23.04.2020

Projekt indywidualny - algorytm genetyczny - generator stolików.

1. Opis problemu

   Podczas tworzenia projektu “Inteligentny kelner”, pojawił się problem generowania ustawień stolików. Przy losowym generowaniu okazywało się, że dużo stolików jest blokowanych przez inne i kelner nie mógł się do nich dostać. Wprowadzenie kontroli sąsiedztwa podczas losowania również nie rozwiązywało w całości problemu, pojawiały się cykle, które i tak powodowały, że część stolików była nadal nieosiągalna. Ogólne ustawienie było mało optymalne i nie pozwalało w pełni wykorzystać dostępnej powierzchni lokalu. Do rozwiązania problemu postanowiłem zastosować algorytm genetyczny.

2. Ogólny opis algorytmu

   Algorytm oparty jest na doborze naturalnym, znanym z przyrody i mechanizmie ewolucji. Populacja składa się z osobników, z których każdy zawiera zestaw genów. Geny nowych organizmów są kombinacją genów pochodzących od rodziców. Siłę danego osobnika określa funkcja dostosowania (fitness). Osobnik silniejszy ma większą szansę przekazania swoich genów następnej generacji. Dodatkową zmienność zapewnia mechanizm mutacji, który w bardzo małym stopniu modyfikuje zestaw genów, w taki sposób, że zmutowany gen nie pochodzi bezpośrednio od żadnego z rodziców.
   Dla odpowiednio dużej populacji i dużej ilości generacji, algorytm powinien promować najlepsze osobniki w każdej z kolejnych iteracji.

3. Zastosowanie algorytmu do problemu generowania stolików

   Najbardziej elementarnym obiektem jest osobnik, tworzony przez klasę Individual. Zestawem genów każdego osobnika jest rozmieszczenie zer i jedynek w dwuwymiarowej tablicy, o wymiarach sali naszej restauracji. Zera reprezentują puste miejsca, a jedynki oznaczają miejsca ze stolikiem. Główna klasa to GATableGenerator. Wewnątrz niej dochodzi do wyboru rodziców, prokreacji, krzyżowania osobników i mutacji. Cały przebieg algorytmu od wygenerowania pierwszej generacji do ostatniej odbywa się wewnątrz wspomnianej klasy. Parametry sterujące procesem pobierane są z atrybutów klasy GADefaults. Funkcja dostosowania liczona jest jako liczba stolików - liczba stolików niedostępnych. W ten sposób osobnik z małą liczbą stolików jest słabszy niż ten z większa ich liczbą. Jednocześnie osobnik, który ma za dużo, źle rozmieszczonych stolików będzie karany za każdy zablokowany stolik. Widać tu pewien konflikt interesów, z jednej strony algorytm dąży do uzyskania jak największej liczby stolików na planszy, ale z drugiej strony gdy jest ich zbyt wiele, bardzo łatwo będzie mu zablokować dojście do wielu z nich, powodując drastyczny spadek siły osobnika. Tak liczona funkcja dostosowania, zmusza algorytm do szukania optymalnych rozwiązań naszego

problemu. Do implementacji funkcji obliczającej przystosowanie osobnika wykorzystałem już zaimplementowany algorytm A*. Dla każdego stolika sprawdzam, czy istnieje przynajmniej jedna ścieżka prowadząca do jego sąsiedztwa.

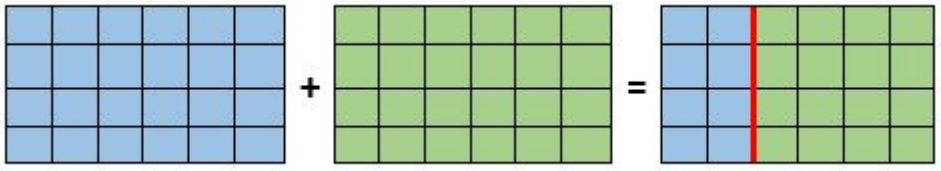
4. Metody selekcji

- metoda ruletki, wynik fitness danego osobnika stanowi procentowy udział w losowaniu go na rodzica. Proces powtarzam dwukrotnie, za drugim razem nie uwzględniając wcześniej wybranego już rodzica.

