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 sześcio-elementowe 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 finalną 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 wodnych (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 cztero elementowy 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 i wyłącznie 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 < 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.

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 celu. 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.

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.

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,

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.

Przeprowadzone eksperymenty i badania

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

Podsumowanie