.45	137.37

Średni czas oceny jednej generacji (s)		3475.59	3051.15
----------------------------------------	--	---------	---------

#### 5.3.1.4.7 Wpływ wielkości populacji na wynik końcowy i proces uczenia

{wip}

Wielkość populacji	50	110	170	230	300
Wynik	847	861	854	841	854
Generacja	24	14	8	5	7
Największy wynik	24201	6316	12577	11687	11262
Średni wynik	3976.36	1599.83	2851.28	2050	2504
Średni czas jednej gry (s)	111.2	41.6	85.5	59.9	72.40
Średni czas oceny jednej generacji (s)	1166.27	2619.87	1896.50	1940.18	1896.5

### 5.3.2 Particle Swarm Optimization

Algorytm PSO został opracowany na podstawie obserwacji zależności zauważonych w zachowaniu zwierząt stadnych takich jak stada ptaków polujących na pożywienie oraz ruchów ławic ryb. Pierwszy raz został zaproponowany w 1995 r. przez J. Kennedy'ego i R. Eberharta w artykule „Particle Swarm Optimization”. PSO zastosowano w prawie każdym obszarze optymalizacji, inteligencji obliczeniowej i aplikacjach projektowych. Istnieją co najmniej dwa tuziny wariantów PSO, a algorytmy hybrydowe łączące PSO z innymi istniejącymi algorytmami są również szeroko badane.

PSO stało się jednym z najczęściej używanych algorytmów opartych na inteligencji roju ze względu na swoją prostotę i elastyczność. Zamiast używać mutacji / krzyżowania wykorzystuje losowość liczb rzeczywistych i globalną komunikację między elementami roju.

Algorytm PSO przeszukuje przestrzeń rozwiązań, dostosowując trajektorie ruchu poszczególnych agentów, zwanych „cząstkami”. Ruch cząstki składa się z dwóch głównych elementów: komponentu stochastycznego i komponentu deterministycznego. Każda cząstka przemieszcza się w stronę najlepszej obecnej pozycji w roju, najlepszej swojej dotychczas znalezionej pozycji z jednoczesnym zachowaniem losowości ruchu. Kiedy cząstka odkryje pozycję lepszą niż ta która do tej pory była jego najlepszą, zostaje ona zapisana jako nowa najlepsza. Przemieszczanie cząstek trwa tak długo jak następuje poprawa bądź przez określoną ilość iteracji. [53]

#### 5.3.2.1 Aktualizacja cząsteczki

Pozycje każdej z cząstek ( $c_x$ ) aktualizuje się formułą korzystającą z dotychczasowego kierunku ( $v_x$ ), dotychczasowo najlepszej pozycji ( $bp_x$ ) oraz globalnej najlepszej pozycji ( $Bp_i$ ). Formuła 11 przedstawia sposób aktualizacji pozycji cząstki  $p_{x,i+1}$ .

$$v_{x,i+1} = w * v_{x,i} + \varphi_1 * r_1 * (bp_x - p_x) + \varphi_2 * r_2 * (Bp_i - p_x)$$

$$p_{x,i+1} = p_{x,i} + v_{x,i+1}$$

Formuła 11: Wzór na aktualizację pozycji cząstki w algorytmie PSO

Pozostałe parametry mające wpływ na zmianę położenia cząsteczki to:

$w$ - nazywane współczynnikiem bezwładności. Oznacza on wpływ wywierany na ruch danej cząsteczki przez zmianę położenia wykonany w poprzedniej iteracji. Wartość tego współczynnika ma wpływ na zdolność cząsteczek do przeszukiwania nowych przestrzeni rozwiązań. Zazwyczaj jego wartość ustawia się w przedziale  $\langle 0.4, 0.9 \rangle$ .

$\varphi_1$ - współczynnik określający w jakim stopniu na zmianę położenia cząsteczki ma wpływ jej dotychczasowe najlepsze położenie. Im większa wartość tego parametru, tym większa skłonność cząstki do oscylacji wokół swojej najlepszej pozycji ( $bp_x$ ).

