Wprowadzenie

Czym jest obrazek logiczny?

Obrazek logiczny jest łamigłówką o bardzo prostych zasadach
Polega on na zaznaczeniu odpowiednich pól, a po zaznaczeniu wszystkich pól otrzymamy obrazek w wyniku

Zasady rozwiązywania obrazków logicznych

Liczby u góry i z lewej strony diagramu określają, które pola należy zaznaczyć.
Każda liczba określa długość grupy zamalowanych pól w danym rzędzie lub kolumnie.
Pomiędzy grupami zamalowanych pól musi być co najmniej jedno pole puste. Kolejność liczb mówi o kolejności grup zamalowanych pól.

Rozwiązywanie obrazków logicznych z użyciem NonogramSolvera

Logika trójwartościowa:

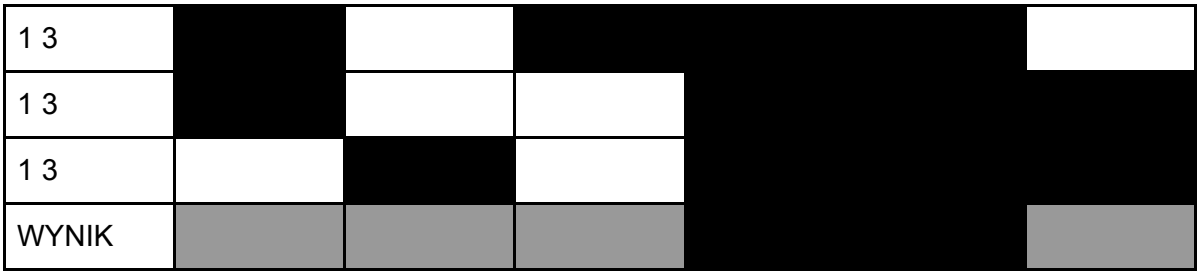
Każdy z kwadratów może przyjąć 3 stany(wartości):
PEŁNY - dany kwadrat należy do obiektu - jest wypełniony
PUSTY - dany kwadrat NIE należy do obiektu, jest niewypełniony
NIEZNANY - stan przejściowy - nie wiadomo czy kwadrat należy

A\B	PEŁNY	PUSTY	NIEZNANY
PEŁNY	PEŁNY	NIEZNANY	NIEZNANY
PUSTY	NIEZNANY	PUSTY	NIEZNANY
NIEZNANY	NIEZNANY	NIEZNANY	NIEZNANY

Tabela wynikowa AND'a logicznego dla 2 wartości

PEŁNY + PEŁNY = PEŁNY
PUSTY + PUSTY = PUSTY
NIEZNANY + _ = NIEZNANY

Przykład użycia:



Operacja AND'a dla przykładowego fragmentu obrazka logicznego o szerokości 6 i podpowiedziach 1,3 - mamy trzy różne możliwości rozmieszczenia i na podstawie już pierwszego porównania ich możemy odnaleźć 2 pewne pola z wartością PEŁNY

Wejściowe dane(podpowiedzi)

Przykładowe dane wejściowe:
[[
[4,5,3,3], [4,8,3,3], [5,4,3,3], [5,3,3,3], [7,4,3,3], [2,3,3,3,10], [2,3,3,3,10], [9,4,3,3,3], [9,3,3,3,3], [2,3,4,3,3,3], [3,4,8,3,3], [2,3,6,3,3]
],[
[2], [5], [7], [9], [5,2], [5,2], [9], [9], [8], [5], [2], [0], [0], [0], [4], [8], [10], [4,4], [3,3], [2,2], [2,2], [2,7], [2,7], [1,7], [0], [0], [0], [0], [0], [12], [12], [12], [2], [2], [2], [2], [12], [12], [12]
]]

jak widzimy jest tu jedna tablica w środku, której są dwie tablice, w pierwszej tablicy mamy podpowiedzi do wierszy, a w drugiej podpowiedzi do kolumn
Przykładowo "[4,8,3,3]" zawiera podpowiedzi dla pojedynczego wiersza

