

Kombinatoryczna Teoria Liczb

12 b) Anty-Van der Waerden

Sylwia Nowak

Radosław Kutkowski

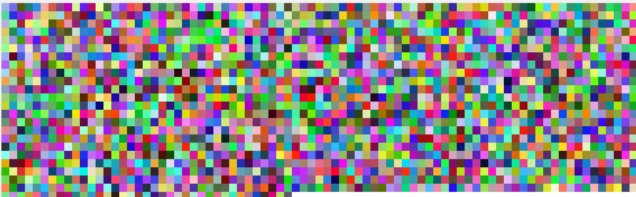
Roman Siry

Anty-Van der Waerden

Rozmiar zbioru liczb (n)

Długość tęczy wygrywająca (k)

Liczba dostępnych kolorów (c)



Poziom trudności gracza

Poziom zaawansowania strategii komputera

2 listopada 2016

1. Temat projektu

Dane wejściowe: liczba naturalna **n** liczba naturalna **k** liczba dostępnych kolorów **c**

Gracze na zmianę wybierają liczbę ze zbioru $[n]$ i kolorują ją na wybrany kolor.

Zwycięstwo:

- pierwszy gracz wygrywa, jeśli w otrzymanym pokolorowaniu zbioru $[n]$ będzie tęczyowy podciąg arytmetyczny o długości k (każdy wyraz tego ciągu musi mieć inny kolor)
- drugi gracz wygrywa, jeśli nie będzie takiego podciągu

Gra komputer kontra człowiek.

2. Wybrane technologie

Omawiany projekt napisany będzie w języku C#, na platformie .Net przy użyciu technologii WindowsForms. Wytworzona aplikacja pozwalająca użytkownikowi obcyć grę z symulowanym przeciwnikiem tzw. komputerem przeznaczona jest na dla systemu Windows 10. Jest to aplikacja desktopowa nie wymagająca dostępu do internetu.

3. Przechowywanie danych w aplikacji

```
public class Game
{
    public List<Color> Colors { get; set; }
    public List<Tuple<Tuple<Color, Boolean>, Boolean>> Fields { get; set; }
    public int K { get; set; }
    public Int32 HumanLevel { get; set; }
    public Int32 ComputerLevel { get; set; }
}
```

Do przechowywania stanu gry wykorzystywana jest powyższa struktura zawierająca w sobie:

- Listę dostępnych kolorów w grze - *Colors*. Rozmiar listy *Colors* odpowiada zmiennej c z definicji zadania.
- Listę par kolor-wartość logiczna *Fields* odpowiadających za przechowywanie informacji o aktualnym kolorze pola ze zbioru liczb $[n]$ o pewnym numerze. Wartość logiczna z głównej pary mówi o tym, czy użytkownik (gracz) może pokolorować dane pole. Wartość logiczna w wewnętrznej parze mówi o tym, czy dane pole jest podpowiadane użytkownikowi w danym kroku. Rozmiar listy *Fields* odpowiada zmiennej n z definicji problemu.
- Zmienną *K* reprezentującą zmienną k z definicji zadania.
- Wartość *HumanLevel* jest z zakresu $[0,4]$, wyznaczającą poziom trudności gry dla gracza. Wartość 0 reprezentuje brak możliwości uzyskania podpowiedzi, 4 – możliwość uzyskania podpowiedzi po każdym ruchu komputera.
- Zmienną *ComputerLevel* wyznaczającą stopień zaawansowania strategii komputera. Wartość jest z zakresu $[0,4]$, gdzie 0 to losowe kolorowanie w każdym ruchu a 4 to zaawansowana strategia dążąca do maksymalizacji szans na wygraną.

4. Interfejs użytkownika

Interfejs użytkownika zaprojektowany jest w taki sposób by umożliwić użytkownikowi parametryzowanie problemu. Dostępna jest możliwość zmiany wartości n , k , c . Po wyborze ilości kolorów c generowana jest tablica kolorów, które można będzie wykorzystać po rozpoczęciu gry. Dodatkowo dzięki panelowi przycisków odpowiedzialnych za poziom trudności można testować różne strategie komputera oraz algorytm podpowiedzi przeznaczonych dla użytkownika. Przycisk start jest odpowiedzialny za rozpoczęcie gry po wyborze parametrów rozgrywki.