$\varphi_2$ - współczynnik określający w jakim stopniu na zmianę położenia cząsteczki ma wpływ obecne najlepsze rozwiązanie w całej populacji. Zwiększenie tego parametru sprawia, że wśród cząstek zwiększa się tendencja do grupowania się wokół najlepszego rozwiązania ( $Bp_i$ )

$r_1, r_2$ - losowe wartości z przedziału  $\langle 0, 1 \rangle$ . [54]

### 5.2.2.2 Eksperyment

{wip}

$\varphi_1$	$\varphi_2$	$w$	Iteracje	Treningowy wynik	Średni Wynik Testowy	Najwyższy wynik
0.5	0.5	0.4				
0.5	0.5	0.5				
0.5	0.5	0.6				
0.5	0.5	0.7				
0.5	0.5	0.8				

$\varphi_1$	$\varphi_2$	w	Iteracje	Treningowy wynik	Średni Wynik Testowy	Najwyższy wynik
0.5	0.5	0.9				
1	0.5	0.4				
1	0.5	0.5				
1	0.5	0.6				
1	0.5	0.7				
1	0.5	0.8				
1	0.5	0.9				
1.5	0.5	0.4				
1.5	0.5	0.5				
1.5	0.5	0.6				
1.5	0.5	0.7				
1.5	0.5	0.8				
1.5	0.5	0.9				
0.5	1	0.4				
0.5	1	0.5				
0.5	1	0.6				
0.5	1	0.7				
0.5	1	0.8				
0.5	1	0.9				
1	1	0.4	19	1066	1868.74	9313



$\varphi_1$	$\varphi_2$	w	Iteracje	Treningowy wynik	Średni Wynik Testowy	Najwyższy wynik
1	1	0.5	25	1097	1038	5173
1	1	0.6	12	1076	1324.14	4503
1	1	0.7	11	1028	2367.22	17606
1	1	0.8	19	1093	2141.37	10983
1	1	0.9	40	1107	1172.92	9496
1.5	1	0.4	18	1092		
1.5	1	0.5		1049		
1.5	1	0.6		1039		
1.5	1	0.7		1113		
1.5	1	0.8		1031		
1.5	1	0.9		1097		
0.5	1.5	0.4		1059	1733.1	10487
0.5	1.5	0.5		1018	7792	4437
0.5	1.5	0.6				
0.5	1.5	0.7				
0.5	1.5	0.8				
0.5	1.5	0.9				
1	1.5	0.4	40	1049	1393.46	7792
1	1.5	0.5	30	1102	1793.42	6096
1	1.5	0.6	25	1090	333.2	1098

$\varphi_1$	$\varphi_2$	w	Iteracje	Treningowy wynik	Średni Wynik Testowy	Najwyższy wynik
1	1.5	0.7	37	1039	134.9	425
1	1.5	0.8	16	1058	2122.98	11821
1	1.5	0.9	7	1050	1146.28	4595
1.5	1.5	0.4				
1.5	1.5	0.5				
1.5	1.5	0.6				
1.5	1.5	0.7				
1.5	1.5	0.8				
1.5	1.5	0.9				

## 5.3 Podejście wykorzystujące uczenie maszynowe

Sieci neuronowe znalazły zastosowanie w różnego rodzaju aplikacjach, przykładowo w szukaniu wzorca, data mining albo przewidywaniu szeregów czasowych (ang. time series prediction). W ostatnich latach pojawiło się również wiele prób implementacji sztucznej inteligencji w grach poprzez wykorzystanie uczenia maszynowego. Dla przykładu Tesauro już w 1995 z sukcesem zaimplementował jednokierunkową sieć neuronową (ang. Fastforward neural network), która była w stanie grać w grę Backgammon.<sup>55</sup> W przypadku tej pracy jednokierunkowa sieć neuronowa została wykorzystana razem z algorytmem uczenia ze wzmocnieniem. Takie kombinacje znalazły już zastosowanie w próbach stworzenia agenta do gry Tetris<sup>56</sup> a także innych gier jak choćby *English Draughts* albo *Connect Four*<sup>57</sup>. Zastosowany algorytm ze wzmocnieniem do wyszkolenia agenta to Q-learning.

