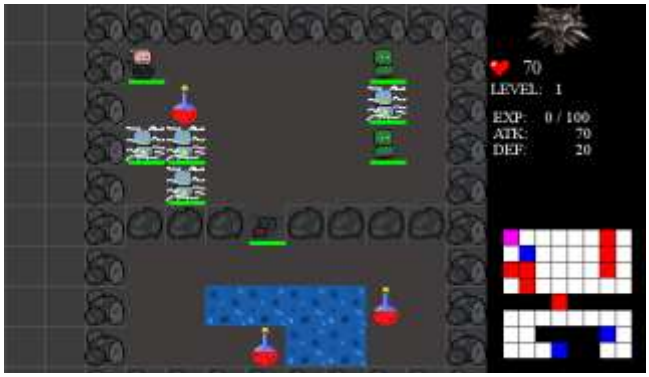
Metody uczenia: **algorytmy genetyczne – Mikołaj Balcerek 416040**



Gra turowa polegająca na znalezieniu takiego przejścia po mapie, by zabić wszystkie potwory

Dla gry „Sztuczny Wiedźmin" - https://github.com/MikolajSzumigalski/SZILATO2018

Algorytmy genetyczne (GA) – zainspirowana procesem ewolucji heurystyka przeszukująca przestrzeń rozwiązań.
Znajdują one główne zastosowanie w projektowaniu małych części (reflektory, aerodynamiczne wykończenia, proste silniki - http://eprints.gla.ac.uk/3818/) i problemów rodzaju układania planów czasowych (lista zastosowań https://en.wikipedia.org/wiki/List_of_genetic_algorithm_applications).

Próba rozwiązania gry tym sposobem wydaje się trafna, ze względu na poprzednie takie eksperymenty (http://lbrandy.com/blog/2010/11/using-genetic-algorithms-to-find-starcraft-2-build-orders/).

Ich głównymi wadami jest **skłonność do znajdywania lokalnie dobrych rozwiązań (a nie globalnych)**, konieczność **często kosztownego wielokrotnego przeliczania *evaluate fitness function*** oraz słabe skalowanie dla problemów, gdzie osobnik składałby się z bardzo wielu genów.

Dodatkowo konieczne jest, by możliwe było stopniowe ocenianie poprawienia/pogorszenia jakości rozwiązania (to rozwiązanie jest o 10% lepsze niż poprzednie, a nie: poprawne/niepoprawne).

Do zastosowania algorytmów genetycznych potrzeba:

Graficzna reprezentacja wewnętrznego stanu osobnika

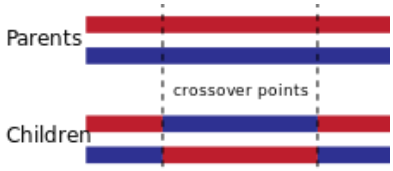
- osobnika, składającego się z genów. Jest to genetyczna reprezentacja rozwiązania problemu.
Dla problemu *„Sztucznego Wiedźmina"* osobnikiem jest lista o stałej długości (wielokrotność liczby wszystkich obiektów na mapie), zawierająca kolejne indeksy obiektów na mapie do odwiedzenia (jeden indeks to jeden gen).
- populacja, zbiór osobników (300)
- *fitness function,* funkcja oceniająca rozwiązanie. Ważne, by mogła ocenić stopień poprawności rozwiązania na pewnej skali. W mojej implementacji jest to funkcja przypisująca osobnikowi trzy wyniki, biorąc również też pod uwagę ich cele (ilość pozostałych potworów (minimalizacja), życie bohatera (maksymalizacja) oraz suma punktów życia potworów (minimalizacja). W trakcie porównywania osobników pod uwagę brane są jego argumenty w kolejności leksykograficznej. Znaczy to, że dwie późniejsze cechy służą tylko do oceniania takich posunięć, które zabiją tą samą ilość potworów.

Wykorzystany algorytm: **Simple Evolutionary Algorithm** (Back, Fogel and Michalewicz, “Evolutionary Computation 1 : Basic Algorithms and Operators”, 2000)

Kroki:

1. Stwórz losową populację o ustalonym rozmiarze (losowe liczby z zakresu <0, liczba obiektów na mapie>)
2. Dla każdego osobnika z populacji wylicz jego *fitness*
3. Dla ustalonej liczby generacji, lub do warunku stopu (np. brak poprawy *fitness* w populacji):

    a. Selekcja. Z populacji wybierz osobniki, które wezmą udział w *crossover*.
    W projekcie została zastosowana **metoda turniejowa**.
    Polega ona na przeprowadzeniu turnieju (na podstawie *fitness*) między losowo wybranymi 15-osobnikami, gdzie zwycięzca dodawany jest do populacji następnego pokolenia. Losowe wybranie chromosomów do konkursu ma na celu zwiększenie puli genetycznej, która będzie brała udział w mieszaniu.
    b. Crossover. Wśród wszystkich możliwych par wybranych uprzednio osobników losuje się część (1/2), dla której zajdzie proces reprodukcji. Tak powstałe i rodzice są częścią nowej generacji.
    Została wykorzystana metoda **Two Point Crossover.**



Wikimedia

Metoda polega na zaznaczeniu losowego przedziału na listach posunięć rodziców (o stałej długości). Chromosomy dzieci będą składały się z kopii rodziców, z pozamienianym wybranym fragmentem.

    c. Mutacja. Dla losowych osobników (0.2) nowej generacji istnieje szansa, że jego kod genetyczny ulegnie zmianie.
    Zastosowany sposób polega na małym prawdopodobieństwie (0.05), że gen zostanie zastąpiony nowym z puli wszystkich możliwych.
    d. Rekalkulacja fitness. Dla zmodyfikowanych chromosomów oblicza się jeszcze raz wartość *fitness.*

Wyniki

Stan gry po przejściu najlepszego chromosomu

W trakcie działaniu programu przechowuję informacje o wyniku średnim, maksymalnym, minimalnym i odchyleniu standardowym dla *fitness* populacji. Dodatkowo przechowywanych jest kilku najlepszych osobników wśród wszystkich generacji.

