POP 25Z

DOKUMENTACJA PROJEKTOWA

Bartosz Psik

Amadeusz Lewandowski

Treść zadania projektowego

Znajdź idealny skład pokemonów które będą w stanie wygrać jak najwięcej pojedynków. Każdy pokemon charakteryzuje się innymi parametrami (atak, obrona) oraz ich skutecznością przeciwko konkretnym typom pokemonów. Z reguły np. wodne pokemony są skuteczne przeciwko ogniowym, natomiast mało skuteczne przeciwko trawiastym itp. Bohater może posiadać jednocześnie maksymalnie 6 pokemonów, dlatego zadanie polegać będzie na znalezieniu takiej idealnej kombinacji, która będzie skuteczna przeciwko największej liczbie przeciwników, jednocześnie nie będąc zbyt wrażliwa na niektórych z nich. W dobieraniu kombinacji proszę wziąć pod uwagę częstotliwość występowania poszczególnych gatunków. Do poprawnego rozwiązania konieczne będzie dokładne przeanalizowanie danych, ponieważ występuje w nich kilka obserwacji odstających które należy w jakiś sposób uwzględnić.

Dane: https://www.kaggle.com/rounakbanik/pokemon

Założenia projektowe

Przestrzeń przeszukiwań

Jako przestrzeń przeszukiwań traktujemy sześcioelementowe wektory zawierające pokemony, czyli po prostu drużyny. Jako sąsiada danej drużyny w przestrzeni traktujemy drużynę, która różni się od niego jednym pokemonem.

W celu spełnienia wymagania zadania, aby drużyna nie była zbyt wrażliwa na niektórych przeciwników (tj., aby była „kontra” na wszystko) i zmniejszenia przestrzeni (szybsze i łatwiejsze przeszukiwanie) drużyna będzie musiała zawierać unikalne typy pokemonów np. jeśli mamy już w drużynie np. Tentacruela (typ wodny i trujący) to nie możemy mieć już Arboka (typ trujący) ani Vaporeona (typ wodny). Ograniczenie to wymusza różnorodność funkcjonalną drużyny i sprzyja znajdowaniu rozwiązań bardziej odpornych na kontry typów.

Dodatkowymi założeniami, które powstały w celu szybszego i łatwiejszego przeszukiwania jest po pierwsze wykluczenie pokemonów legendarnych ze zbioru wszystkich pokemonów. Są bardzo potężne i najprawdopodobniej dominowałyby wszystkie inne, a w grach gracz uzyskuje do nich dostęp w końcowym etapie gry (tzw. „endgame”), gdy już wszystkie znaczące pojedynki ma ze sobą, więc takie rozwiązanie byłoby nieprzydatne, bo musiałby je jakoś wcześniej z pomocą jakiejś drużyny złapać.

Po drugie wykluczenie wszystkich pokemonów, które nie są ostatecznymi ewolucjami ze zbioru. Niektóre pokemony posiadają jedną lub dwie formy, przez które muszą przejść zanim będą mogłyby ewoluować w ostateczną (np. Charmander -> Charmeleon -> Charizard), które to są zawsze słabsze od niej. W związku z tym używanie tych słabszych form jest bez sensu, gdyż gracz i tak zawsze dąży do uzyskania wersji ostatecznej, aby być potężniejszym (wyjątkiem jest tutaj tylko serial, w którym bohater przez cały czas trwania serii nie ewoluuje swojego Pikachu w Raichu). Nawet gdy rozwiązanie zwróci mu Charizarda, to i tak wie, że musi zacząć od Charmandera.

Używany zbiór danych nie posiada pola identyfikującego czy dany pokemon jest końcową formą, więc aby nie robić tego ręcznie posłużyliśmy się innym zbiorem dostępnym w Internecie.

Dane: https://github.com/Laetitia-Deken/Pokemon_Dataset_Exploratory

Przygotowane rozwiązanie umożliwia wyłączenie tych ograniczeń (unikalności typów, legendarnych pokemonów i wcześniejszych form), ale to w ramach ciekawostki i jako pole do ewentualnych analiz.

Podczas tworzenia rozwiązania pominęliśmy częstotliwość występowania danych gatunków pokemonów, gdyż z perspektywy gracza, który chciałby użyć naszego rozwiązania jest to bez sensu i tylko zaburzy to wynik. Większość jak nie wszystkie pokemony w danym regionie (nie licząc legendarnych) są dostępne przez końcową fazą gry więc jak rzadkie by nie były gracz i tak skompletuje zespół jaki będzie chciał, nawet gdyby musiał spędzić dużo czasu w wysokiej trawie, jaskiniach czy akwenach (w tych miejscach w grach występują dzikie pokemony). Liczy się siła pokemonów, a nie czas w jakim zostaną pozyskane, a osoby robiące speedruny są w zdecydowanej mniejszości. Gracz nie potrzebuje od razu całej szóstki. Gra jest skonstruowana tak, żeby mógł zacząć od jednego i bez większych problemów progresował dalej.

Symulacja walki

Na potrzeby zadania symulacja walki została znacząco uproszczona. Odwzorowanie mechaniki walk znanej z gier w pełnym zakresie wymagałoby zastosowania znacznie bardziej zaawansowanych metod oraz użycia zupełnie innego zbioru danych. W szczególności konieczne byłoby uwzględnienie poziomów pokemonów, dostępnych ruchów, ich typów i mocy, a także mechanik takich jak zmiany statystyk, efekty statusowe czy użycie przedmiotów przez trenera.

Dla każdego pokemona musielibyśmy dodatkowo dobrać czteroelementowy zestaw ruchów, spośród kilkudziesięciu dostępnych oraz opracować algorytm decyzyjny sterujący przebiegiem walki. Tego typu podejście znacząco zwiększałoby złożoność problemu i wykraczałoby poza zakres niniejszego zadania.

Dlatego ograniczyliśmy się do brania po uwagę podstawowych statystyk pokemonów dostępnych w zbiorze takich jak atak, obrona, atak specjalny, obrona specjalna, zdrowie oraz szybkość, a także oczywiście typy i dana skuteczność przeciwko typowi.

W grze istnieje mechanizm walk 2 vs 2, aczkolwiek w naszym rozwiązaniu pomijamy tę mechanikę i bierzemy pod uwagę tylko walki 1 vs 1 zespołów z sześcioma pokemonami.

Przebieg walki

Walkę rozpoczynamy od zdecydowania, który z pokemonów na pierwszej, startowej pozycji jest szybszy za pomocą statystyki szybkości. Gdy już wiemy to pokemony atakują się naprzemiennie zaczynając od szybszego, aż któryś nie zostanie pokonany. W tym przypadku trener pokonanego pokemona wybiera tego na następnej pozycji i cykl rozpoczyna się od początku (decydujemy kto pierwszy i walka).

Ze względu na specyfikę symulacji i formuły jakich używamy trzeba było zastosować jeszcze kilka uproszczeń między innymi:

1. **Brak ruchów innych niż atakujące i przedmiotów**
   Nie bierzemy pod uwagę tego, że pokemon może użyć ruchu, który nie zadaje obrażeń (wzmocni swoje lub obniży statystyki przeciwnika) czy zmieni otoczenie, ani że trener może użyć jakiegoś przedmiotu (np. leczącego). Mamy tylko atak każdej ze stron. Każdy z ruchów jest jednakowy i zadaje obrażenia na podstawie statystyk pokemonów i ich skuteczności na dane typy.

2. **Minimalne obrażenia**
   Jeżeli obliczone obrażenia byłyby bardzo małe ($0 < \text{obrażenia} < 1$) to zaokrąglamy to do 1, aby walka mogła się skończyć w rozsądnym czasie. Może się tak zdarzyć ze względu na mnożnik skuteczności typów.

3. **Sytuacja patowa**
   Jeżeli zadane obrażenia wynosiłyby 0, a może się tak zdarzyć ze względu na mnożnik skuteczności typów, to aby uniknąć, że nasze ataki będą nieskuteczne, aż do porażki pokemona, wymieniamy go na kolejnego. Jeżeli nie mielibyśmy możliwości wymiany (to ostatni pokemon w drużynie) to jest to sytuacja patowa i uznajemy wtedy wygraną przeciwnika. Dzieje się tak dlatego, że w grze mamy jeszcze ruchy, które mają swój typ i to głównie na nich i na typach atakowanego pokemona opieramy obrażenia (typ atakującego może ewentualnie wzmocnić ruch, jeżeli ma ten sam typ), więc przez tą większą różnorodność (pokemon może się na uczyć ruchów różnych typów) taka sytuacja patowa praktycznie nie występuje. W przypadku porażki aktualnej drużyny w ten sposób jej punkty zdrowia potrzebne do funkcji oceny są zerowane.

4. **Maksymalna liczba tur**
   Podczas testów algorytm potrafił się zawieszać podczas walki, ze względu na małe obrażenia, przez co walka ciągnęła się przez bardzo długi czas. Dlatego postanowiliśmy nadać jej ograniczenie w postaci maksymalnej liczby tur, która domyślnie wynosi 1000. Jeżeli po upływie tego limitu nasza drużyna, ma mniej lub tyle samo zdrowia co przeciwnik to jest ono zerowane, a rezultat walki to porażka.

Formuły

Rozwiązanie umożliwia użycie różnych formuł do obliczania obrażeń i mnożnika skuteczności typów, lecz w badaniach zdecydowaliśmy się na ten najbardziej zbliżony do oryginalnego wzoru, który używany jest w grach bez ustalonych przez nas uproszczeń. W związku z tym wzór prezentuje się następująco:

damage = A / D * type1 * type2,

gdzie

A – suma ataku i ataku specjalnego atakującego

D – suma obrony i obrony specjalnej atakowanego

type1 – skuteczność atakującego przeciwko pierwszemu typowi atakowanego

type2 – skuteczność atakującego przeciwko drugiemu typowi atakowanego

Funkcja oceny

Początkowym założeniem dotyczącym funkcji oceny oraz ogólnego sposobu rozwiązania problemu było poruszanie się po przestrzeni rozwiązań w oparciu o sąsiedztwo drużyn. Dla każdej rozważanej drużyny planowaliśmy przeprowadzać symulacje walki ze wszystkimi możliwymi permutacjami jej sąsiadów, a następnie obliczać średni procent zdrowia pozostały naszej drużynie po tych walkach. Algorytm miał przemieszczać się w kierunku tych przeciwników, którzy odbierali naszej drużynie największą ilość zdrowia.

Takie podejście po rozpoczęciu prac okazało się jednak problematyczne. Przede wszystkim uwzględnienie wszystkich permutacji sąsiadów znacząco wydłużało czas działania solverów. Nawet po rezygnacji z pełnego przeglądu permutacji czas obliczeń pozostawał bardzo duży, co w praktyce uniemożliwiało uzyskanie rozwiązania w rozsądnym czasie.