Generowanie możliwych propozycji początkowych

Czyli znalezienie wszystkich możliwych kombinacji kratek pustych i pełnych dla każdego wiersza i kolumny

Najpierw wyznaczamy wszystkie możliwe kombinacje przerw między grupami zamalowanych pól
Przykłady:
Dla wiersza [1,3] o długości 6:
-mamy do rozstawienia 1 kratkę (6-(1+3)-len([1,3])+1)
{0,0,0}, 3, 1
	{1,0,0}, 2, 0
		{1,0,0}, 1, 0 => 1,0,0
{0,0,0}, 3, 1
	{0,0,0}, 2, 1

{0,1,0}, 1, 0 => 0,1,0
{0,0,0}, 3, 1
{0,0,0}, 2, 1
{0,0,0}, 1, 1 => 0,0,1

1,0,0	PUSTE	GRUPA PKT	PUSTY (MUST)	GRUPA PKT
0,1,0	GRUPA PKT	PUSTE	PUSTY (MUST)	GRUPA PKT
0,0,1	GRUPA PKT	PUSTY (MUST)	GRUPA PKT	PUSTE

Następnie na tej podstawie generowane są wszystkie możliwe rozstawienia dla wierszy i kolumn, gdzie 1 odpowiada stanowi pełnemu, a -1 pustemu

Dla powyższego przykładu otrzymamy:

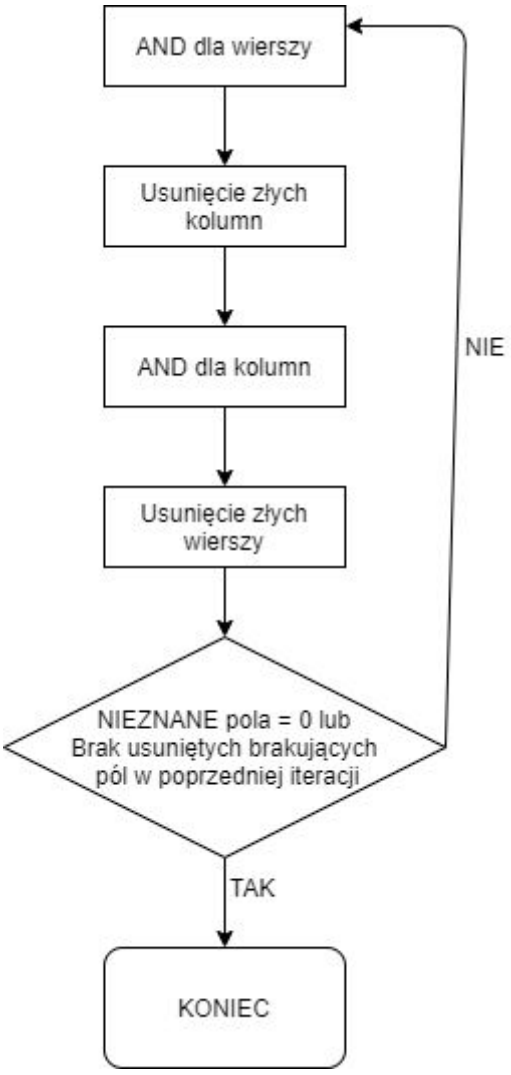
1,0,0							
0,1,0							
0,0,1							

Optymalizacja

W celu optymalizacji wybieram czy zacząć od kolumn czy od wierszy,
Tak naprawdę sprowadza się to do sum(len(w_i)) > sum(len(h_i)) ? start od kolumn : start od wierszy - czyli jeżeli jest mniej podpowiedzi w kolumnach -> mają one dłuższą długość -> łatwiej będzie uzyskać szybciej więcej stanów PUSTYCH i PEŁNYCH

Loop algorytmu

Przykładowo gdy zaczynamy od wierszy(dla kolumn różni się tylko początkiem)