### 5.3.1 Rodzaje sieci neuronowych

Sieć neuronowa to rodzaj architektury systemu komputerowego. Polega on na przetwarzaniu danych przez struktury zwane neuronami. Neurony sumują wartości sygnałów wejściowych, a następnie, jeśli ta suma przekroczy wymaganą wartość dobranej funkcji aktywacji, wysyłają swój sygnał do następnego neuronu. Neurony pogrupowane są w warstwy a każdy neuron jest połączony z neuronem warstwy poprzedzającej i warstwy następującej. Pierwszą warstwę nazywa się warstwą wejściową, a ostatnią warstwą wyjściową. Odpowiednie wyniki uzyskuje się dzięki procesowi uczenia, który polega na modyfikowaniu wag neuronów. Rozróżnia się co najmniej trzy rodzaje sieci neuronowych:

1. **Jednokierunkowe sieci neuronowe** (ang. Fastforward Neural Network) – Są to sieci, których przepływ sygnałów odbywa się w kierunku od warstwy wejściowej do warstwy wyjściowej. W praktyce jednak pod tą nazwę można przypisać kilka rodzajów sieci neuronowych, jak na przykład wielowarstwowy preceptron (MLP), sieć radialną (RBF), sieci uogólnionej regresji (GRNN) oraz probabilistyczne sieci neuronowe (PNN). Najczęściej jednak nazwa ta odnosi się do sieci MLP.

2. **Rekurencyjne sieci neuronowe** (ang. *Recurrent Neural Network* ) - Są to sieci, w których występuje ponowne wykorzystywanie sygnału wyjściowego niektórych neuronów lub nawet całej sieci. RNN jest bardzo użyteczna w każdym przypadku, gdy zależy nam na odkryciu powtarzającego się wzorca w danych. Ten rodzaj sieci znajduje często zastosowanie przy przetwarzaniu języka naturalnego.

3. **Sieć Neuronowa Kohonena** – (ang. Kohonen's neural networks) jest to rodzaj sieci, który posiada zdolność do reorganizacji własnej struktury, żeby dopasować się do nowego rodzaju danych. Uczenie tego rodzaju sieci polega na zmianach neuronów tak, by dążyły one do wzorca zgodnego ze strukturą analizowanych danych. Sieci zatem „rozpinają się” wokół zbiorów danych, dopasowując do nich swoją strukturę [58].

### 5.4.3 Q-Learning

Q-learning jest sposobem szukania najlepszej możliwej akcji w danym stanie środowiska. Najbardziej powszechna wersja tego algorytmu polega na skonstruowaniu tablicy ( $Q$ ) łączącej w sobie stan środowiska ( $s$ ) i akcje ( $a$ ) jaką należy podjąć w tej sytuacji. Proces uczenia składa się z dwóch etapów: eksplorowania oraz użytkowania. Podczas pierwszego z nich agent podejmuje losowe decyzje, które zostają wykorzystane do poznania przez agenta środowiska oraz zbudowania tablicy. W drugim etapie decyzje będą wybierane na podstawie stworzonej tablicy. Po każdej akcji tablica jest aktualizowana następującą formułą:

$$Q[s, a]_i = Q[s, a]_i + \alpha * (reward + \gamma * \max(Q[s, a]_{i+1}, Q[s, a]_i)) - Q[s, a]_i$$

gdzie,

$reward$  - oznacza nagrodę jaką otrzymał agent w akcji, którą wykonał

$\alpha$  - jest współczynnikiem nauki oznaczającym w jakim stopniu akceptuje się nagrodę kolejnego ruchu  $i$  jako lepszą niż obecnie posiadana

$\gamma$  - współczynnik balansujący, zazwyczaj przyjmuje wartość  $\gamma = \langle 0.8, 0.99 \rangle$

#### 5.4.4 Zaimplementowana sieć neuronowa

Na potrzeby tej pracy zaimplementowana została sieć neuronowa typu Fastforward. Składa się ona z pięciu warstw neuronów. Pierwsza warstwa wejściowa składa się z 6 neuronów, po jednym na każdą heurystykę służącą do opisu planszy Tetrisa. Następnie są trzy warstwy po 32 neurony każda. Każda z tych trzech warstw korzysta też z rektyfikowanej liniowej funkcji aktywacji jednostki (ang rectified linear unit activation. RELU). Ostatnia warstwa składa się z jednego neuronu z liniową funkcją aktywacyjną. Wartość zwracana ( $r$ ) przez sieć jest domniemaną nagrodą jaką bot otrzyma z danego stanu planszy. Na podstawie tej wartości można ocenić czy ruch jest lepszy, czy gorszy niż inne, które można w danej sytuacji wykonać.