Dodatkowo podejście to prowadziło do zmiany definicji rozwiązywanego problemu. Zamiast poszukiwania globalnie najlepszej drużyny, algorytmy optymalizowały jedynie rozwiązanie lokalnie najlepsze w obrębie aktualnego sąsiedztwa. W efekcie funkcja celu nie była stabilna ani obiektywna, drużyna dobrze radząca sobie wśród swoich sąsiadów mogła wypadać znacznie gorzej w konfrontacji z innymi przeciwnikami. W praktyce prowadziło to do sytuacji, w której algorytm (np. ewolucyjny) przedwcześnie zatrzymywał się w punkcie lokalnie dobrym, mimo że istniały znacznie lepsze rozwiązania globalne.

W związku z powyższym zdecydowaliśmy się na zmianę podejścia do definiowania funkcji oceny. Przy każdym uruchomieniu solvera generowana jest ustalona liczba drużyn przeciwników (liczba ta stanowi hiperparametr danego solvera), z którymi przeprowadzane są symulacje walki. Na podstawie wyników tych walk obliczana jest ocena danej drużyny. Liczba przeciwników musi być na tyle duża, aby uzyskana ocena była możliwie obiektywna i niezależna od pojedynczych, losowo wygenerowanych

drużyn, a jednocześnie na tyle mała, aby czas obliczeń pozostał akceptowalny. W świecie idealnym używalibyśmy wszystkich możliwych do ułożenia drużyn składających się z sześciu pokemonów. Porównywanie wszystkich rozważanych drużyn na tym samym zestawie przeciwników pozwala uzyskać spójny i porównywalny wskaźnik jakości rozwiązania.

Ostatecznie funkcja celu została przez nas zdefiniowana jako **średnia procentów pozostałego zdrowia naszej drużyny po walkach z ustalonym zestawem przeciwników** przy poprzedniej definicji przestrzeni przeszukiwań. Czyli mamy zadanie maksymalizacji i zbiór wartości funkcji to <0, 1>.

Warto podkreślić, że zastosowane podejście ma charakter niedeterministyczny, co oznacza, że przy kolejnych uruchomieniach solverów bardzo możliwe, a niemal pewne jest uzyskanie różnych drużyn o zbliżonej wartości funkcji celu. Takie zachowanie jest jednak w pełni akceptowalne i zgodne z naturą problemu. Przestrzeń rozwiązań jest bardzo duża, a liczba potencjalnie dobrych kombinacji pokemonów znaczna, przez co nie istnieje jedno, jednoznacznie najlepsze rozwiązanie. Ostateczny wybór drużyny zależy również od preferencji gracza, takich jak preferowany styl rozgrywki, poziom ryzyka czy podatność na określone typy przeciwników. Zaproponowane rozwiązanie umożliwia więc znalezienie wielu alternatywnych, jakościowo porównywalnych drużyn, zamiast jednego sztywnego i uniwersalnego rozwiązania. Tym samym możemy stwierdzić, że nasza funkcja celu posiada bardzo dużą liczbę optimów lokalnych, które bardzo, ale to bardzo zbliżone będą do optimum globalnego przez co jego dokładne znalezienie niestety graniczy z cudem.

Opis rozwiązania

Do stworzenia rozwiązania zadania został wykorzystany język Python wraz z jego biblioteką standardową oraz kilkoma innymi do obsługi danych ze zbioru. Są to:

- **numpy** – do operacji matematycznych,

- **pandas** – do operacji na zbiorach danych,

- **pandera** – do walidacji zbiorów danych,

- **pydantic** – do walidacji klas, w naszym przypadku do walidacji parametrów solverów,

- **matplotlib** – do tworzenia wykresów,

Dodatkowo użyliśmy narzędzi formatowania i analizy statycznej kodu takich jak **Ruff** i **MyPy**, aby kod był jak najbardziej czytelny i najwyższej jakości.

Moduły

Rozwiązanie składa się z kilku modułów będących kluczowymi elementami całości. Są to:

Moduł danych

Moduł danych odpowiada za załadowanie i walidację naszego zbioru pokemonów, abyśmy mogli uzyskać podstawę do tworzenia drużyn niezawierającą żadnych błędów. Uwzględnia wyłączenie lub wyłączenie wcześniej wspomnianych ograniczeń co do zbioru danych.

Moduł drużyny

Moduł drużyny jest swojego rodzaju nakładką na wektor drużyny pokemonów, który zawiera wszelkie potrzebne metody, który zawiera wszelkie potrzebne walidacje i metody pozwalające na symulację walki i poruszanie się po przestrzeni przeszukiwań.

Moduł symulacji

Moduł symulacji zawiera wszelkie potrzebne funkcje do przeprowadzenia symulacji walki w tym między innymi formuły obliczania obrażeń i mnożnika skuteczności typów, które mają za zadanie wyciągnięcie danych potrzebnych do obliczenia funkcji oceny.

Moduł solverów

Zawiera solvery dla algorytmów wykorzystywanych do znalezienia optymalnej wartości funkcji oceny w tym algorytmu ewolucyjnego i symulowanego wyżarzania oraz funkcje potrzebne do ich analizy.

Moduł eksperymentów

Zawiera funkcje potrzebne do przeprowadzenia badań oraz eksperymentów i przetworzenia ich wyników.

Przeprowadzone eksperymenty i badania

Do przeprowadzenia eksperymentów i badań użyliśmy dwóch algorytmów, ewolucyjnego oraz symulowanego wyżarzania.

Algorytm Ewolucyjny

Przygotowany algorytm to klasyczny algorytm ewolucyjny dostosowany pod zespoły pokemonów. Nie posiada w sobie mechanizmu krzyżowania ze względu na to, że przy krzyżowaniu skomplikowane byłoby zapewnienie unikalności typów i mogłoby to destabilizować algorytm, dlatego jego głównym mechanizmem napędowym jest mutacja. Posiada on w sobie również selekcję turniejową jak i mechanizm elity (określona najlepsza liczba osobników zostaje nietknięta).

Solver algorytmu posiada następujące hiperparametry:

population_size (int) - rozmiar populacji (domyślnie 5),

elite_size (int) – rozmiar wspomnianej wcześniej elity (domyślnie 1),

generations (int) – liczba generacji (domyślnie 20),

tournament_size (int) – liczba osobników w turnieju w selekcji turniejowej (domyślnie 3),

mutation_rate (float) – prawdopodobieństwo wystąpienia mutacji (domyślnie 0.6),

mutation_replacements (int) – liczba pokemonów do podmiany w drużynie w przypadku wystąpienia mutacji (domyślnie 2),

opponents_limit (int) – liczba przeciwników, na których będziemy oceniać nasze drużyny (domyślnie 100),

unique_types (boolean) – czy ograniczenie unikalności typów w drużynie jest aktywne (domyślnie true),

Domyślne parametry solvera zostały dobrane w sposób losowy (a raczej intuicyjny, bo nie wygenerował ich rng) w taki sposób, aby czas wykonywania się algorytmu był sensowny i nie zbliżał się do nieskończoności, ale wyniki były reprezentatywne.

Dodatkowo przy samym uruchomieniu solvera możemy ustawić odpowiednią formułę wyliczania obrażeń, jednakże domyślna to ta przedstawiona w wyższych paragrafach i ta będzie używana przy badaniach.

Porównanie z rozwiązaniem naiwnym

Podstawowym kryterium sukcesu w przypadku takich zadań powinno być to czy solver jest lepszy od rozwiązania naiwnego. Jeśli jest gorszy to mamy pewność, że idziemy w złym kierunku i trzeba zmienić model lub go odpowiednio nastroić. W związku z tym pierwszym przeprowadzonym eksperymentem będzie porównanie z rozwiązaniem naiwnym. Jako rozwiązanie naiwne przyjmujemy solver zwracający losowe drużyny oceniane na tym samym zestawie przeciwników co solver algorytmu ewolucyjnego. Eksperymenty przeprowadziliśmy dla domyślnych parametrów (testy związane z parametrami pojawią się w dalszej części) z ośmioma uruchomieniami.

Wyniki testu znajdują się w poniższych tabelach.

Numer próby	Solver Naiwny – wartość funkcji oceny	Solver EA – wartość funkcji oceny	Solver Naiwny – suma statystyk drużyny	Solver EA – suma statystyk drużyny
1	0.1101	0.2770	2202	2206
2	0.0164	0.3680	2024	1765
3	0.0620	0.3509	1991	2299
4	0.1846	0.2734	2239	1951
5	0.0124	0.2831	2001	2173
6	0.0876	0.3759	2184	2229
7	0.0568	0.3052	2019	1995
8	0.1974	0.3084	2025	2382

Solver	Średnia ocen	Mediana ocen	Odchylenie standardowe ocen	Najlepsza drużyna	Najlepsza ocena	Suma statystyk najlepszej drużyny
Naiwny	0.0909	0.0748	0.0654	'Accelgor', 'Jumpluff', 'Snorlax', 'Rampardos', 'Jellicent', 'Camerupt'	0.1974	2025
EA	0.3178	0.3068	0.0389	‘Exeggutor’, ‘Gyarados’, ‘Throh’, ‘Blissey’, ‘Ampharos’, ‘Hippowdon’	0.3759	2229

Jak widać po powyższych wynikach solver EA okazał się zdecydowanie skuteczniejszy od rozwiązania naiwnego, ponieważ w każdym przypadku przy tych samych przeciwnikach, na których liczyliśmy ocenę jest ona zdecydowanie wyższa. Postawione kryterium sukcesu zostało spełnione, co oznacza, że idziemy w dobrą stronę. Oceny EA były w 5 z 8 przypadków o rząd wielkości większe. Średnia procentów zachowanych na koniec starcia punktów zdrowia drużyn zwróconych przez rozwiązanie naiwne to mniej niż 1%, podczas gdy przy EA mamy ich ponad 30, co dogłębnie pokazuje tę przepaść. Odchylenie standardowe również jest większe co pokazuje, że rozwiązania losowe są dużo bardziej od siebie rozrzucone, podczas gdy rozwiązania EA jest bardziej zbliżone (w kontekście wartości funkcji oceny).

Najlepsza z ocen zwróconych przez EA to 0.3760 w próbie 8, czyli średni pozostały procent zdrowia po walce to 37,6% co jest wynikiem całkiem dobrym, aczkolwiek przy innym uruchomieniu i zmianie hiperparametrów solvera na pewno dałoby się go poprawić i wycisnąć z niego maksymalne możliwości. Mimo wszystko wartości funkcji oceny per uruchomienie trochę się różnią i przydałoby się to ustabilizować. Będzie to

przedmiotem dalszych rozważań i badań. Najwyższy wynik rozwiązania naiwnego to 19,74%, co daje nam 17,86 punktów procentowych różnicy, co jest różnicą znaczną.