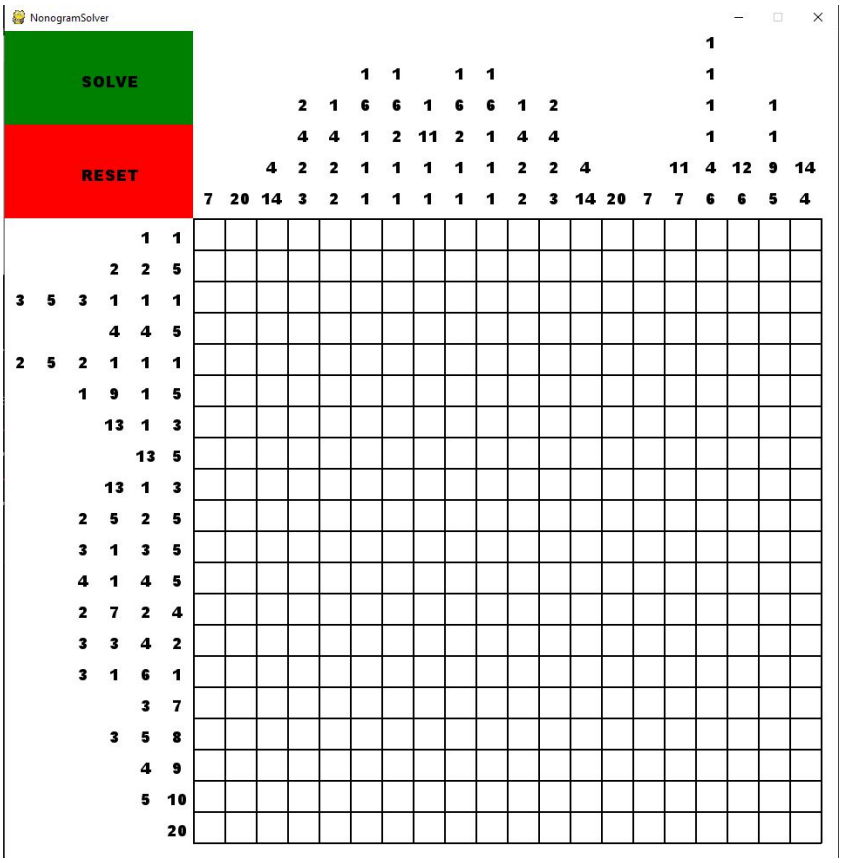


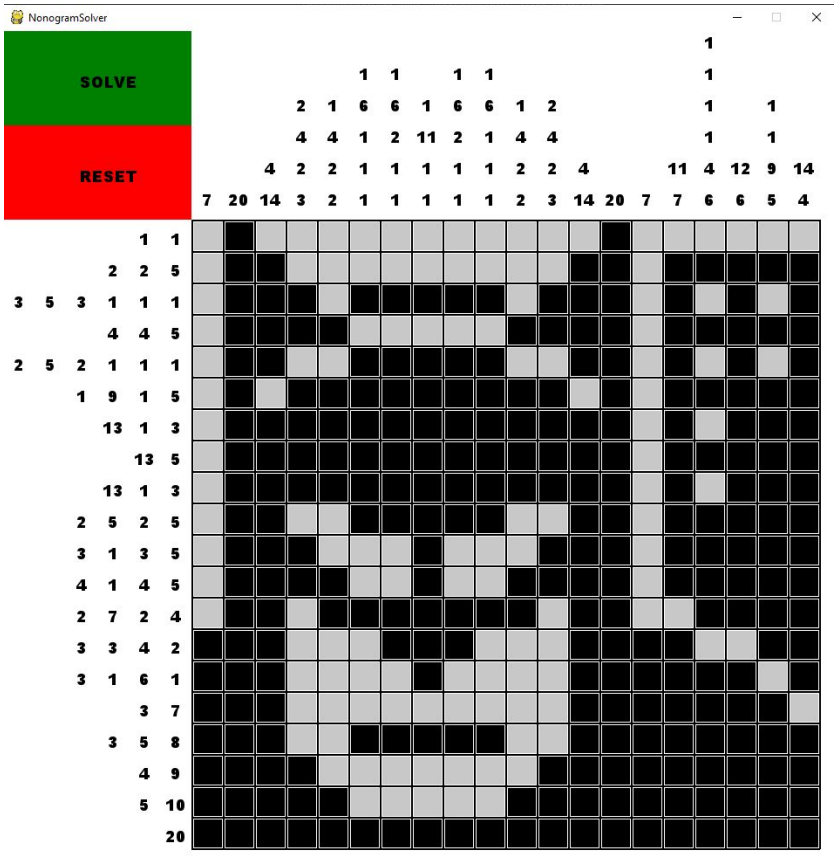
Możemy zakończyć w 2 sposoby:

- nieznane pola = 0 -> rozwiązaliśmy cały obrazek logiczny
- brak progress'u w iteracji -> obrazek jest niedeterministyczny i istnieje wiele rozw.

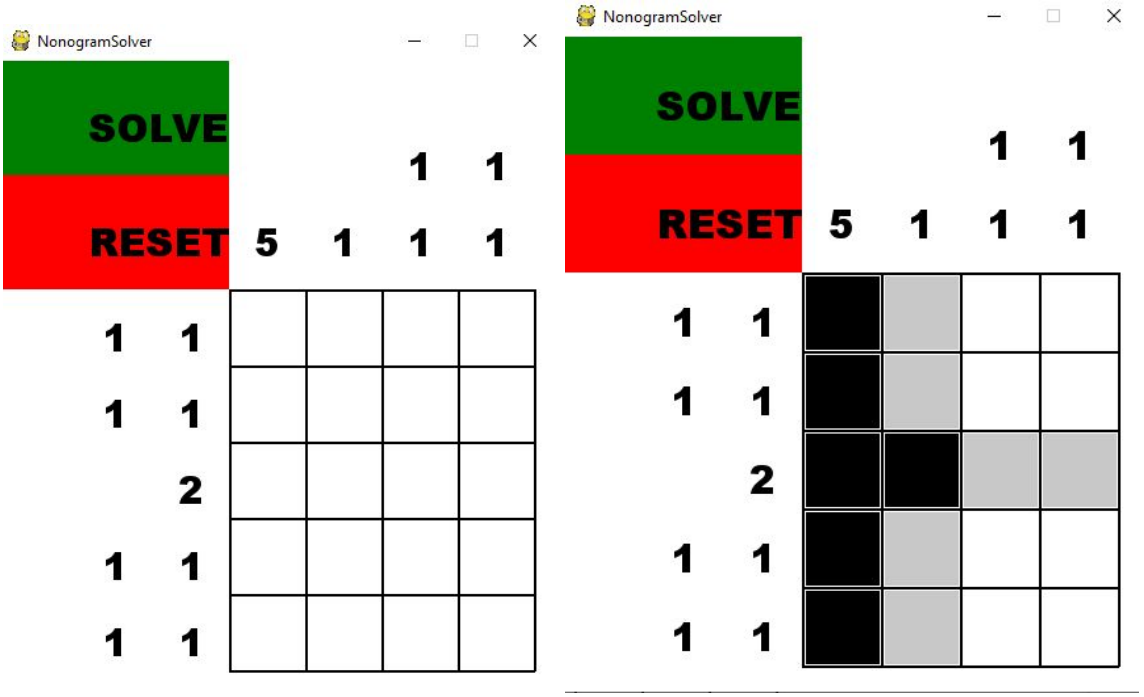
Działanie programu

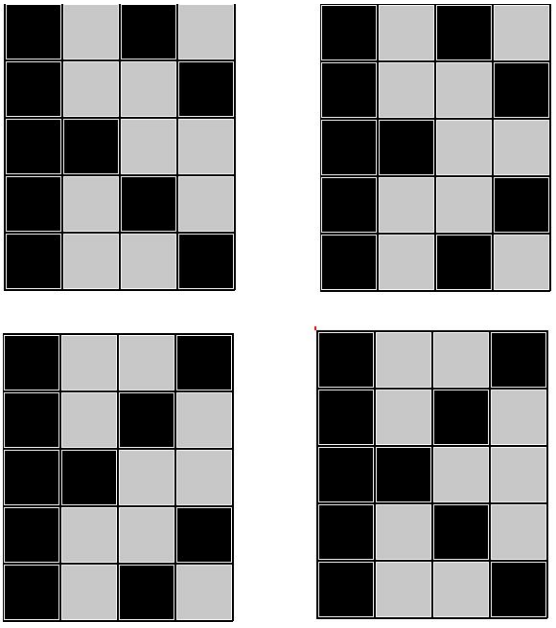
Przykładowe rozwiązanie dla obrazka logicznego