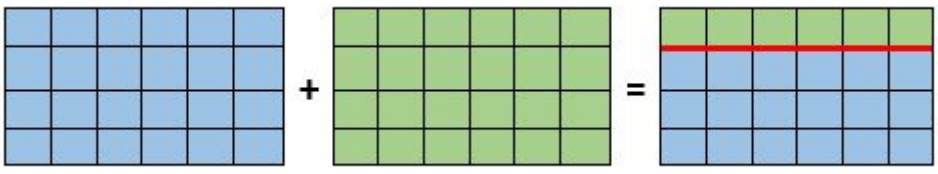
- metoda turnieju, spośród wszystkich osobników populacji losuję z równym prawdopodobieństwem 25% organizmów. Najlepszy (względem fitnessu) z tak wybranych stanie się jednym z rodziców, proces powtarzam dwukrotnie. Przy wyborze drugiego z pary rodziców, wcześniej wyselekcjonowany nie bierze już udziału.

5. Metody krzyżowania

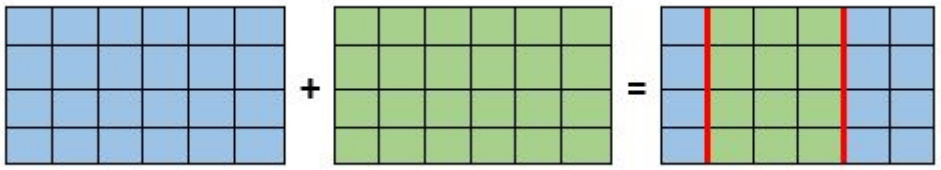
- pojedynczy pionowy, losuję pionową granicę, po współrzędnych osi X sali restauracji. Nadaję margines o szerokości jednej kratki na dwóch końcach sali, w taki sposób by wylosowana granica nigdy nie miała wartości skrajnych osi X. Wszystkie stoliki po lewej stronie granicy organizm odziedziczy po pierwszym rodzicu i analogicznie po prawej stronie, wszystkie geny dostanie od drugiego rodzica. W skrajnym przypadku organizm dostanie tylko jedną kolumnę od jednego z rodziców, pozostałą część swoich genów odziedziczy zaś po drugim.



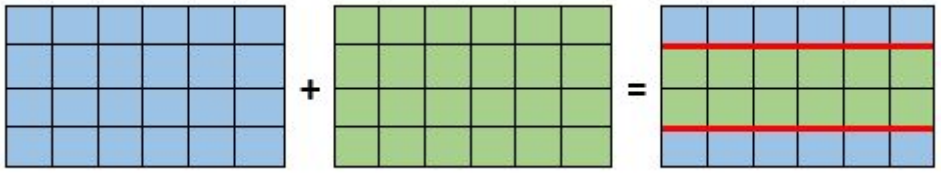
- pojedynczy poziomy, również losuję granicę. Tym razem względem osi Y. Ponownie margines wynosi jedną kratkę na początku i końca osi. Wszystkie geny powyżej granicy zostaną odziedziczone przez jednego z rodziców a poniżej przez drugiego. W skrajnym przypadku organizm odziedziczy tylko jeden z wierszy przez jednego z rodziców, pozostałe wiersze sali będą kopią drugiego.



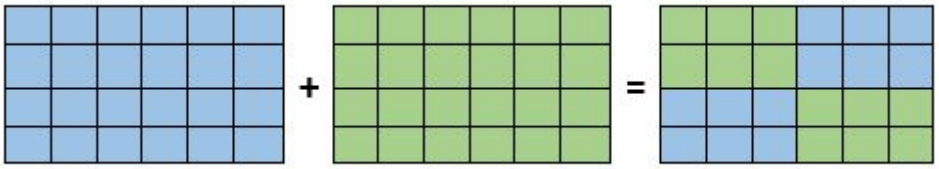
- podwójny pionowy, tym razem losuję dwie granice na osi X. Uwzględniam margines jednej kolumny dla wartości skrajnych osi. Elementy znajdujące się pomiędzy obiema granicami pochodzą od jednego z rodziców pozostałe od drugiego.