Zwrócona najlepsza drużyna (dla EA) posiada typy odpowiednio:

Exeggutor – psychiczny i trawiasty

Gyarados – wodny i latający

Throh – walczący

Blissey – normalny

Ampharos – elektryczny

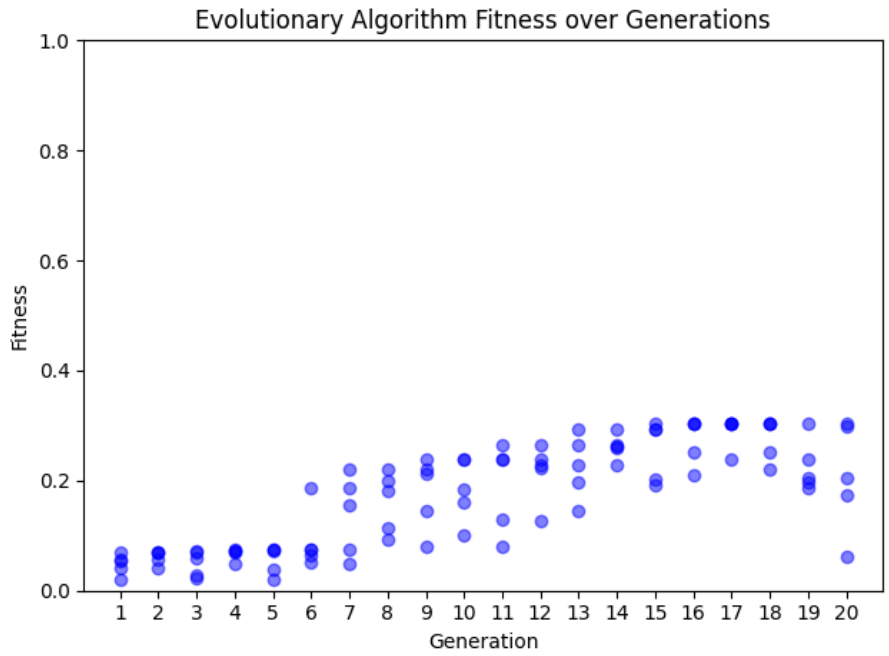
Hippowdon – ziemny

Jak widać nie mamy wszystkich typów pokemonów, bo byłoby to niemożliwe (typów mamy 18) i niestety nie mamy kontry na wszystko, co jest kosztem uproszczonej symulacji. W grach mamy jeszcze ruchy, które mają swoje typy i to one są realną kontrą przeciwko innym pokemonom. Jednakże i tak taka drużyna poradzi sobie w wielu sytuacjach co pokazują wyniki.

Bardzo ciekawym zjawiskiem, które można zaobserwować, jest to, że suma statystyk drużyn (ataki i obrony) nie zawsze determinuje to, kto jest lepszy. Dla przykładu w próbie drugiej, pomimo że drużyna znaleziona przez EA spisała się dużo, dużo lepiej (ponad 30 punktów procentowych różnicy) to ma znacząco mniejszą sumę statystyk, bo aż o 259, podobną sytuację mamy w próbie czwartej i ósmej. Pokazuje to tylko, że statystyki pokemonów nie są najważniejsze, a ich pozycja i szybkość jest również bardzo istotna, a w realnej grze oczywiście taktyka trenera.

Analiza przebiegu działania algorytmu

Poniżej przedstawiono wykres ocen od generacji dla pojedynczego uruchomienia solvera z domyślnymi parametrami.



Jak widać rozpoczynamy od bardzo niskich wartości funkcji oceny (zgodnie z tym co zwracało rozwiązanie naiwne) i utrzymują się one do szóstej generacji. Najprawdopodobniej podczas pierwszych pięciu mutacja zadziałała w sposób niezadowalający, a przez mechanizm elity najlepsze rozwiązanie trzymane było w populacji więc funkcja pozostała w dołku. Dopiero w generacji szóstej mutacja dokonała „dobrego strzału" i znalazła wyjątkowo dobrą kombinację z funkcją oceny na poziomie ok. 0,2. Później do generacji piętnastej bardzo stopniowo zwiększała się i poprzez wcześniej wspomniany mechanizm elity już do końca pozostała na jednym poziomie ok. 0,3, ponieważ algorytm nie znalazł już nic lepszego.

Najlepsza drużyna - 'Conkeldurr', 'Golurk', 'Tyranitar', 'Kangaskhan', 'Vespiquen', 'Lanturn'

Ocena - 0.3026

Suma statystyk - 2240

Początkowa populacja:

'Sudowoodo', 'Heatmor', 'Silvally', 'Hitmontop', 'Electrode', 'Donphan' - 0.0528 - 1999

'Steelix', 'Aromatisse', 'Bibarel', 'Ledian', 'Unown', 'Arcanine' - 0.0400 - 1957

'Infernape', 'Porygon-Z', 'Mimikyu', 'Blastoise', 'Bastiodon', 'Masquerain' - 0.0557 - 2217

'Oranguru', 'Comfey', 'Palossand', 'Leavanny', 'Gorebyss', 'Dragalge' - 0.0698 - 2130

'Starmie', 'Spiritomb', 'Ferrothorn', 'Sudowoodo', 'Marowak', 'Porygon-Z' - 0.02137 – 2130

Końcowa populacja:

'Conkeldurr', 'Golurk', 'Tyranitar', 'Kangaskhan', 'Vespiquen', 'Lanturn' - 0.3030 - 2240

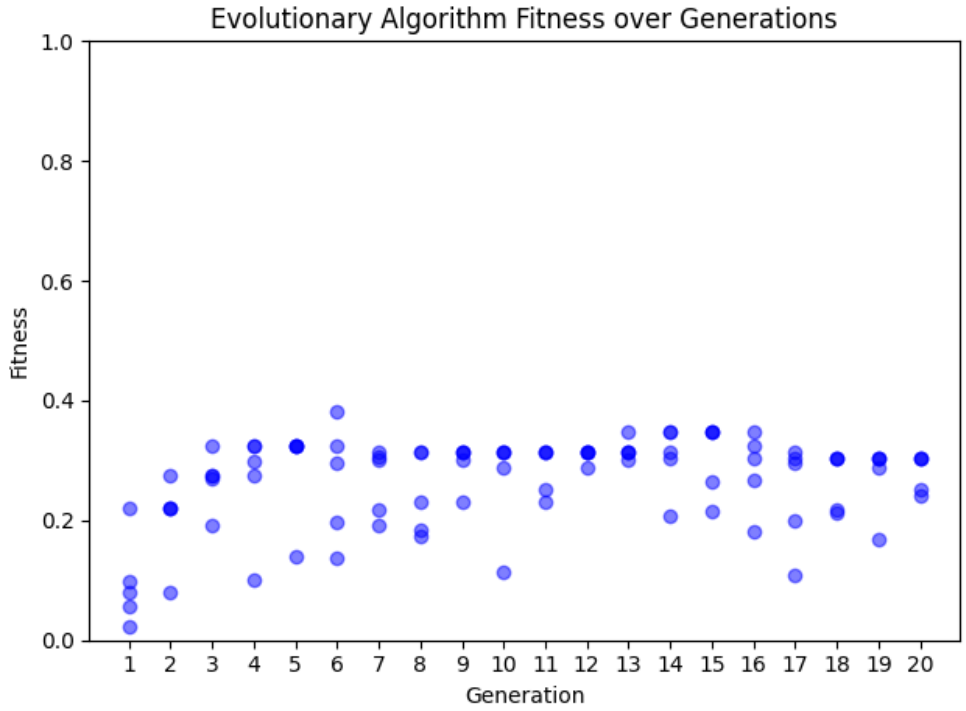
'Simisear', 'Glaceon', 'Gorebyss', 'Kangaskhan', 'Vespiquen', 'Druddigon' - 0.0610 - 2204

'Chimecho', 'Golurk', 'Tyranitar', 'Watchog', 'Vespiquen', 'Lanturn' - 0.2053 - 2098

'Conkeldurr', 'Watchog', 'Tyranitar', 'Amoonguss', 'Vespiquen', 'Lanturn' - 0.3000 - 2119

'Conkeldurr', 'Golurk', 'Malamar', 'Kangaskhan', 'Vespiquen', 'Slurpuff' - 0.1728 – 2092

Poniżej znajduje się wykres ocen algorytmu, ale z wyłączonym mechanizmem elity, aby sprawdzić jak algorytm będzie zachowywał się bez niego.



Jak widać nie mamy już sytuacji, w której najlepsze rozwiązanie pozostaje ciągle na tym samym poziomie. W tym przypadku wyjątkowo otrzymaliśmy dobrą kombinację na początku i ewoluowała ona do jeszcze lepszego rozwiązania w trzeciej generacji. Później przez selekcję rozwiązanie przetrwało (wygrywało turniej), aż do szóstej generacji, ale któryś z innych punktów zmutował i znalazł najlepsze rozwiązanie, którego ocena jest bardzo blisko 0,4. W późniejszym etapie algorytm dalej szuka rozwiązania, niektóre są nawet zadowalające, ale niestety nie znalazł nic lepszego, aczkolwiek oddalił się od bardzo niskich ocen, które wygenerowała pierwsza populacja co jest bardzo dobrym zjawiskiem.

Najlepsza drużyna - 'Snorlax', 'Salamence', 'Gigalith', 'Wobbuffet', 'Lapras', 'Galvantula'

Ocena - 0.3812

Statystyki – 2061

Jak widać, ocena jest znacznie lepsza niż w pierwszym przypadku, ale nie można stwierdzić czy to przypadek czy reguła ze względu na to, że to tylko pojedyncze uruchomienie solvera, no i mieliśmy trochę innych przeciwników.

Hiperparametry

Niestety, ale niemożliwym jest wystrojenie solvera, tak aby miał optymalne hiperparametry w sposób profesjonalny np. poprzez przeszukiwanie po hipersiatce, przeszukiwanie losowe czy hyperband w sensownym czasie. Takie podejścia zajęłyby dni. Mamy bardzo dużą przestrzeń przeszukiwań, drogą obliczeniowo funkcję oceny, ograniczone zasoby sprzętowe i brak dedykowanego sprzętu do wykonywania obliczeń, więc rozważania dotyczące hiperparametrów ograniczą się do prezentacji zachowania algorytmu przy ich zmianach na niektóre wartości przy ograniczonej liczbie uruchomień.

Jednym z najważniejszych hiperparametrów tego solvera jest liczba przeciwników, z którymi toczymy pojedynki, ponieważ to wyniki tych walk składają się na naszą ocenę.

Gdyby liczba przeciwników była bardzo mała to oceny byłyby bardzo wysokie, ze względu na dostosowanie się algorytmu i po prostu małej liczby do średniej, ale to tylko pozornie. Może i oceny byłyby wysokie, jednakże w grze taka drużyna byłaby najprawdopodobniej bezużyteczna, bo co nam po drużynie, która wiemy, że na pewno pokona 5 przeciwników. Dlatego ważne jest zasymulowanie jak największej ilości starć z którymi moglibyśmy spotkać się w grze. Ze względu na ograniczony czas i zasoby pozostaniemy przy 100 w celach prezentacyjnych. Gdybyśmy chcieli rzeczywiście znaleźć najlepszą drużynę bez żadnego ryzyka to musielibyśmy użyć wszystkich możliwości, które idą w setki tysiący. Jednakże oczywiście solver umożliwia takie ustawienia.