NonogramSolver
SOLVE
RESET
7 20 14 3 2 1 1 1 1 1 2 3 14 20 7 7 6 6 5 4
4 2 2 1 1 1 1 2 2 4 11 4 12 9 14
4 4 1 2 11 2 1 4 4 1 1
2 1 6 6 1 6 6 1 2 1 1
1 1 1 1 1 1
1 1
1 1 5
3 5 3 1 1 1
4 4 5
2 5 2 1 1 1
1 9 1 5
13 1 3
13 5
13 1 3
2 5 2 5
3 1 3 5
4 1 4 5
2 7 2 4
3 3 4 2
3 1 6 1
3 7
3 5 8
4 9
5 10
20


NonogramSolver
SOLVE
RESET
7 20 14 3 2 1 1 1 1 1 2 3 14 20 7 7 6 6 5 4
4 2 2 1 1 1 1 2 2 4 11 4 12 9 14
4 4 1 2 11 2 1 4 4 1 1
2 1 6 6 1 6 6 1 2 1 1
1 1 1 1 1 1
1 1
1 1 5
3 5 3 1 1 1
4 4 5
2 5 2 1 1 1
1 9 1 5
13 1 3
13 5
13 1 3
2 5 2 5
3 1 3 5
4 1 4 5
2 7 2 4
3 3 4 2
3 1 6 1
3 7
3 5 8
4 9
5 10
20

W przypadku obrazków niedeterministycznych:

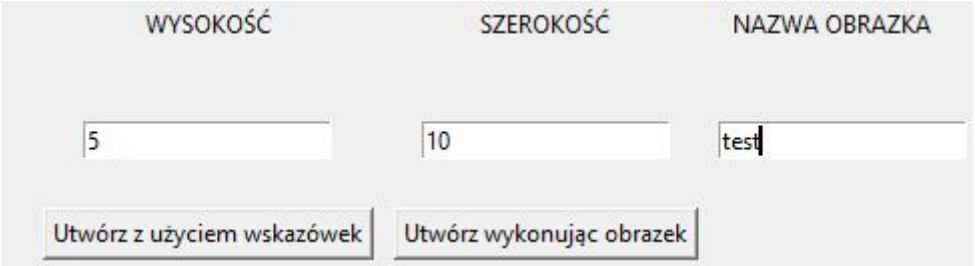




Wszystkie powyższe odpowiedzi byłyby poprawne i nie da się jednoznacznie określić poprawną(chociaż jako ludzie od razu widzimy, że autorowi chodziło o literę ‘K’, ale obrazek jest stworzony w sposób niedeterministyczny i nie możemy tego znaleźć jednoznacznie jeśli kierujemy się tylko zasadami)