Z danych dla 31 generacji widać, że średnia fitness dla argumentu (liczba potworów * 1000) znacząco się obniżyła. W pierwszej generacji prawie wszystkie osobniki giną. Po symulacji większość osobników skutecznie zabija co najmniej jednego potwora.

Najlepsze osobniki osiągają stan [(2.0 potwory, 40.0 życia , 170.0 życia potworów)].
Po 31 pokoleniach żadnemu osobnikowi nie udało się rozwiązać gry (mimo kilku rozwiązań).
Analiza powtórek pokazuje, że należy zmienić tylko kolejność dwóch ruchów, by osiągnąć zwycięstwo.

Gen	Evals	Średnia fitness populacji* wagi	Odchylenie standardowe fitness populacji * wagi
0	300	[9.83333700e+04 6.66666667e-01 9.83381000e+04]	[1.28016279e+04 5.12076383e+00 1.27652964e+04]
1	174	[8.33337e+04 7.60000e+00 8.33811e+04]	[3.72669797e+04 1.86790435e+01 3.71609921e+04]
2	156	[20668.29666667 45.7 20890.7]	[4.04905938e+04 3.59514951e+01 4.03770919e+04]
3	195	[9335.16 87.66666667 9608.43333333]	[29089.31289668 38.49530996 29001.6463075]
4	172	[8001.84333333 94.43333333 8268.83333333]	[27128.77636322 34.63156877 27050.05335125]
5	178	[18001.64333333 85.53333333 18209.8]	[38417.97548799 43.65831981 38320.45739932]
6	182	[7668.51333333 100.63333333 7871.73333333]	[26605.64928012 30.08763127 26547.09235168]
7	186	[4335.25666667 104.06666667 4544.6]	[20360.22806464 23.89132804 20315.67467515]
8	181	[6668.54333333 101.86666667 6872.8]	[24943.88101843 28.39921752 24889.29190422]
9	197	[6668.54333333 101.03333333 6872.8]	[24943.88101843 28.99308347 24889.29222564]
10	172	[7001.86666667 100.8 7205.56666667]	[25514.18952143 29.62026333 25458.30501059]
11	167	[5335.23 103.66666667 5542.83333333]	[22469.2825427 25.25646232 22420.00767853]
12	184	[6335.21 101.16666667 6539.7]	[24355.66569307 28.61769072 24302.49241491]
13	153	[6668.54 100.56666667 6872.86666667]	[24943.88190923 29.40542278 24889.27466029]
14	163	[5668.55666667 102.36666667 5874.46666667]	[23119.99428576 26.9332302 23069.52802686]
15	180	[8001.85 98.86666667 8203.36666667]	[27128.77439745 31.73928516 27069.35154375]
16	198	[9335.15333333 98.7 9534.03333333]	[29089.31503553 32.92380091 29025.50670357]
17	178	[8001.84 100.26666667 8203.76666667]	[27128.7773461 30.63650691 27069.23342368]
18	213	[8668.5 98.83333333 8868.46666667]	[28134.03238257 32.22274904 28072.43489586]
19	171	[6001.88333333 102.93333333 6207.96666667]	[23748.20835831 26.62196253 23696.14337536]
20	193	[7668.51666667 100.53333333 7870.26666667]	[26605.64831967 30.33780187 26547.51437069]
21	170	[6001.88 101.76666667 6207.23333333]	[23748.20920039 27.79470373 23696.32904085]
22	187	[5335.23666667 102.63333333 5543.06666667]	[22469.28096062 26.57189409 22419.95320978]
23	175	[7001.86333333 101.36666667 7205.16666667]	[25514.19043587 28.94475581 25458.41496059]
24	179	[8668.5 98.83333333 8867.9]	[28134.03238257 32.22274904 28072.60949615]
25	186	[7335.19666667 100.3 7538.56666667]	[26067.76221603 30.12601312 26010.55345327]
26	187	[9001.83 98.46666667 9201.3]	[28617.60053431 32.71567752 28554.8710143]
27	156	[10335.13 97.83333333 10530.96666667]	[30438.76516582 34.18048501 30372.28427474]
28	157	[6335.21333333 100.93333333 6540.26666667]	[24355.66482637 28.82930608 24302.34555063]
29	171	[7668.51666667 99.83333333 7869.66666667]	[26605.64831967 30.91340953 26547.68707858]
30	203	[7668.53 100.86666667 7871.]	[26605.64447786 29.88726968 26547.30333449]

Implementacja i uruchomienie

Algorytmy genetyczne oparte na bibliotece **DEAP** dla języka **Python 3**.

Do projektu dołączony jest plik Pipfile, zawierający listę wymaganych bibliotek. Można go wykorzystać za pomocą narzędzia pipenv. Polecenie *>pipenv install >pipenv shell*

Uruchomienie projektu >*python run.py*.

Algorytm generyczny uruchamia się po naciśnięciu „0”.

Po zasymulowaniu ustalonej liczby generacji (zmienna ngen w GeneticAlgorithm/genetic.py) istnieje możliwość odtworzenia przejścia najlepszego osobnika po wpisaniu dowolnego znaku do okna konsoli.

Dodatkowo tworzone są pliki gen_outcome_{} zawierające informacje o całym działaniu algorytmu.

Funkcja map obliczająca wartość *fitness* działa wielordzeniowo.