Hiperparametry – liczba generacji i rozmiar elity

Pierwszym parametrem do rozważenia będzie liczba generacji. Sprawdzimy, czy jeśli damy algorytmowi więcej czasu do poszukiwania to wyniki będą lepsze. Testy zostały przeprowadzone dla czterech uruchomień. Dodatkowo postaramy się odpowiedzieć jaki wpływ na algorytm ma mechanizm elity.

Liczba Generacji	Elita	Średnia Ocen	Mediana	Odchylenie standardowe	Najlepsza ocena	Suma statystyk najlepszej drużyny
20	1	0.3818	0.3816	0.0423	0.4373	1730
20	0	0.3331	0.3107	0.0549	0.4243	2230
30	1	0.3209	0.3223	0.0156	0.3376	2171

30	0	0.3920	0.3796	0.0374	0.4513	1994
40	1	0.4117	0.4143	0.0404	0.4661	2332
40	0	0.3445	0.3345	0.0287	0.3926	2046
50	1	0.3680	0.3682	0.0561	0.4468	1917
50	0	0.4143	0.4082	0.0345	0.46470	1723

Z wyników pokazanych w tabeli wynika, że nie zawsze ze zwiększaniem się liczby generacji mamy zwiększanie się średniej ocen, aczkolwiek przy najlepszej ocenie można zauważyć trend wzrostowy. Tak samo mechanizm elity zdaje się nie mieć, aż takiego znaczenia (jeśli mamy 0 lub 1). Trzyma po prostu jeden bezpieczny najlepszy punkt, aby go nie zgubić kosztem eksploracji przestrzeni. Odchylenia standardowe utrzymują się na podobnym poziomie. Takie wyniki mogą być następstwem małej liczby uruchomień, ale niestety przy większej ilości robienie pomiarów zajęłoby dni.

Tak więc w celu podniesienia poziomu optymalności wyników można by podnieść delikatnie liczbę generacji, ale nie można z tym przesadzić, aby pomiary kończyły się w sensownym czasie.

Dodatkowo przeprowadzone zostały testy już pod samą wielkość elity, dla 6 osobników w populacji, od zera do trzech.

Elita	Średnia Ocen	Mediana	Odchylenie standardowe	Najlepsza ocena	Suma statystyk najlepszej drużyny
0	0.2856	0.3035	0.0421	0.3215	1937
1	0.3043	0.3030	0.0466	0.3691	2163
2	0.3000	0.2997	0.0426	0.3598	2220

3	0.2520	0.2578	0.0252	0.2809	2240

Najlepsza drużyna to 'Slowbro', 'Hariyama', 'Blissey', 'Salamence', 'Arcanine', 'Leavanny' i wystąpiła ona dla elity o rozmiarze 1. Jak widać rozmiar elity ma jednak jakieś znaczenie. Ustawienie elity na poziomie 1 i 2 wspomogło trochę wynik, jednakże już przy połowie populacji znacząco spadł. Ma to sens ze względu na to, że zmniejsza to eksplorację algorytmu, nie ma mutacji i nie mamy możliwości przemieszczenia się w lepsze miejsca. W porównaniu do eksperymentu z liczbą generacji wyniki są bardzo słabe.

Hiperparametry – współczynnik mutacji

Kolejnym z hiperparametrów, jaki jesteśmy w stanie przetestować jest współczynnik mutacji. Określa on jak często dana drużyna ulega mutacji (podmianie pokemonów na inne).

W ramach eksperymentu przeprowadziliśmy badanie dla domyślnych parametrów eksperymentując z współczynnikiem mutacji na poziomach 0.1, 0.3, 0.5, 0.7 i 0,9.

Poniżej znajdują się wyniki uzyskane dla każdego z wcześniej wymienionych współczynników. Warto dodać, że każdy z eksperymentów, był przeprowadzany na tym samym zestawie przeciwników.

Mutacja = 0.1

run	solver	elite_size	fitness	stats_sum	pokemons
0	EA	1	0.12992322456813818	1997	PokemonTeam(size=6, names=['Tangrowth', 'Crobat', 'Rapidash', 'Gastrodon', 'Jynx', 'Pangoro'])
1	EA	1	0.3336452513966481	2073	PokemonTeam(size=6, names=['Abomasnow', 'Alomomola', 'Aromatisse', 'Steelix', 'Wobbuffet', 'Bouffalant'])
2	EA	1	0.23924670433145007	2180	PokemonTeam(size=6, names=['Hypno', 'Blaziken', 'Slaking', 'Nidoking', 'Octillery', 'Ribombee'])
3	EA	1	0.2525688073394496	2235	PokemonTeam(size=6, names=['Flareon', 'Pinsir', 'Slaking', 'Reuniclus', 'Lilligant', 'Crobat'])
4	EA	1	0.27978984238178634	2446	PokemonTeam(size=6, names=['Donphan', 'Aromatisse', 'Slowbro', 'Abomasnow', 'Kangaskhan', 'Goodra'])
5	EA	1	0.288301282051282	2027	PokemonTeam(size=6, names=['Incineroar', 'Dunsparce', 'Dhelmise', 'Gigalith', 'Alomomola', 'Stunfisk'])

solver	mean	median	std	min	max	count
EA	0.253913	0.266179	0.068992	0.129923	0.333645	6

Mutacja = 0.3

run	solver	elite_size	fitness	stats_sum	pokemons
0	EA	1	0.28139158576051776	2230	PokemonTeam(size=6, names=['Incineroar', 'Alomomola', 'Garchomp', 'Glalie', 'Kangaskhan', 'Skarmory'])
1	EA	1	0.26588235294117646	2233	PokemonTeam(size=6, names=['Blaziken', 'Sceptile', 'Donphan', 'Crustle', 'Meowstic', 'Snorlax'])
2	EA	1	0.21749532710280373	2202	PokemonTeam(size=6, names=['Watchog', 'Sylveon', 'Abomasnow', 'Swampert', 'Passimian', 'Solrock'])
3	EA	1	0.21460377358490565	2447	PokemonTeam(size=6, names=['Hydreigon', 'Steelix', 'Dhelmise', 'Lapras', 'Gallade', 'Bouffalant'])
4	EA	1	0.368594674556213	2293	PokemonTeam(size=6, names=['Durant', 'Blissey', 'Tyranitar', 'Hippowdon', 'Venusaur', 'Magmortar'])
5	EA	1	0.2757119741100324	2118	PokemonTeam(size=6, names=['Chesnaught', 'Torkoal', 'Excadrill', 'Azumarill', 'Snorlax', 'Goodra'])

solver	mean	median	std	min	max	count
EA	0.270613	0.270797	0.056078	0.214604	0.368595	6

Mutacja = 0.5

run	solver	elite_size	fitness	stats_sum	pokemons
0	EA	1	0.26944723618090455	2161	PokemonTeam(size=6, names=[`Vanilluxe', 'Blaziken', 'Goodra', 'Slurpuff', 'Stunfisk', 'Alomomola`])
1	EA	1	0.25489932885906036	2090	PokemonTeam(size=6, names=['Tentacruel', 'Emboar', 'Aurorus', 'Lilligant', 'Audino', 'Reuniclus'])
2	EA	1	0.3332592592592593	2124	PokemonTeam(size=6, names=[`Hydreigon', 'Heracross', 'Audino', 'Wobbuffet', 'Meganium', 'Vaporeon`])
3	EA	1	0.34480620155038755	2075	PokemonTeam(size=6, names=[`Slaking', 'Aurorus', 'Tropius', 'Luxray', 'Excadrill', 'Toxicroak`])
4	EA	1	0.3877023498694517	1862	PokemonTeam(size=6, names=[`Rampardos', 'Toxicroak', 'Blissey', 'Granbull', 'Wailord', 'Vanilluxe`])
5	EA	1	0.2736516853932584	2244	PokemonTeam(size=6, names=['Stunfisk', 'Tyranitar', 'Mantine', 'Magmortar', 'Oranguru', 'Leavanny'])

solver	mean	median	std	min	max	count
EA	0.310628	0.303455	0.052516	0.254899	0.387702	6

Mutacja = 0.7

run	solver	elite_size	fitness	stats_sum	pokemons
0	EA	1	0.2695757575757576	1980	PokemonTeam(size=6, names=[`Hippowdon', 'Lickilicky', 'Alomomola', 'Swalot', 'Crabominable', 'Meganium`])
1	EA	1	0.39727659574468077	2011	PokemonTeam(size=6, names=['Gogoat', 'Arcanine', 'Absol', 'Machamp', 'Blissey', 'Slurpuff'])
2	EA	1	0.2457820738137083	2147	PokemonTeam(size=6, names=[`Excadrill', 'Blastoise', 'Lickilicky', 'Comfey', 'Conkeldurr', 'Amoonguss`])
3	EA	1	0.3347519582245431	1992	PokemonTeam(size=6, names=[`Blissey', 'Arcanine', 'Wailord', 'Forretress', 'Aromatisse', 'Kommo-o`])
4	EA	1	0.3488286969253295	1950	PokemonTeam(size=6, names=['Blissey', 'Gigalith', 'Mamoswine', 'Tsareena', 'Skuntank', 'Durant'])
5	EA	1	0.40704581358609787	2381	PokemonTeam(size=6, names=['Tyranitar', 'Slaking', 'Vanilluxe', 'Musharna', 'Gastrodon', 'Eelektross'])

solver	mean	median	std	min	max	count
EA	0.333877	0.34179	0.065564	0.245782	0.407046	6

Mutacja = 0.9

run	solver	elite_size	fitness	stats_sum	pokemons
0	EA	1	0.389249617151608	2205	PokemonTeam(size=6, names=['Gogoat', 'Blaziken', 'Rhyperior', 'Reuniclus', 'Lapras', 'Bouffalant'])
1	EA	1	0.38270976616231084	1977	PokemonTeam(size=6, names=['Tangrowth', 'Solrock', 'Politoed', 'Swalot', 'Hydreigon', 'Blissey'])
2	EA	1	0.3915076923076923	2209	PokemonTeam(size=6, names=[`Rhyperior', 'Slaking', 'Skuntank', 'Gyarados', 'Granbull', 'Crabominable`])
3	EA	1	0.47253317249698434	2143	PokemonTeam(size=6, names=['Garchomp', 'Alomomola', 'Hawlucha', 'Tyranitar', 'Blissey', 'Gogoat'])
4	EA	1	0.4202005730659025	2068	PokemonTeam(size=6, names=['Toxicroak', 'Gyarados', 'Garchomp', 'Blissey', 'Grumpig', 'Kricketune'])
5	EA	1	0.41026206896551726	2088	PokemonTeam(size=6, names=['Granbull', 'Metagross', 'Victreebel', 'Mamoswine', 'Emboar', 'Blissey'])

solver	mean	median	std	min	max	count
EA	0.411077	0.400885	0.033241	0.38271	0.472533	6

Jak widać na zamieszczonych wynikach, wraz ze zwiększaniem parametru mutacji, rosną nam średnie oraz maksymalne wyniki dla poszczególnych runów. Częstsze mutacje, pozwalają algorytmowi na częstsze zmiany drużyn, tym samym wpływając na znalezienie bardziej optymalnych rozwiązań.

Wzrost średnich wartości podskoczył z wartości **0.25** aż do **0.41**, czyli mieliśmy przyrost **16** punktów procentowych.