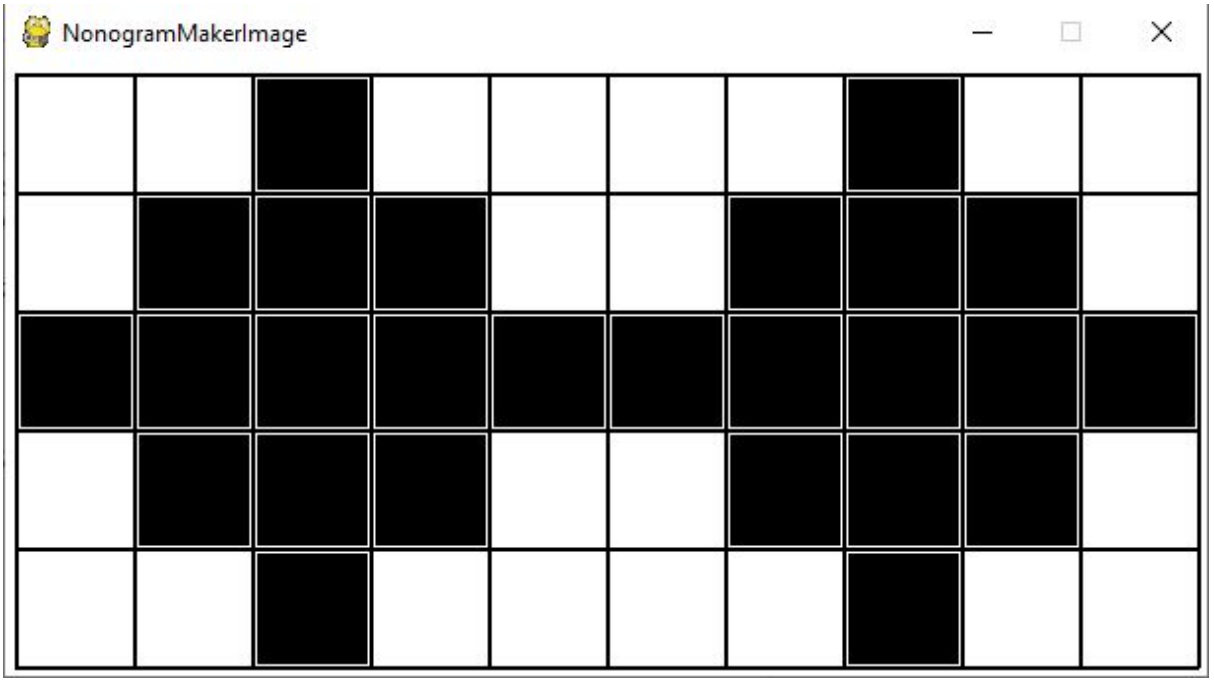
Tworzenie własnych obrazków logicznych

I Sposób: Podanie wymiarów obrazka i ręczne przepisanie wszystkich podpowiedzi



Sposób ten jest przydatny gdy nie możemy poradzić sobie z obrazkiem np. w krzyżówce w gazecie i chcemy, żeby program rozwiązał go za nas

II Sposób: Tworzenie własnego obrazka za pomocą zamalowania kratek



Program automatycznie wygeneruje nam plik z wejściowymi danymi w odpowiednim dla programu formacie, następnie będziemy mogli go otworzyć w programie i rozwiązać sami lub rozwiązać automatycznie (program nie gwarantuje, że utworzyliśmy deterministyczny obrazek logiczny)

Podsumowanie

Co wyszło:

- Udało się zrobić główny program do rozwiązywania obrazków logicznych
- W GUI możemy rozwiązywać zarówno sami, jak i możemy rozwiązywać automatycznie z użyciem programu
- Utworzenie własnego obrazka logicznego poprzez wykonanie obrazka - na początku nie przewidywałem nawet żeby to zrobić

Co nie wyszło:

- Na początku planowałem, żeby interaktywnie wprowadzać dane dla obrazka z użyciem wskazówek, w miarę podobny sposób, co rozwiązywanie obrazka, ale pozostałem przy prostszej wersji kolejnych inputów dla wierszy i kolumn

Wnioski na przyszłość:

- ważny jest dobór odpowiednich bibliotek do projektu, szczególnie do GUI, próbowałem kilka różnych, które okazały się średnio użyteczne do tego typu GUI graficznego, za to pygame bardzo dobrze nadawało się do projektu
- python oferuje masę bibliotek z olbrzymią liczbą funkcji, które potrafią niesamowicie uprościć zapis w wielu miejscach
- muszę dodawać więcej komentarzy do kodu, bo nawet gdy samemu piszę wszystko, to czasem po powrocie muszę się zastanowić dlaczego coś działa tak, a nie inaczej

źródła pomocnicze:
http://www.math.edu.pl/malowanie-liczbami