Do nauki sieci został wykorzystany algorytm Q-Learning. W związku z problemem stworzenia reprezentacji tablicy w pamięci komputera została ona spłycona do kolejki  $B$  przechowującej  $10^6$  kolejnych stanów gry  $s_i$  jak również stanów gry  $s_{i+1}$  po umiejscowieniu tetrmino oraz informacji czy ruch kończy daną grę  $d = \{true, false\}$  i wartości pełnych linii w danym ruchu  $x$ .

$$B_i = (s_i, s_{i+1}, d, x)$$

Po osiągnięciu przez kolejkę połowy swojej pojemności, co kilka akcji (5) wcześniej wspomniana sieć neuronowa jest uczona przez 10 epok, 128 losowo wybranymi stanami gry. Informacją wejściową jest stan  $s_i$ , a oczekiwaną wartość końcową oblicza się przy pomocy poniższej formuły:

$$Q(B_i) = \{d: true, x\} \vee \{d: false, x + \gamma * r(s_{i+1})\}$$

#### 5.4.5 Otrzymane wyniki

{wip}

## ***Rozdział VI : Podsumowanie***

{wip}

# Bibliografia

[<sup>1</sup>] *Terminator* [online], [dostęp: 16.06.2021], dostępny w internecie:

[https://pl.wikipedia.org/wiki/Terminator\\_\(film\)](https://pl.wikipedia.org/wiki/Terminator_(film))

[<sup>2</sup>] *Najlepsza gra Alien jest obecnie bezpłatna na PC* [online], [dostęp: 16.06.2021], dostępny w internecie:

<https://www.cyberfeed.pl/najlepsza-gra-alien-jest-obecnie-bezplatna-na-pc/>

[<sup>3</sup>] *Keras* [online], [dostęp: 16.06.2021], dostępny w internecie:

<https://keras.io/>

[<sup>4</sup>] Box Brown, *Tetris ludzie i gry*, przeł. M, Wróbel, Warszawa, Marginesy, 2019, ISBN: 9788366140752, s. 1.

[<sup>5</sup>] Box Brown, *Tetris ludzie i gry*, przeł. M, Wróbel, Warszawa, Marginesy, 2019, ISBN: 9788366140752, s. 72-73.

[<sup>6</sup>] *Classic Tetris World Championship* [online], [dostęp: 28.06.2021], dostępny w internecie:

<https://thectwc.com/>

[<sup>7</sup>] *Tetrimino* [online], [dostęp: 15.11.2020], dostępny w internecie:

<https://en.wikipedia.org/wiki/tetrimino>

[<sup>8</sup>] *Tetrimino* [online], [dostęp: 15.11.2020], dostępny w internecie:

<https://en.wikipedia.org/wiki/tetrimino>

[<sup>9</sup>] *Rotation Systems* [online], [dostęp: 15.11.2020], dostępny w internecie:

[https://tetris.wiki/Category:Rotation\\_systems](https://tetris.wiki/Category:Rotation_systems)

[<sup>10</sup>] *Tetris* [online], [dostęp: 15.11.2020], dostępny w internecie:

[https://tetris.fandom.com/wiki/Tetris\\_\(NES,\\_Nintendo\)](https://tetris.fandom.com/wiki/Tetris_(NES,_Nintendo))

[<sup>11</sup>] *Original Rotation System* [online], [dostęp: 15.11.2020], dostępny w internecie:

[https://tetris.wiki/Original\\_Rotation\\_System](https://tetris.wiki/Original_Rotation_System)

[<sup>12</sup>] Heidi Burgiel, *How to lose Tetris* [online], [dostęp: 20.03.2021], dostępny w internecie:

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.55.8562&rep=rep1&type=pdf>

[<sup>13</sup>] Simon Laroche, *The history of tetris randomizers* [online], [dostęp: 20.03.2021], dostępny w internecie:

<https://simon.lc/the-history-of-tetris-randomizers>

[<sup>14</sup>] *Playing forever* [online], [dostęp: 20.03.2021], dostępny w internecie:

[https://harddrop.com/wiki/Playing\\_forever](https://harddrop.com/wiki/Playing_forever)

[<sup>15</sup>] *A deadly piece sequence* [online], [dostęp: 15.11.2020], dostępny w internecie:

[http://harddrop.com/wiki/A\\_deadly\\_piece\\_sequence](http://harddrop.com/wiki/A_deadly_piece_sequence)

[<sup>16</sup>] Tsitsiklis, John N and Van Roy, Benjamin. *Feature-based methods for large scale dynamic programming*. Machine Learning, 22(1-3):59–94, 1996.

- [17] Demaine, Erik D, Hohenberger, Susan, and Liben-Nowell, David. *Tetris is hard, even to approximate*. In International Computing and Combinatorics Conference, pp.351–363. Springer, 2003.
- [18] <https://github.com/OscarWang114/PSO-Tetris-AI> (28 II 2021)
- [19] N. Bohm, G. Kokai, and S. Mandl, *An evolutionary approach to tetris*, in *The Sixth Metaheuristics International Conference (MIC2005)*, 2005
- [20] [https://pl.wikipedia.org/wiki/Summit\\_\(superkomputer\)](https://pl.wikipedia.org/wiki/Summit_(superkomputer)) (07 II 2021)
- [21] Rutkowski I..., *Metody i techniki sztucznej inteligencji*, Wydawnictwo Naukowe PWN, Warszawa 2005, s. 12
- [22] Samuel J. Sarjant, *SmartAgent - Creating Reinforcement Learning Tetris AI*, , 2008
- [23] Tsitsiklis, John N and Van Roy, Benjamin. *Feature-based methods for large scale dynamic programming*. Machine Learning, 22(1-3):59–94, 1996.
- [24] Bertsekas, Dimitri P and Tsitsiklis, John N. *Neuro-Dynamic Programming*. Athena Scientific, 1996.
- [25] Lagoudakis, Michail G, Parr, Ronald, and Littman, Michael L. Least-squares methods in reinforcement learning for control. In Hellenic Conference on Artificial Intelligence, pp. 249–260. Springer, 2002.
- [26] Kakade, Sham. A natural policy gradient. Advances in neural information processing systems, 2:1531–1538, 2002.
- [27] Fahey, Colin. Tetris ai, June 2003. URL <http://www.colinfahey.com/tetris/tetris.html>.
- [28] J. Ramon and K. Driessens, “On the numeric stability of gaussian processes regression for relational reinforcement learning,” in *ICML-2004 Workshop on Relational Reinforcement Learning*, 2004, pp. 10–14.
- [29] Bohm, N, Kóokai, G, and Mandl, S. An Evolutionary Approach to Tetris. Proceedings of the 6th Metaheuristics International Conference (MIC2005), pp. 137–148, 2005.
- [30] Romdhane, Houcine and Lamontagne, Luc. Reinforcement of local pattern cases for playing tetris. In FLAIRS Conference, pp. 263–268, 2008.
- [31] Farias, Vivek F and Van Roy, Benjamin. Tetris: A study of randomized constraint sampling. In Probabilistic and Randomized Methods for Design Under Uncertainty, pp. 189–201. Springer, 2006.
- [32] I. Szita and A. Lorincz, “Learning tetris using the noisy cross-entropy method,” *Neural computation*, vol. 18, no. 12, pp. 2936–2941, 2006.
- [33] Samuel J. Sarjant, *SmartAgent - Creating Reinforcement Learning Tetris AI*, , 2008
- [34] X. Chen, H. Wang, W. Wang, Y. Shi, and Y. Gao, “Apply ant colony optimization to tetris,” in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. ACM, 2009, pp. 1741–1742.