Natomiast wzrost wartości funkcji ewaluacji dla najlepszej drużyny (między mutacją 0.1 a 0.9) wyniósł z **0.33** do **0.47, czyli skok o 14 punktów procentowych.**

Zatem słusznym stwierdzeniem może być uznanie tego współczynnika za istotny, zwłaszcza że jest on czynnikiem napędowym tego algorytmu. Wspieranie mechanizmu eksploracji w tym przypadku popłaca.

Hiperparametry - wielkość populacji

Podobne eksperymenty przeprowadzono dla parametru określającego wielkość populacji. Populacja wpływa na liczbę drużyn, które w danej iteracji ze sobą będą rywalizować. Testowaliśmy warianty 20 oraz 50, przy pozostawieniu pozostałych parametrów algorytmu na wielkościach domyślnych. Dla każdej wartości parametru przeprowadzono 6 eksperymentów i ewaluowano wyniki na tym samym zestawie oponentów.

Populacja = 20

run	solver	elite_size	fitness	stats_sum	pokemons
0	EA	1	0.39444444444444443	2461	PokemonTeam(size=6, names=['Slaking', 'Salazzle', 'Garchomp', 'Tyranitar', 'Slurpuff', 'Avalugg'])
1	EA	1	0.4950489510489511	2459	PokemonTeam(size=6, names=['Tyranitar', 'Heracross', 'Goodra', 'Slowbro', 'Blissey', 'Beartic'])
2	EA	1	0.4379156327543424	1876	PokemonTeam(size=6, names=['Tangrowth', 'Inclneroar', 'Blissey', 'Yanmega', 'Golem', 'Wobbuffet'])
3	EA	1	0.41108860759493665	1900	PokemonTeam(size=6, names=['Alomomola', 'Rhyperior', 'Blissey', 'Haxorus', 'Tropius', 'Luxray'])
4	EA	1	0.5094054054054054	2120	PokemonTeam(size=6, names=['Rhyperior', 'Abomasnow', 'Snorlax', 'Milotic', 'Arcanine', 'Wobbuffet'])
5	EA	1	0.4668187744458932	2000	PokemonTeam(size=6, names=['Tyrantrum', 'Abomasnow', 'Krookodile', 'Clefable', 'Drifblim', 'Blissey'])

solver	mean	median	std	min	max	count
EA	0.452454	0.452367	0.045938	0.394444	0.509405	6

Populacja = 50

run	solver	elite_size	fitness	stats_sum	pokemons
0	EA	1	0.45598958333333334	2064	PokemonTeam(size=6, names=['Blissey', 'Tyranitar', 'Kommo-o', 'Wobbuffet', 'Ninetales', 'Octillery'])
1	EA	1	0.4816986301369864	2201	PokemonTeam(size=6, names=['Torkoal', 'Slaking', 'Tyranitar', 'Donphan', 'Vaporeon', 'Wobbuffet'])
2	EA	1	0.5207317073170732	1925	PokemonTeam(size=6, names=['Rhyperior', 'Tangrowth', 'Passimian', 'Blissey', 'Togekiss', 'Alomomola'])
3	EA	1	0.45420138888888884	1898	PokemonTeam(size=6, names=['Donphan', 'Blissey', 'Wobbuffet', 'Tangrowth', 'Blastoise', 'Drifblim'])
4	EA	1	0.4761123595505618	1827	PokemonTeam(size=6, names=['Blissey', 'Sylveon', 'Goodra', 'Inclneroar', 'Wobbuffet', 'Alomomola'])
5	EA	1	0.4894948335246843	1920	PokemonTeam(size=6, names=['Blissey', 'Tyranitar', 'Wobbuffet', 'Volbeat', 'Wallord', 'Dragonite'])

solver	mean	median	std	min	max	count
EA	0.479705	0.478905	0.024531	0.454201	0.520732	6

Jeżeli spojrzymy na wartości średnie eksperymentów, to widać niewielki skok w wartości funkcji ewaluacyjnej. Średnia wyników wzrosła z 0.45 do 0.48, czyli zanotowaliśmy skok o 3 punkty procentowe. Było to do przewidzenia, gdyż zwiększając populację, zwiększamy liczbę potencjalnych rozwiązań, przez co przeszukujemy większą część przestrzeni przeszukiwań i mamy większą szansę na natrafienie na dobre drużyny. Trzeba jednak wziąć pod uwagę czas obliczeń podczas doboru tego parametru, gdyż zbyt duży wpłynie bardzo negatywnie na czas obliczeń. Najlepiej jest znaleźć optimum, które pozwoli nam na przeszukiwanie dużej przestrzeni w umiarkowanym czasie. Z doświadczenia, właśnie wielkość około 20 sprawowała się całkiem dobrze i w czasie prawie 3x krótszym znalazła drużynę, która jest tylko o 2 punkty procentowe gorsza, niż drużyna znaleziona przy populacji wynoszącej 50.

Najlepsza drużyna znaleziona przy eksperymentach z elitą:

Funkcja oceny: 0.3691

Drużyna: 'Slowbro', 'Hariyama', 'Blissey', 'Salamence', 'Arcanine', 'Leavanny'

Najlepsza drużyna znaleziona przy eksperymentach z mutacją:

Funkcja oceny : 0.4725

Drużyna: 'Garchomp', 'Alomomola', 'Hawlucha', 'Tyranitar', 'Blissey', 'Gogoat'

Najlepsza drużyna znaleziona przy eksperymentach z populacją:

Funkcja oceny: 0.5207

Drużyna : 'Rhyperior', 'Tangrowth', 'Passimian', 'Blissey', 'Togekiss', 'Alomomola'

Warto zauważyć, że w trzech przypadkach pojawia się ten sam pokemon, Blissey a w dwóch 'Alomomola', co może znaczyć, że są one dosyć uniwersalne.

Przeglądając poszczególne wyniki, często można natknąć się na te same nazwy. Oznacza to, że silnych uniwersalnych pokemonów jest dosyć sporo.

Algorytm symulowanego wyżarzania

Algorytm ten inspirowany jest procesem stygnięcia metali w metalurgii. Jego główną cechą jest **akceptacja gorszych rozwiązań z określonym prawdopodobieństwem.** Na początku procesu (wysoka "temperatura") algorytm jest bardzo elastyczny i często akceptuje gorsze ruchy, co pozwala mu swobodnie eksplorować całą przestrzeń i "wyskakiwać" z optimów lokalnych. W miarę upływu czasu temperatura spada, a algorytm staje się coraz bardziej rygorystyczny, ostatecznie zachowując się jak Algorytm wspinaczkowy, aby precyzyjnie dotrzeć do szczytu w obiecującym obszarze.

Niniejsza część dokumentacji stanowić będzie analizę algorytmu symulowanego wyżarzania. Głównym celem będzie wpływ działania hiperparametrów na jakość przeszukiwania oraz dodatkowo porównamy ten algorytm z:

- **przeszukiwaniem losowym** - służy jako punkt odniesienia, pozwalając ocenić, o ile bardziej zaawansowane metody przewyższają czysty przypadek
- **algorytmem wspinaczkowym** – reprezentuje klasyczne podejście zachłanne, skupione na lokalnej poprawie wyniku

Solver algorytmu posiada następujące hiperparametry:

T0(float) - tempratura startowa (domyślnie 0.5),

Tmin(float) – temperatura minimanla (domyślnie 1e-4),

alpha(float) – współczynnik chłodzenia(domyślnie 0.995),

Iters_per_temp (int) – ilość iteracji przed dokonaniem chłodzenia (domyślnie 3),

max_evaluations(int) – maksymalna ilość iteracji podczas rozwiazywania (domyślnie 200),

neighbour_replacement(int) – ilość zmienianych pokemonów po przegranej,

opponents_limit (int) – liczba przeciwników, na których będziemy oceniać nasze drużyny (domyślnie 100),

unique_types (boolean) – czy ograniczenie unikalności typów w drużynie jest aktywne (domyślnie true),

Wpływ temperatury na wyniki

Pierwsze eksperymenty pokazują jak bardzo temperatura wpływa na osiągane wyniki. Porównałem dwie sytuacje:

1. Wysoka temperatura i niska wartość kroku [t=0.75 i a=0.995]
2. Średnia temperatura i średnia wartość kroku [t=0.45 i a=0.6]

Sytuacja 1:

Run	Drużyna	Wartość funkcji oceny	Suma statystyk
1	'Snorlax','Accelgor','Sylveon','Zoroark','Hariyama','Jynx'	**0.2411**	**1910**
2	'Sceptile', 'Scizor', 'Slowking', 'Oricorio', 'Krookodile', 'Slurpuff'	**0.0986**	**2201**
3	'Tyranitar', 'Ambipom', 'Beedrill', 'Phione', 'Garchomp', 'Granbull'	**0.2383**	**2241**
4	'Dragonite', 'Wailord', 'Slaking', 'Torterra', 'Machamp', 'Weavile'	**0.2807**	**2168**
5	'Electrode', 'Mandibuzz', 'Turtonator', 'Golurk', 'Slowbro', 'Glaceon'	**0.0934**	**2188**
6	'Donphan', 'Dunsparce', 'Tyranitar', 'Jellicent', 'Toxicroak', 'Meowstic'	**0.2421**	**2089**
7	'Sunflora', 'Tyranitar', 'Salazzle', 'Simipour', 'Exploud', 'Cofagrigus'	**0.1603**	**2179**
8	'Passimian', 'Simisage', 'Garbodor', 'Musharna', 'Blissey', 'Slurpuff'	**0.3383**	**1849**

W powyższej tabeli można dostrzec ciekawą sytuację, że najlepsze wyniki osiągnęła drużyna, która miała sumarycznie najniższe statystyki. Dobrze to pokazuje, że w walkach pokemonów statystyki nie zawsze są najważniejsze i duży wpływ na walkę, mogą mieć typy pokemonów w drużynie i jak bardzo efektywne są one na oponentów.