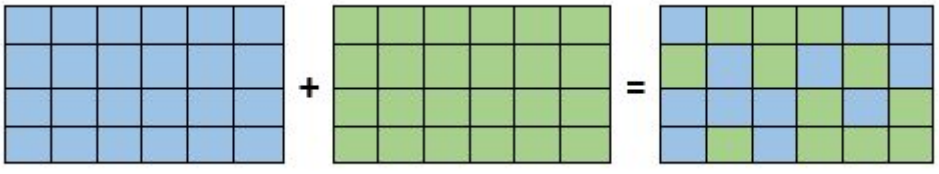
- podwójny poziomy, analogicznie jak w przypadku wyżej, losowe dwie poziome granice podzielą salę na 3 strefy. Strefa pomiędzy oboma granicami wyznacza fragment genomu od jednego z rodziców, pozostałe dwie strefy będą fragmentem pochodzącym od drugiego.



- ćwiartek, w przeciwieństwie do powyższych, w tym wypadku miejsce podziału jest na stałe ustalone. Dzielę salę na 4 części jedną granicą pionową i jedną poziomą. Miejscem przecięcia granic jest środek sali, czyli części są równe. Lewa górna i prawa dolna część przekazane będą potomkowi przez jednego z rodziców, pozostałe przez drugiego.

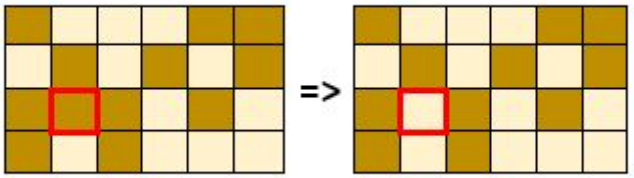


- losowa, dla każdego z pól u potomka losowany jest rodzic, który przekaże mu swoją wartość tego samego pola.

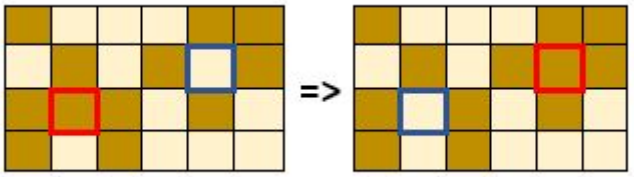


6. Metody mutacji

- inwersja, losowo wybieramy punkty na planszy organizmu potomnego. Jeśli organizm miał w tym punkcie stolik to go usuwamy. W przeciwnym razie dodajemy stolik.