Anty-Van der Waerden

Rozmiar zbioru liczb (n)

Długość tęczy wygrywająca (k)

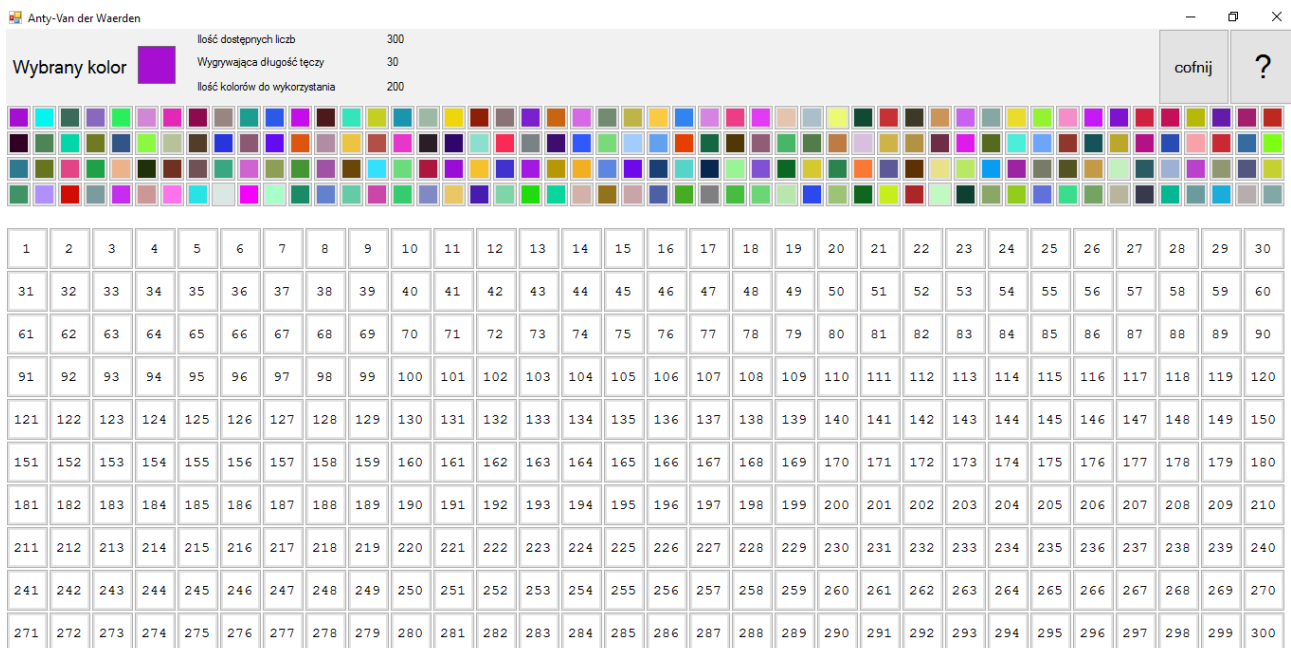
Liczba dostępnych kolorów (c)

Poziom trudności gracza

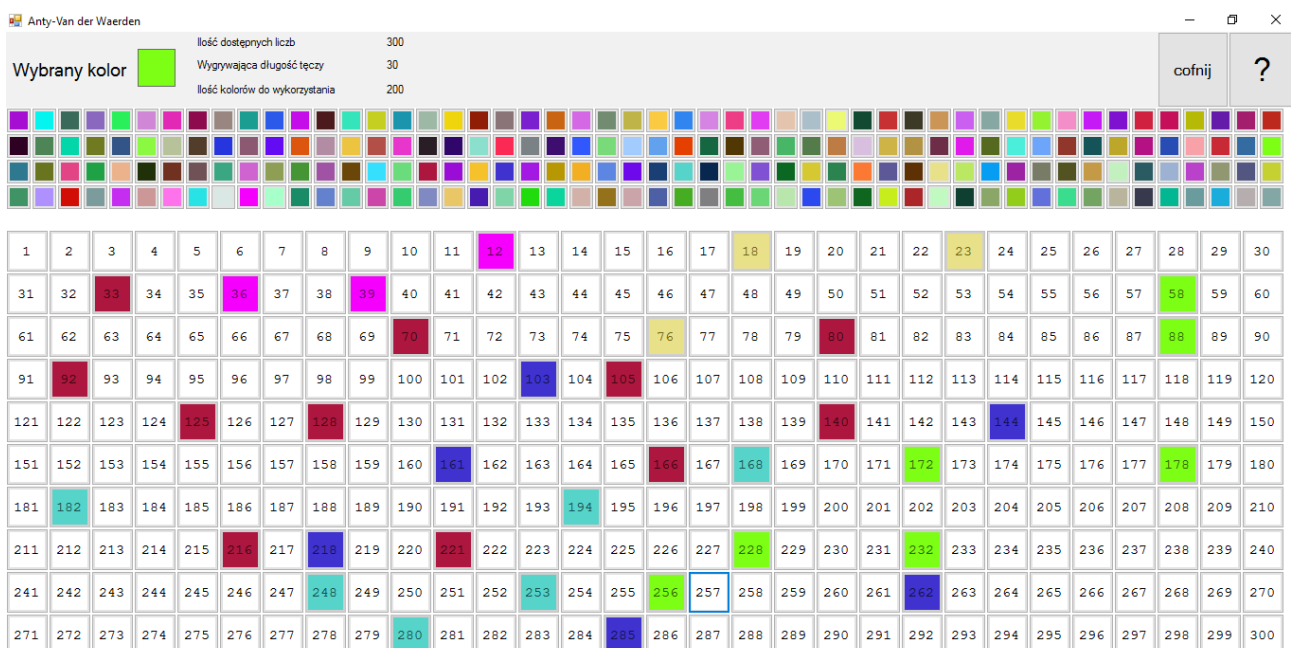
Poziom zaawansowania strategii komputera