Sytuacja 2 (porównanie z 80 drużynami):

Run	Drużyna	Wartość funkcji oceny	Suma statystyk
1	'Sableye', 'Snorlax', 'Gogoat', 'Whiscash', 'Salamence', 'Delphox'	0.2644	2238

2	'Kommo-o', 'Golisopod', 'Mudsdale', 'Avalugg', 'Tangrowth', 'Blissey'	0.3524	**2226**
3	'Phione', 'Tangrowth', 'Blissey', 'Krookodile', 'Primeape', 'Ninetales'	0.3157	**1885**
4	'Crabominable', 'Blissey', 'Umbreon', 'Carracosta', 'Weezing', 'Metagross'	**0.3058**	**2197**
5	'Blaziken', 'Rampardos', 'Amoonguss', 'Clawitzer', 'Magnezone', 'Blissey'	0.3623	**2115**
6	'Aurorus', 'Gogoat', 'Hippowdon', 'Milotic', 'Exploud', 'Durant'	0.2724	**2049**
7	'Abomasnow', 'Toxicroak', 'Blissey', 'Blastoise', 'Absol', 'Aerodactyl'	0.3419	**2269**
8	'Weezing', 'Goodra', 'Absol', 'Comfey', 'Whiscash', 'Blissey'	0.2867	2042

Sytuacja 2 (porównanie z 200 drużynami):

Run	**Drużyna**	**Wartość funkcji oceny**	**Suma statystyk**
1	'Tyranitar', 'Blaziken', 'Mudsdale', 'Vaporeon', 'Amoonguss', 'Musharna'	0.3807	2336
2	'Gogoat', 'Rhyperior', 'Ampharos', 'Blissey', 'Drifblim', 'Wailord'	0.5229	**1963**
3	'Tyranitar', 'Emboar', 'Hippowdon', 'Musharna', 'Slaking', 'Lapras'	0.4252	**2359**
4	'Hippowdon', 'Abomasnow', 'Ampharos', 'Swalot', 'Alomomola', 'Wobbuffet'	0.4363	**2053**
5	'Tyranitar', 'Amoonguss', 'Hippowdon', 'Snorlax', 'Alomomola', 'Hariyama'	0.4805	**2089**
6	'Blaziken', 'Tyranitar', 'Goodra', 'Mamoswine', 'Alomomola', 'Blissey'	0.50136	**2219**
7	'Pangoro', 'Ferrothorn', 'Swampert', 'Aerodactyl', 'Slaking', 'Wobbuffet'	0.3620	**2189**
8	'Mamoswine', 'Shiftry', 'Arcanine', 'Blissey', 'Hariyama', 'Drifblim'	0.4784	1798

Dla algorytmu wyżarzania warto zauważyć, że dla mniejszej temperatury, średnia wartość funkcji straty (zarówno dla sytuacji z 80 drużynami i 200 drużynami) była znacznie większa.

Średnia dla sytuacji 1: 0,2116

Średnia dla sytuacji 2 (80 oponentów): 0.310797

Średnia dla sytuacji 2 (200 oponentów): 0.4484

Wzrost średniej wartości funkcji w drugiej sytuacji wzrósł o **0.10 / 0.23** w zależności od ilości oponentów. Więc mniejsza temperatura w tym przypadku pozytywnie wpływa na jakość.

Warto jednak zauważyć, że dla innej grupy oponentów wynik mógłby się różnić, a wartości są tylko poglądowe. Powtarzalność pokemonów w drużynach, może pokazywać, jakie pokemony warto rozważać. Przykładami mogą być:

- Tyranitar (6/24)
- Blissey (10/24)
- Blaziken (3/24)

Mimo ponad 300 pokemonów do wyboru, te pojawiały się często, co może świadczyć, że mają wysokie statystyki oraz dobre typy (które są efektywne przeciwko większej ilości pokemonów)

Porównanie algorytmu symulowanego wyżarzania do algorytmu wspinaczkowego

W ramach porównań dwóch algorytmów, przeprowadziliśmy prosty eksperyment, by zobaczyć, która metoda może dawać lepsze rezultaty. W obu sytuacjach drużyny były oceniane na tych samych zestawach przeciwników oraz w obu przypadkach, zaczynaliśmy od tej samej startowej drużyny.

Sytuacja 1 (temperatura=1, 200 przeciwników oraz 250 iteracji):

```
Comparing SA solver to Hill Climbing solver...
Run 1/8
SA Best fitness: 0.25804777594728173
HC Best fitness: 0.410251497005988
Run 2/8
SA Best fitness: 0.2856919642857143
HC Best fitness: 0.4979959239130434
Run 3/8
SA Best fitness: 0.35015006821282396
HC Best fitness: 0.49086561743341406
Run 4/8
SA Best fitness: 0.1877534791252485
HC Best fitness: 0.436855421686747
Run 5/8
SA Best fitness: 0.1902054794520548
HC Best fitness: 0.3701649175412294
Run 6/8
SA Best fitness: 0.28984349258649095
HC Best fitness: 0.39900271370420626
Run 7/8
SA Best fitness: 0.2313403880070547
HC Best fitness: 0.442192118226601
Run 8/8
SA Best fitness: 0.2405046583850932
HC Best fitness: 0.45615789473684215
Comparison summary:
SA Mean fitness: 0.254192 ± 0.050872
HC Mean fitness: 0.437936 ± 0.041284
SA wins: 0, HC wins: 8, Draws: 0
```

Sytuacja 2 (temperatura=0.75, 50 przeciwników oraz 150 iteracji):

```
Comparing SA solver to Hill Climbing solver...
Run 1/8
SA Best fitness: 0.08645833333333335
HC Best fitness: 0.33071641791044776
Run 2/8
SA Best fitness: 0.2519677419354839
HC Best fitness: 0.30737931034482757
Run 3/8
SA Best fitness: 0.19676521739130437
HC Best fitness: 0.379127988748242
Run 4/8
SA Best fitness: 0.13607003891050581
HC Best fitness: 0.3304918032786885
Run 5/8
SA Best fitness: 0.2334622144112478
HC Best fitness: 0.4657622377622378
Run 6/8
SA Best fitness: 0.08804166666666667
HC Best fitness: 0.37570186335403727
Run 7/8
SA Best fitness: 0.13175757575757577
HC Best fitness: 0.31877216916780354
Run 8/8
SA Best fitness: 0.16844537815126048
HC Best fitness: 0.3166816143497757
Comparison summary:
SA Mean fitness: 0.161621 ± 0.058324
HC Mean fitness: 0.353079 ± 0.049356
SA wins: 0, HC wins: 8, Draws: 0
```

```
Best SA team:
Team: PokemonTeam(size=6, names=['Dewgong', 'Heatmor', 'Gogoat', 'Mothim', 'Garchomp', 'Hariyama']) Fitness: 0.2519677419354839, Stats sum: 2057
Best HC team:
Team: PokemonTeam(size=6, names=['Abomasnow', 'Rhyperior', 'Primarina', 'Blaziken', 'Salamence', 'Blissey']) Fitness: 0.4657622377622378, Stats sum: 2409
```

Dodatkowy eksperyment porównawczy (300 iteracji, 50 przeciwników):

run	solver	fitness	stats_sum	pokemons	winner
0	SA	0.2416	2178	Bellossom|Porygon-Z|Alomomola|Lucario|Donphan|Hydreigon	HC
0	HC	0.4964	1843	Blissey|Emboar|Stunfisk|Vaporeon|Drifblim|Aromatisse	HC
1	SA	0.2626	1889	Wobbuffet|Clawitzer|Mamoswine|Eelektross|Pidgeot|Kricketune	HC
1	HC	0.4663	2454	Metagross|Abomasnow|Slaking|Heracross|Mudsdale|Alomomola	HC
2	SA	0.2255	2253	Hariyama|Tyranitar|Dragalge|Comfey|Audino|Trevenant	HC
2	HC	0.5535	1936	Mudsdale|Gigalith|Abomasnow|Hariyama|Blissey|Wobbuffet	HC
3	SA	0.2175	2189	Lickilicky|Golurk|Aromatisse|Blaziken|Archeops|Hypno	HC
3	HC	0.5041	2075	Tyranitar|Blissey|Stunfisk|Alomomola|Conkeldurr|Avalugg	HC
4	SA	0.3371	2006	Sunflora|Mamoswine|Slaking|Alomomola|Chandelure|Zebstrika	HC
4	HC	0.378	2345	Gigalith|Tangrowth|Machamp|Lapras|Snorlax|Garchomp	HC
5	SA	0.2844	2151	Slaking|Masquerain|Druddigon|Hariyama|Dhelmise|Eelektross	HC
5	HC	0.5434	1904	Rhyperior|Tropius|Slaking|Pangoro|Wobbuffet|Wailord	HC
6	SA	0.407	2142	Poliwrath|Maractus|Garchomp|Hypno|Blissey|Escavalier	HC
6	HC	0.4721	2042	Blissey|Tangrowth|Tyranitar|Hariyama|Reuniclus|Lantum	HC
7	SA	0.2287	2214	Hawlucha|Beartic|Forretress|Slaking|Omastar|Hydreigon	HC
7	HC	0.4451	2240	Wobbuffet|Lapras|Garchomp|Tyranitar|Snorlax|Yanmega	HC
8	SA	0.3516	1875	Cryogonal|Hariyama|Grumpig|Stunfisk|Blissey|Ferrothorn	HC
8	HC	0.5219	1767	Stunfisk|Gogoat|Blissey|Gigalith|Hariyama|Wobbuffet	HC
9	SA	0.331	2041	Phione|Wobbuffet|Snorlax|Rhyperior|Tsareena|Lucario	HC
9	HC	0.4707	2045	Blaziken|Blissey|Gogoat|Stunfisk|Lapras|Reuniclus	HC

solver	mean	median	std	min	max	count
HC	0.4852	0.4842	0.0513	0.378	0.5535	10
SA	0.2887	0.2735	0.0647	0.2175	0.407	10

Kolejny przypadek pokazujący, że zachłanne wybieranie coraz to silniejszych drużyn daje lepsze efekty niż symulowane wyżarzanie. W tych iteracjach, algorytm

wspinaczkowy osiągnął wiele wartości powyżej progu 0.5. Trzeba jednak wziąć pod uwagę, że zostało to dokonane na grupie 50 przeciwników, która jest 4 krotnie mniejsza niż grupa, na której algorytm wspinaczkowy osiągnął wartość powyżej 0.5.

Porównanie najlepszych drużyn

Symulowane wyżarzanie:

Drużyna: ‘Gogoat', 'Rhyperior', 'Ampharos', 'Blissey', 'Drifblim', 'Wailord'

Ocena: 0.5229

Suma statystyk: 1963

Algorytm wspinaczkowy:

Drużuna: 'Mudsdale', 'Gigalith', 'Abomasnow', 'Hariyama', 'Blissey', 'Wobbuffet'

Suma statystyk: 1936

Wartość funkcji ewaluacji: 0.5535

Zwycięzca: Algorytm symulowanego wyżarzania

W obecnych testach, lepiej wypadł wspinaczkowy, dając średnio lepsze wyniki. Zmienianie drużyny na gorszej w algorytmie symulowanego wyżarzania, może czasami powodować znaczne pogorszenie drużyny i spowodować, że algorytm będzie krążył w okolicy gorszych drużyn. Zwłaszcza przy mniejszej ilości iteracji, algorytm ma większą szansę, na “utknięcie”.