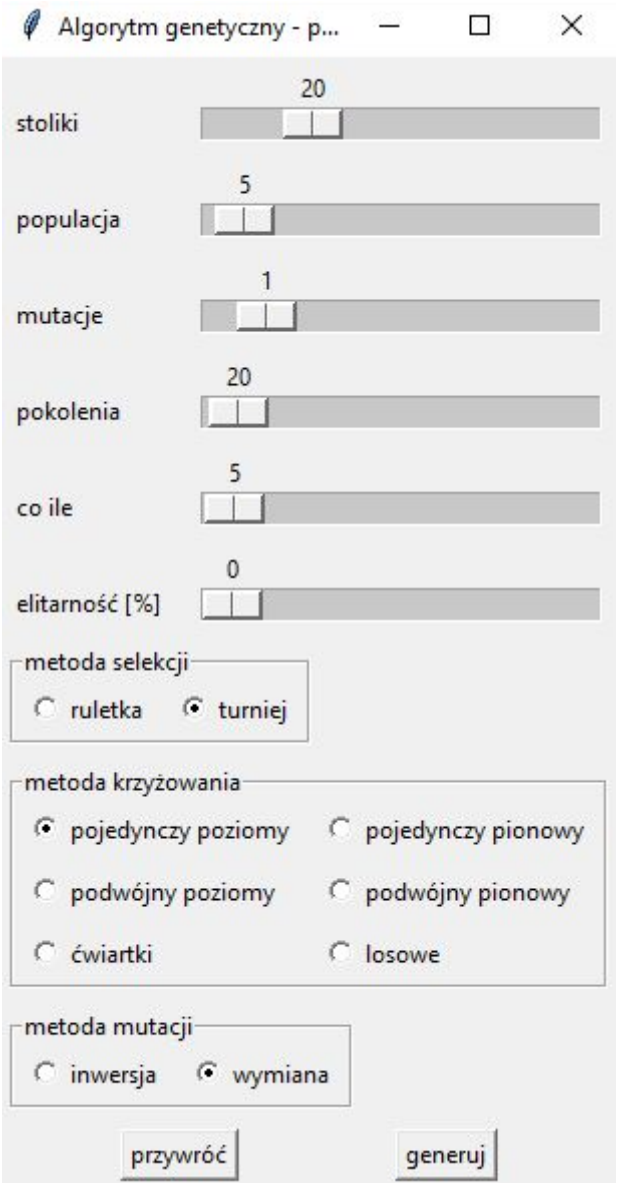
- wymiana, losujemy parę punktów i wartości poszczególnych kratek zamienimy ze sobą.



7. Parametry startowe

   Przed rozpoczęciem działania algorytmu, użytkownik ma możliwość ustawienia parametrów. Każdy z możliwych do ustawienia parametrów ma wartość domyślną. Aby algorytm zaczął działać użytkownik nie musi zmieniać żadnej wartości. Odpowiednia kombinacja ustawień i wartości parametrów, znacząco wpływa na pracę programu jak i efekt końcowy. Okno dialogowe zaimplementowane jest za pomocą klasy GADialog. Powstały obiekt komunikuje się z obiektem klasy GADefaults, genetyczny generator stolików ma dostęp do zmiennych tej samej klasy. W ten sposób zmieniając parametry w oknie dialogowym wpływamy na pracę algorytmu genetycznego. Na dole okna znajdują się dwa przyciski “przywróć” i “generuj”. Po kliknięciu pierwszego, wartości domyślne wszystkich parametrów zostaną przywrócone, drugi z nich startuje pracę z podanymi parametrami.

Okno dialogowe z wartościami domyślnymi



8. Opis parametrów

- stoliki, liczba stolików w pierwszej losowej populacji
- populacja, liczba organizmów w jednej populacji
- mutacje, intensywność mutacji np. dla wartości 3, nowy organizm ma szansę na zmutowanie zera, jednej, dwóch lub trzech swoich pól/pary pól
- pokolenia, liczba pokoleń
- co ile, określa co które pokolenie prezentować wyniki
- elitarność [%], określa ile procent najlepszych osobników z populacji przejdzie niezmieniona do następnej generacji
- metody selekcji, opisane w punkcie 4
- metody krzyżowania, opisane w punkcie 5
- metody mutacji, opisane w punkcie 6