START

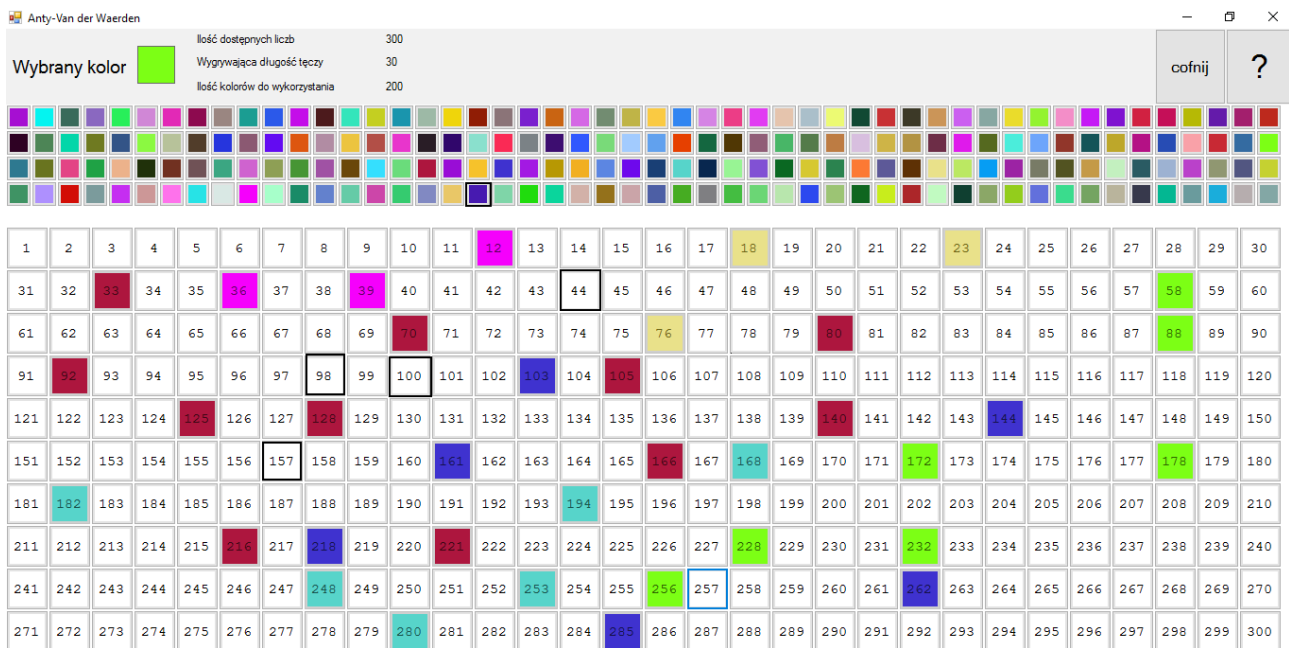
Po uruchomieniu okna rozgrywki w lewym górnym rogu widnieje aktualny kolor, którym gracz będzie kolorować wybrane pole. Aby zmienić kolor należy kliknąć w wybrany kolorowy kafelek. Podczas jednego ruchu gracz klika w pole zawierające numer. Pola, które zostały już pokolorowane nie mogą być pokolorowane ponownie.



Jeżeli użytkownik nie wybrał najtrudniejszego poziomu gry, w lewym górnym rogu znajduje się przycisk odpowiedzialny za ukazanie podpowiedzi dotyczących danego ruchu. Ruchy, które są optymalne w danej kolejce dla gracza zaznaczone są w sposób pogrubiony – dla pól i wybranego koloru.