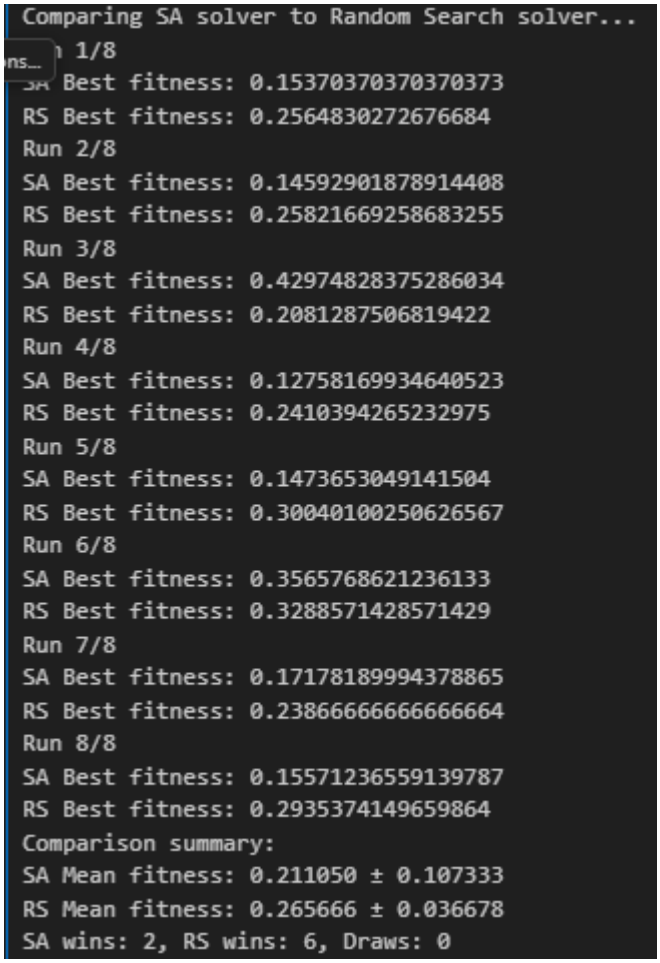
Aczkolwiek porównując najlepsze zespoły ze sobą, można zauważyć, że drużyna stworzona za pomocą algorytmu symulowanego wyżarzania, bardzo dobrze radzi sobie z drużyną stworzoną przez algorytm wspinaczkowy.

Dlatego jednoznacznie w tym wypadku nie możemy powiedzieć, który algorytm wypadł lepiej. Co nie zmienia faktu, że oba algorytmu mogą dawać satysfakcjonujące wyniki i znajdować optymalne drużyny do walki. A wszystko zależy od liczby drużyn przeciwnych, na których mierzymy skuteczność.

Porównanie algorytmu symulowanego wyżarzania do przeszukiwania losowego

W kolejnym eksperymencie porównałem algorytm symulowanego wyżarzania do przeszukiwania losowego, w którym zamiast przeszukiwać sąsiedztwa, za każdym razem losowaliśmy drużynę na nowo.

Dotychczasowy eksperyment porównał algorytm SA z wysoką temperaturą, z przeszukiwaniem losowym na budżecie 50 iteracji i ocenianym na 30 drużynach przeciwników.



Mimo zbliżonych średnich, to przeszukiwanie losowe, częściej dawało lepszą drużynę, aczkolwiek najlepszy wynik uzyskaliśmy stosując symulowane wyżarzanie.

Dobre wyniki przeszukiwania losowego, mogły też być po prostu szczęśliwym trafem, dlatego lepiej korzystać z przeszukiwania losowego.

Przeszukiwanie z uwzględnieniem pokemonów legendarnych

run	solver	fitness	stats_sum	pokemons
0	SA	0.3791	2487	Lunala|Blastoise|Groudon|Braviary|Nihilego|Throh
1	SA	0.4215	2278	Blissey|Gyarados|Zekrom|Nidoking|Throh|Xerneas
2	SA	0.3733	2554	Tyranitar|Mewtwo|Zekrom|Whiscash|Tangrowth|Exploud
3	SA	0.4248	2288	Wailord|Tangrowth|Aurorus|Mewtwo|Guzzlord|Xerneas
4	SA	0.4297	2217	Blissey|Necrozma|Blastoise|Kyurem|Stunfisk|Hariyama
5	SA	0.4522	2563	Buzzwole|Nihilego|Mewtwo|Glalie|Zygarde|Arceus
6	SA	0.405	1836	Crabominable|Gogoat|Wobbuffet|Snorlax|Nihilego|Drifblim
7	SA	0.4124	2342	Blissey|Aromatisse|Rayquaza|Rhyperior|Entei|Ampharos
8	SA	0.5038	2543	Buzzwole|Goodra|Groudon|Mewtwo|Tangrowth|Blissey
9	SA	0.4641	2224	Buzzwole|Mamoswine|Solgaleo|Slaking|Guzzlord|Relicanth

solver	mean	median	std	min	max	count
SA	0.4266	0.4232	0.0392	0.3733	0.5038	10

Przeprowadziliśmy 10 testów (dla tych samych przeciwnków), żeby znaleźć optymalne drużyny uwzględniając pokemony legendarne w naszym zbiorze. Jako iż pokemony legendarne mają najlepsze statystyki, można było się spodziewać, że będą pojawiać się bardzo licznie w najlepszych drużynach. Średnio w zespole pojawiały się 2 legendarne pokemony. Średnia funkcji oceny wynosiła około **0.42**. Może się ona wydawać niższa, gdyż legendarne pokemony, również pojawiały się w drużynach, z którymi walczyliśmy przez co były one cięższe do pokonania. Sam wynik jest zadowalający i można śmiało stwierdzić, że obecność legendarnych pokemonów znacząco może wpłynąć na siłę drużyny. Nie może być ich jednak zbyt dużo, gdyż zazwyczaj legendarne pokemony mają bardziej pospolite i powtarzalne typy, przez co łatwo im trafić na przeciwnika efektywnego względem ich.

Porównanie algorytmów: ewolucyjny i symulowane wyżarzanie

Oba algorytmy służą przeszukiwaniu przestrzeni i każdy z nich jest w stanie znaleźć nam dosyć mocną drużynę. Jednak na podstawie przeprowadzonych elementów, dotychczasowo lepsze wyniki przyniósł nam algorytm wspinaczkowy.

Algorytm ewolucyjny:

Najlepsza drużyna - 'Snorlax', 'Salamence', 'Gigalith', 'Wobbuffet', 'Lapras', 'Galvantula'

Ocena - 0.3812

Statystyki – 2061

Algorytm symulowanego wyżarzania:

Drużyna: ‘Gogoat', 'Rhyperior', 'Ampharos', 'Blissey', 'Drifblim', 'Wailord'

Ocena: 0.5229

Suma statystyk: 1963

Oczywiście algorytmy były odpalane na różnym zestawie oponentów, dlatego też jednoznacznie określić który, z nich jest lepszy.

Symulacja walki między dwoma drużynami, przyniosła zwycięstwo drużynie znalezionej przez algorytm symulowanego wyżarzania.

```
winner = compare_teams(sa_team, eq_team)
✓ 0.1s

Team1: PokemonTeam(size=6, names=['Ampharos', 'Blissey', 'Wailord', 'Drifblim', 'Rhyperior', 'Gogoat'])
Team2: PokemonTeam(size=6, names=['Lapras', 'Snorlax', 'Wobbuffet', 'Salamence', 'Gigalith', 'Galvantula'])
Team1 remaining HP: 403  /  903
Winner is Team1
```

Wpływ hiper parametrów na symulowane wyżarzanie - liczba zmienianych sąsiadów

Porównywaliśmy liczbę zmienianych pokemonów w drużynie, przy przechodzeniu do kolejnej sprawdzanej drużyny. Można to porównać do sprawdzania sąsiedztwa stopnia n tego. Sprawdzaliśmy to dla następujących parametrów:

Liczba przeciwników - 200

Ilość iteracji na eksperyment – 300

Temperatura początkowa - 0.2 -> ustawiliśmy niską temperaturę, gdyż w poprzednich eksperymentach wyszło, że wypada ona lepiej niż temperatury wysokie

Chłodzenie - 0.8

Iteracje na temperaturę - 5

Liczba zmienianych pokemonów (stopień sąsiedztwa): 1, 2, 3, 4

Domyślne = 1

run	solver	fitness	stats_sum	pokemons
0	SA	0.466	2032	Musharna|Garchomp|Blissey|Aurorus|Throh|Mandibuzz
1	SA	0.4374	1951	Mudsdale|Blissey|Tangrowth|Reuniclus|Emboar|Drifblim
2	SA	0.4041	2183	Wobbuffet|Rhyperior|Charizard|Venusaur|Hariyama|Slaking
3	SA	0.3076	2176	Houndoom|Lickilicky|Chesnaught|Aurorus|Stunfisk|Samurott
4	SA	0.3551	2122	Musharna|Golurk|Alomomola|Tyranitar|Hariyama|Audino
5	SA	0.2518	2070	Gliscor|Hypno|Rampardos|Garbodor|Snorlax|Seismitoad
6	SA	0.4824	2000	Abomasnow|Blaziken|Blissey|Drifblim|Lantum|Excadrill
7	SA	0.3733	1928	Archeops|Meganium|Blissey|Slowking|Zebstrika|Hariyama

solver	mean	median	std	min	max	count
SA	0.3847	0.3887	0.0792	0.2518	0.4824	8

N = 2

run	solver	fitness	stats_sum	pokemons
0	SA	0.3297	2345	Blaziken|Golem|Metagross|Tangrowth|Beartic|Wallord
1	SA	0.3587	2283	Slaking|Salamence|Slurpuff|Vaporeon|Musharna|Rhyperior
2	SA	0.313	2290	Mandibuzz|Kingdra|Rhyperior|Sceptile|Heracross|Snorlax
3	SA	0.4423	1891	Blissey|Rhyperior|Wobbuffet|Lapras|Florges|Blisharp
4	SA	0.3181	1921	Excadrill|Garbodor|Mandibuzz|Wobbuffet|Sylveon|Slaking
5	SA	0.3179	1849	Crabominable|Wobbuffet|Heatmor|Snorlax|Granbull|Relicanth
6	SA	0.2853	2231	Beheeyem|Bewear|Banette|Aurorus|Swampert|Mandibuzz
7	SA	0.3189	1920	Gastrodon|Meganium|Blissey|Granbull|Incineroar|Sudowoodo

solver	mean	median	std	min	max	count
SA	0.3355	0.3185	0.0476	0.2853	0.4423	8

N = 3

run	solver	fitness	stats_sum	pokemons
0	SA	0.345	2198	Pangoro|Blastoise|Meowstic|Blissey|Meganium|Garchomp
1	SA	0.3915	1749	Throh|Clefable|Magmortar|Blissey|Wobbuffet|Noivern
2	SA	0.434	1774	Blissey|Kommo-o|Amoonguss|Whiscash|Masquerain|Wobbuffet
3	SA	0.3618	1919	Krookodile|Magmortar|Blissey|Relicanth|Leavanny|Swalot
4	SA	0.3211	1892	Golduck|Mothim|Rhyperior|Simisage|Kangaskhan|Wobbuffet
5	SA	0.4056	2008	Rhyperior|Blissey|Abomasnow|Crawdaunt|Aromatisse|Vivillon
6	SA	0.4153	2258	Metagross|Blastoise|Blissey|Mudsdale|Gogoat|Aurorus
7	SA	0.2353	1979	Golurk|Gorebyss|Ampharos|Aurorus|Farfetch'd|Wobbuffet