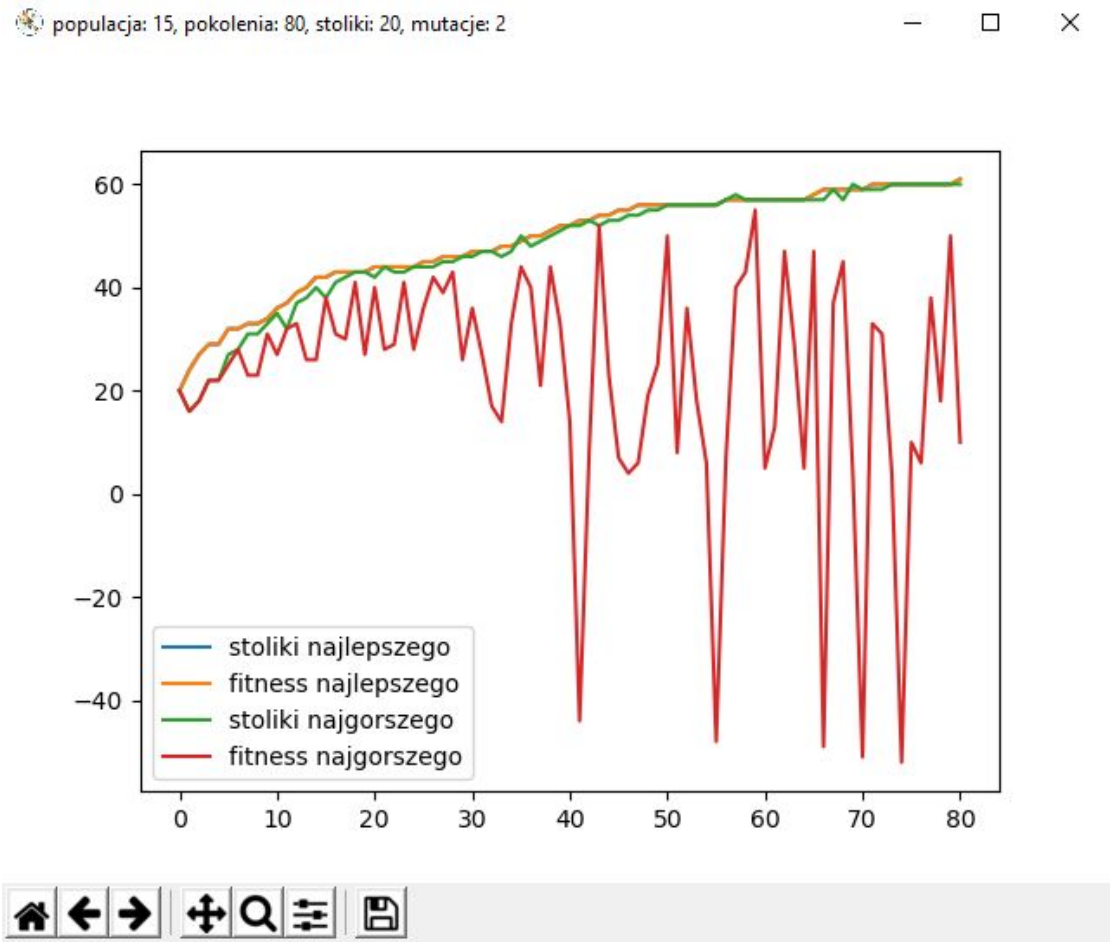
9. Przebieg algorytmu

   Po naciśnięciu przycisku generuj, algorytm genetyczny zaczyna generować kolejne pokolenia. Użytkownik widzi tylko najlepszego osobnika z każdej generacji.

Co podaną przez użytkownika liczbę pokoleń w parametrze “co ile”, prezentowany jest messagebox. W tytule okna wypisane są wyniki prezentowanego osobnika. Użytkownik chcący zakończyć przedwcześnie pracę algorytmu może kliknąć przycisk “Nie”. W takiej sytuacji wygeneruje się wykres przedstawiający statystyki dotyczące najlepszego i najgorszego osobnika z każdej generacji. Po zamknięciu okna wykresu kelner z głównej części aplikacji zaczyna swoją pracę na obecnie wygenerowanej najlepszej planszy.

10. Koniec algorytmu i prezentacja wyników

Algorytm może zakończyć się na dwa sposoby. Przedwcześnie, za pomocą wyżej wspomnianego messageboxa, albo gdy wygenerują się wszystkie pokolenia podane w parametrach początkowych. W obu przypadkach powstanie wykres prezentujący najważniejsze dane dotyczące wszystkich pokoleń. Wykresy opisane są legendą.



11. Diagram prezentujący działanie algorytmu