Dodatkowo dla poziomu trudności gry dla gracza z przedziału $[0,3]$ możliwe jest używanie przycisku cofnij. Zamknięcie okna planszy gry powoduje powrót do okna parametryzacji gry.



5. Symulacja gry komputera

Zaproponowane rozwiązanie zawiera w sobie poziomy trudności gry związane z zaawansowaniem strategii komputera. Zaproponowane będą 4 strategie doboru pól ponumerowanych oraz kolorów w taki sposób by komputer, który rozpoczyna grę mógł ułożyć tęczę. Dodatkowo będzie istniała możliwość wyłączenia inteligencji komputera i zastąpienie jej losowym wyborem pól oraz kolorów do ich kolorowania. Ruch komputera będzie wykonywany po kliknięciu przez użytkownika danego pola z numerem. Jeżeli strategia komputera jest na tyle zaawansowana, że wyliczenie optymalnego ruchu dla komputera zajmuje powyżej 1s na ekranie pojawi się animacja ładowania w postaci kręcącego się kółka. Na czas wykonywania ruchu przez komputer interfejs jest blokowany na przyciski użytkownika.

6. Weryfikacja wyników

Podczas działania w aplikacji w czasie rzeczywistym po każdym ruchu komputera sprawdzany jest status planszy gry. Aby zweryfikować czy gracz numer 1 (komputer) wygrał należy sprawdzić czy istnieje ciąg k kolejnych liczb naturalnych pokolorowanych na k różnych kolorów. Takie sprawdzenie można wykonać w czasie $O(n \cdot k)$. Grę wygrywa gracz numer 2 (człowiek), jeżeli takiego ciągu nie da się osiągnąć. Oczywiście nie trzeba rozgrywać gdy aż do wykorzystania wszystkich liczb naturalnych z przedziału $[1, n]$. Wystarczy, po każdym ruchu gracza numer 2 sprawdzać czy nadal gracz 1 ma możliwość zbudowania k -elementowej tęczy. Do przedstawionej weryfikacji zastosujemy algorytm, który również działa w czasie $O(n \cdot k)$. Gra kończy się wyświetleniem informacji o zwycięzcy.

Aby zweryfikować poprawność strategii zaproponowanych zarówno dla gracza w formie odpowiedzi jak i samego symulatora ruchów dla gracza 1 – komputera stworzony będzie moduł testów, odpowiedzialny za weryfikację poszczególnych strategii dla komputera grających przeciwko strategiom generującej standardowo wskazówki dla gracza 2 – człowieka. Zostaną sporządzone wykresy w zależności od ilości kolorów c , wielkości n oraz k . Dzięki tak zaprojektowanemu modułowi testów możliwe będzie rozpoznanie, która strategia dla komputera powinna zostać oznaczona jako najlepsza (poziom trudności 4) oraz, który typ

wskazówek ułatwia grę człowiekowi najbardziej.

7. Harmonogram

1. **21.10.2016** Zaprojektowanie modelu przechowywania danych.
2. **28.10.2016** Zaprojektowanie interfejsu gry.
3. **4.11.2016** Implementacja interfejsu gry.
4. **11.11.2016** Zaprojektowanie architektury oraz modelu modułu testów.
5. **18.11.2016** Opis matematyczny wybranego problemu.
6. **25.11.2016** Implementacja prostej strategii gry komputera, polegającej na losowym wybraniu koloru oraz numeru pola kolorowanego oraz nieużytecznej strategii podpowiadania ruchu graczowi 2 – generator losowy ruchu. Symulacja gry w module testów.
7. **2.12.2016** Implementacja zaawansowanej strategii dla gracza 1 – komputera. Zastosowanie do testów algorytmu wybierającego losowe pole oraz kolor dla gracza 2.
8. **9.12.2016** Implementacja zaawansowanej strategii dla gracza 2 – testowanie strategii z losowym generatorem ruchu dla komputera.
9. **16.12.2016** Testowanie zaawansowanej strategii dla gracza 1 oraz gracza 2.
10. **23.12.2016** Rozbicie strategii gry dla komputera oraz człowieka na poziomy zaawansowania – wykorzystanie pewnego zbioru atrybutów wpływających na jakość strategii.
11. **30.12.2016** Wykorzystanie zaprojektowanych strategii w połączeniu z interfejsem użytkownika.
12. **6.01.2017** Raport z przeprowadzonych testów.
13. **13.01.2017** Przygotowanie prezentacji wyników oraz aplikacji.