solver	mean	median	std	min	max	count
SA	0.3637	0.3767	0.0641	0.2353	0.434	8

N = 4

run	solver	fitness	stats_sum	pokemons
0	SA	0.3355	1748	Clefable|Electrode|Slaking|Accelgor|Wobbuffet|Drifblim
1	SA	0.3702	1846	Snorlax|Golurk|Sylveon|Alomomola|Volcarona|Wobbuffet
2	SA	0.3173	1924	Sawk|Drapion|Blissey|Vaporeon|Heatmor|Exeggutor
3	SA	0.3765	2036	Skarmory|Hariyama|Vaporeon|Cradily|Garchomp|Blissey
4	SA	0.3672	2200	Kommo-o|Tangrowth|Flareon|Sharpedo|Mudsdale|Blissey
5	SA	0.3411	1918	Goodra|Conkeldurr|Serperior|Butterfree|Blissey|Whiscash
6	SA	0.4172	2023	Dhelmise|Blissey|Gyarados|Wobbuffet|Machamp|Rhyperior
7	SA	0.2491	2021	Slaking|Trevenant|Poliwrath|Zebstrika|Hippowdon|Vivillon

solver	mean	median	std	min	max	count
SA	0.3468	0.3542	0.0498	0.2491	0.4172	8

Wszystkie wyniki były w okolicach 0.36 średnich wyników. Może to oznaczać, że zmienianie dużej liczby pokemonów w naszej drużynie niekoniecznie musi wpływać pozytywnie. W niektórych sytuacjach może wymienić dwa słabe pokemony na silniejsze i tym samym wzmocnić nasz zespół, ale może tak samo usunąć dwa najlepsze pokemony i mocno go osłabić. Jest to parametr, który należy dobierać ostrożnie i najlepsze mogłoby być zastosowanie hybrydowe, czyli np. zmieniać 2-3 pokemony z drużyny, jeśli przez dłuższy okres nie nastąpiła żadna poprawa. Aczkolwiek takiej opcji nie udało nam się przetestować.

Grid Search – symulowane wyżarzanie

W ramach eksperymentów, przeprowadziliśmy również przeszukiwanie hiper siatki parametrów w celu znalezienia jakiegoś dogodnego zestawu. W tym celu przeprowadziliśmy 24 eksperymenty, poruszając się po zestawie:

```
GRID_SEARCH_PARAMS = {
    'neighbor_replacements': [1, 2, 3],
    'initial_temperature': [0.1, 0.3],
    'cooling': [0.8, 0.9],
    'legendaries': [False, True],
}
```

Jest to również część porównująca wpływ liczby zmienianych pokemonów, aczkolwiek dodatkowo uwzględniamy różne wartości temperatury i chłodzenia oraz uwzględniamy

pokemony legendarne. Wyników do wklejania jest tutaj za dużo, aczkolwiek wszystkie wyniki dostępne są na naszym repozytorium w zakładce **src/data/reports/sa_experiment_{index}_...**

Poniżej umieszczę kilka najlepszych drużyn, które udało się znaleźć:

- Ocena: 0.4320700896495518
  Drużyna: ['Musharna', 'Primarina', 'Blissey', 'Amoonguss', 'Hariyama', 'Stunfisk']

- Ocena: 0.4873111782477341
  Drużyna: ['Cresselia', 'Guzzlord', 'Blissey', 'Vaporeon', 'Entei', 'Drifblim']

- Ocena: 0.48408262454434997
  Drużyna: ['Abomasnow', 'Slowbro', 'Gengar', 'Guzzlord', 'Blissey', 'Groudon']

- Ocena: 0.46498740554156176
  Drużyna: ['Musharna', 'Rhyperior', 'Tropius', 'Blissey', 'Hariyama', 'Illumise']
- Ocena: 0.4964166666666667
  Drużyna: ['Groudon', 'Zekrom', 'Blissey', 'Typhlosion', 'Rampardos', 'Wailord']

- Ocena: 0.4646706586826347
  Drużyna: ['Buzzwole', 'Kyogre', 'Blissey', 'Eelektross', 'Guzzlord', 'Weezing']

- Ocena 0.48423955870764374
  Drużyna: ['Kyogre', 'Slaking', 'Tropius', 'Hariyama', 'Zygarde', 'Solgaleo']

Większość lepszych drużyn znajdowano przy podmianie 1 bądź 2 pokemonów, oraz przy uwzględnieniu pokemonów legendarnych. Wyniki potwierdzają dwie tezy:

- Uwzględnianie pokemonów legendarnych polepsza wyniki drużyn
- Lepiej zmieniać mniej pokemonów w pojedynczej iteracji, by uniknąć powiększenia szansy na usunięcie tych lepszych z zespołu

Turniej najlepszych znalezionych drużyn z poszczególnych eksperymentów

Z najlepszych drużyn, które udało nam się zebrać na przestrzeni eksperymentów musimy wyłonić tą najlepszą. Zebraliśmy 16 drużyn, które możemy zobaczyć poniżej na załączonym obrazku i sprawdziliśmy ich skuteczność w dwóch kategoriach:

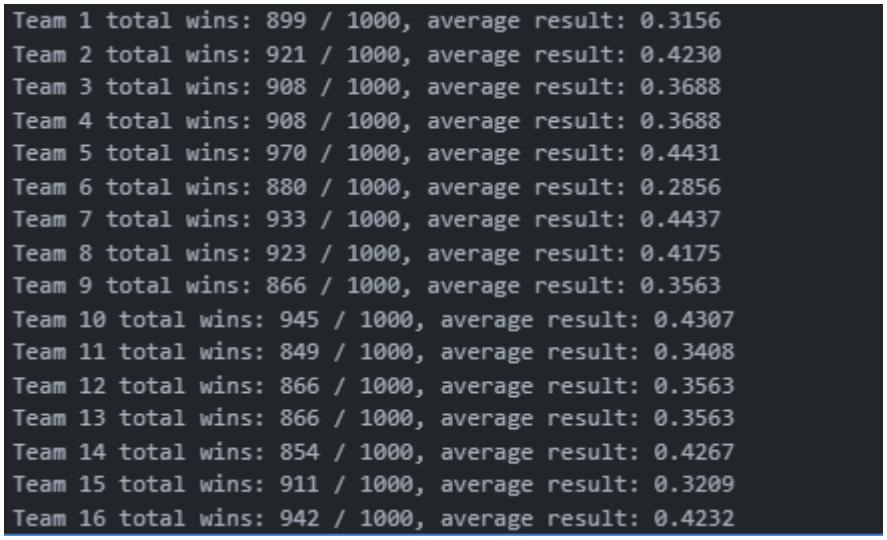
1. Najwięcej pokonanych drużyn: Drużyna nr 14 -> ['Cresselia', 'Guzzlord', 'Blissey', 'Vaporeon', 'Entei', 'Drifblim'] - pokonała 9 / 15 drużyn
2. Najlepsza funkcja oceny: Ta sama drużyna co wyżej z wynikiem 0.2890

```
TEAMS = {
    1: ['Rhyperior', 'Tropius', 'Slaking', 'Pangoro', 'Wobbuffet', 'Wailord'],
    2: ['Tyranitar', 'Blissey', 'Stunfisk', 'Alomomola', 'Conkeldurr', 'Avalugg'],
    3: ['Blissey', 'Emboar', 'Stunfisk', 'Vaporeon', 'Drifblim', 'Aromatisse'],
    4: ['Blissey', 'Emboar', 'Stunfisk', 'Vaporeon', 'Drifblim', 'Aromatisse'],
    5: ['Buzzwole', 'Goodra', 'Groudon', 'Mewtwo', 'Tangrowth', 'Blissey'],
    6: ['Snorlax', 'Salamence', 'Gigalith', 'Wobbuffet', 'Lapras', 'Galvantula'],
    7: ['Garchomp', 'Alomomola', 'Hawlucha', 'Tyranitar', 'Blissey', 'Gogoat'],
    8: ['Rhyperior', 'Tangrowth', 'Passimian', 'Blissey', 'Togekiss', 'Alomomola'],
    9: ['Gogoat', 'Rhyperior', 'Ampharos', 'Blissey', 'Drifblim', 'Wailord' ],
    10: ['Blaziken', 'Tyranitar', 'Goodra', 'Mamoswine', 'Alomomola', 'Blissey' ],
    11: ['Mudsdale', 'Gigalith', 'Abomasnow', 'Hariyama', 'Blissey', 'Wobbuffet'],
    12: ['Gogoat', 'Rhyperior', 'Ampharos', 'Blissey', 'Drifblim', 'Wailord'],
    13: ['Gogoat', 'Rhyperior', 'Ampharos', 'Blissey', 'Drifblim', 'Wailord'],
    14: ['Cresselia', 'Guzzlord', 'Blissey', 'Vaporeon', 'Entei', 'Drifblim'],
    15: ['Groudon', 'Zekrom', 'Blissey', 'Typhlosion', 'Rampardos', 'Wailord'],
    16: ['Kyogre', 'Slaking', 'Tropius', 'Hariyama', 'Zygarde', 'Solgaleo']
}
```

```
team_wins
✓ 0.0s
{1: 4,
 2: 1,
 3: 5,
 4: 5,
 5: 4,
 6: 0,
 7: 2,
 8: 2,
 9: 7,
 10: 6,
 11: 2,
 12: 7,
 13: 7,
 14: 9,
 15: 2,
 16: 5}
```

Porównanie najlepszych drużyn na nowych zestawach przeciwników

Postanowiliśmy zebrać najlepsze drużyny, które udało nam się zdobyć podczas eksperymentów oraz przeprowadzić dodatkową analizę funkcji oceny, poddając te drużyny symulacji z 1000 różnych drużyn. Zrobiliśmy to, żeby sprawdzić jak dobrze będą się one adaptowały do większych, a tym samym bardziej różnorodnych drużyn.



Mimo że drużyna nr 14 okazała się najlepsza w walce z pozostałymi 15 drużynami, to porównując je na większym zestawie oponentów, możemy dostrzec, że nie wypada ona już najlepiej. Bardziej optymalną drużyną jest drużyna 5, która wygrała 970 z 1000 pojedynków. Co ciekawe, mimo największej liczby zwycięstw, nie osiągnęła ona najlepszego wyniku funkcji oceny. Przypada on bowiem drużynie numer 7, która wygrała 37 mniej walk, aczkolwiek zachowywała średnio 0.0006 % punktów życia więcej. Jak widać nasza funkcja oceny, mimo że nie przebija granicy 0.5 to i tak drużyny wygrywają ponad 80/90 % wszystkich swoich walk. Oznacza to, że nawet współczynniki 0.52 / 0.55 uzyskiwane w poprzednich eksperymentach mogą sugerować bardzo mocne i optymalne drużyny.

Podsumowanie

Ze względu na specyfikację problemu, nie można stwierdzić, który z algorytmów jest lepszy, bo jest to zależne w dużym stopniu od tego z kim musimy walczyć, gdyż nie jesteśmy w stanie znaleźć “najlepszej drużyny”. Nasz sposób symulacji jest dosyć uproszczoną wersją, gdyż w oryginalnych grach o tym kto wygra nie świadczą tylko statystyki ani typy, ale bardziej ruchy pokemonów, ich konkretne zestawienia w danej sytuacji oraz taktyka.