- [35] A. Boumaza, "On the evolution of artificial tetris players," in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*. IEEE, 2009, pp. 387–393.
- [36] Thiery, Christophe and Scherrer, Bruno. Improvements on learning tetris with cross entropy. *Icga Journal*, 32(1):23–33, 2009a.
- [37] L. Langenhoven, W. S. van Heerden, and A. P. Engelbrecht, "Swarm tetris: Applying particle swarm optimization to tetris," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE, 2010, pp. 1–8.
- [38] V. M. Romero, L. L. Tomes, and J. P. T. Yusiong, *Tetris agent optimization using harmony search algorithm*, International Journal of Computer Science Issues, vol. 8, no. 1, pp. 22–31, 2011.
- [39] Gabillon, Victor, Ghavamzadeh, Mohammad, and Scherrer, Bruno. Approximate dynamic programming finally performs well in the game of tetris. In *Advances in neural information processing systems*, pp. 1754–1762, 2013.
- [40] S. Phon-Amnuaisuk, "Evolving and discovering tetris gameplay strategies," *Procedia Computer Science*, vol. 60, pp. 458–467, 2015.
- [41] Thawsitt Naing, Adrien Truong, Orein Zenh, "Tetris AI", 2016
- [42] O. Wang, V. Nguyen, M. Hritane, R. Massart, A. George, „Learning to Play Tetris with Big Data”, 2018
- [43] Renan Samuel da Silva, Rafael Stubs Parpinelli, *Playing the Original Game Boy Tetris Using a Real Coded Genetic Algorithm*, Brazilian Conference on Intelligent Systems, 2017
- [44] Simon Algorta, Ozgur Simsek, *The Game of Tetris in Machine Learning*, 2017
- [45] Tsitsiklis, John N and Van Roy, Benjamin. *Feature-based methods for large scale dynamic programming*. Machine Learning, 22(1-3):59–94, 1996.
- [46] Renan Samuel da Silva, Rafael Stubs Parpinelli, *Playing the Original Game Boy Tetris Using a Real Coded Genetic Algorithm*, Brazilian Conference on Intelligent Systems, 2017
- [47] Bohm, N, Kóokai, G, and Mandl, S. An Evolutionary Approach to Tetris. *Proceedings of the 6th Metaheuristics International Conference (MIC2005)*, pp. 137–148, 2005.
- [48] O. Wang, V. Nguyen, M. Hritane, R. Massart, A. George, „Learning to Play Tetris with Big Data”, 2018
- [49] John H. Holland. *Genetic Algorithms Computer programs that "evolve" in ways that resemble natural selection can solve complex problems even their creators do not fully understand*. [online], [dostęp: 15.04.2021], dostępny w internecie: <https://www2.econ.iastate.edu/tesfatsi/holland.gaintro.htm>
- [50] Kornel Chromiński. *Zastosowanie procesów epigenetycznych w algorytmach genetycznych* [online], [dostęp: 15.04.2021], dostępny w internecie: <https://core.ac.uk/download/pdf/270093602.pdf>

- [51] Kornel Chromiński . *Zastosowanie procesów epigenetycznych w algorytmach genetycznych*  
[online], [dostęp: 15.04.2021], dostępny w internecie: <https://core.ac.uk/download/pdf/270093602.pdf>
- [52] J. Arabas, *Wykłady z algorytmów ewolucyjnych*, Warszawa: Wydawnictwo Naukowo-Techniczne, 2001.
- [53] Xin-She Yang , *Nature-Inspired Optimization Algorithms* , Londyn: Wydawnictwo Elsevier , 2014. ISBN 9780124167438, s. 32-33.
- [54] <http://www.alife.pl/optymalizacja-rojem-czastek> (2 III 2021)
- [55] Tesauro G, „Temporal difference learning and TD-Gammon”. Commun, 1995
- [56] Patrick Thiam, Viktor Kessler, and Friedhelm Schwenker, „A Reinforcement Learning Algorithm to Train a Tetris Playing Agent”, 2014
- [57] Fauser, S., Schwenker, F, „Ensemble methods for reinforcement learning with function approximation” Springer, Heidelberg (2011)
- [58] R. Tadeusiewicz, M. Szaleniec „Leksykon sieci neuronowych” , Wydawnictwo Fundacji „Projekt Nauka” , Wrocław, 2015