****

Michał Kaliszczyk

nr albumu: 29958

kierunek studiów: Informatyka

specjalność: Systemy komputerowe i oprogramowanie

forma studiów: stacjonarne

Zastosowanie algorytmu wykorzystującego ruchy skoczka szachowego do szyfrowania chaotycznego plików graficznych.

The application of the algorithm that uses the movements of a chess knight to encrypt chaotic gźźraphic files.

praca dyplomowa inżynierska

napisana pod kierunkiem:

dr inz. Dariusz Burak

Data wydania tematu pracy:

Data złożenia pracy: Szczecin,

OŚWIADCZENIE

AUTORA PRACY DYPLOMOWEJ

Oświadczam, że praca dyplomowa inżynierska pn.

“Zastosowanie algorytmu wykorzystującego ruchy skoczka szachowego do szyfrowania chaotycznego plików graficznych”

napisana pod kierunkiem:

dr inz. Dariusza Buraka

jest w całości moim samodzielnym autorskim opracowaniem sporządzonym przy wykorzystaniu wykazanej w pracy literatury przedmiotu i materiałów źródłowych.

Złożona w dziekanacie Wydziału Informatyki treść mojej pracy dyplomowej w formie elektronicznej jest zgodna z treścią w formie ~~pisemnej~~/pisemnej i graficznej.

Oświadczam ponadto, że złożona w dziekanacie praca dyplomowa ani jej fragmenty nie były wcześniej przedmiotem procedur procesu dyplomowania związanych z uzyskaniem tytułu zawodowego w uczelniach wyższych.

.........................................

podpis dyplomanta

Szczecin, dn. ........................

\*niepotrzebne skreślić

STRESZCZENIE

Celem niniejszej pracy inżynierskiej zatytułowanej “ Zastosowanie algorytmu wykorzystującego ruchy skoczka szachowego do szyfrowania plików graficznych” jest zaprojektowanie oraz stworzenie aplikacji komputerowej przeznaczonej do szyfrowania plików graficznych. W pracy przedstawiony został projekt aplikacji oraz uzasadnienie potrzeby jej utworzenia. Przedstawiony został również opis użytych technologii, oraz została utworzona dokumentacja rozwiązania wraz z analizą możliwości zrównoleglenia operacji szyfrowania, deszyfrowania i generowania klucza szyfrującego. Aplikacja została również przetestowana pod kątem poprawności działania, efektywności oraz wykorzystania pamięci. Efektem realizacji założeń przedstawionych w niniejszej pracy jest działająca aplikacja z zaprojektowanym interfejsem użytkownika oraz testy, które są dostępne w publicznym repozytorium pod adresem: <https://github.com/Kali1994/Praca_Inzynierska>

SŁOWA KLUCZOWE**:** (TO DO)

ABSTRACT

(TO DO)

Spis treści

[1. WPROWADZENIE 5](#_Toc27423691)

[1.1 Motywacja 5](#_Toc27423692)

[1.2 Cel i założenia projektu 5](#_Toc27423693)

[1.3 Uzasadnienie wyboru algorytmu 6](#_Toc27423694)

[1.4 Zawartość rozdziałów 6](#_Toc27423695)

[2. Algorytm szyfrowania plików graficznych 7](#_Toc27423696)

[2.1 Wprowadzenie 7](#_Toc27423697)

[2.1 Opis ruchów skoczka szachowego 7](#_Toc27423698)

[2.2 Projekt scramblera 8](#_Toc27423699)

[2.3 Opis algorytmu 10](#_Toc27423700)

[2.3.1 Szyfrowanie 13](#_Toc27423701)

[2.3.2 Deszyfrowanie 14](#_Toc27423702)

[3. Algorytm przeszukiwania szachownicy 15](#_Toc27423703)

[3.1 Problem skoczka szachowego 15](#_Toc27423704)

[3.2 Wymagania dotyczące wymiarów szachownicy 15](#_Toc27423705)

[3.2.1 Rozmiary planszy do rozwiązania Zamkniętego 16](#_Toc27423706)

[3.2.2 Rozmiary planszy do rozwiązania Otwartego 20](#_Toc27423707)

[3.3 Metody w znajdowaniu ścieżki 20](#_Toc27423708)

[3.4 Wykorzystanie w Kryptografii 25](#_Toc27423709)

[3.5 Algorytmy rozwiązujące problem skoczka szachowego 25](#_Toc27423710)

[3.5.1 Brute-force 25](#_Toc27423711)

[3.5.2 Divide and conquer 25](#_Toc27423712)

[3.5.3 Warnsdorff’s rule 26](#_Toc27423713)

[3.5.4 Neural network solution 27](#_Toc27423714)

[3.5.5 Analiza algorytmów - wybór najbardziej optymalnego. 27](#_Toc27423715)

[4. Przedstawienie wykorzystanych technologii 28](#_Toc27423716)

[4.1 Język C++ 28](#_Toc27423717)

[4.2 Język C# 28](#_Toc27423718)

[4.3 Środowisko programistyczne 29](#_Toc27423719)

[4.4 System kontroli wersji 29](#_Toc27423720)

[4.5 Visual Paradigm 31](#_Toc27423721)

[4.6 Matlab 31](#_Toc27423722)

[5. Projekt Aplikacji 32](#_Toc27423723)

[5.1 Wprowadzenie 32](#_Toc27423724)

[5.2 Interfejs graficzny użytkownika 32](#_Toc27423725)

[5.3 Model wymagań 32](#_Toc27423726)

[5.3.1 Wymagania funkcjonalne 32](#_Toc27423727)

[5.3.1.1 Diagram przypadków użycia 32](#_Toc27423728)

[5.3.1.2 Diagram sekwencji 32](#_Toc27423729)

[5.3.1.3 Diagram aktywności 32](#_Toc27423730)

[5.3.2 Model projektowy 32](#_Toc27423731)

[5.3.2.1 Diagram Klas 33](#_Toc27423732)

[5.3.2.2 Diagram ERD 33](#_Toc27423733)

[5.4 Testy 33](#_Toc27423734)

[6. Opis Implementacji 34](#_Toc27423735)

[6.1 Wstęp 34](#_Toc27423736)

[6.2 Opis klas 34](#_Toc27423737)

[6.3 Szczegółowy opis metod 34](#_Toc27423738)

[7. Analiza 34](#_Toc27423739)

# 1. WPROWADZENIE

## 1.1 Motywacja

W dzisiejszych czasach zabezpieczanie danych na komputerze jest jedną z ważniejszych rzeczy o ,którym każdy użytkownik powinien pamiętać. Wynika to z tego ,że coraz więcej informacji przechowywanych jest na naszym komputerze nie tylko prywatnych ,ale także służbowych, dlatego bardzo istotne jest żeby zadbać o ich bezpieczeństwo. Szczególnie istotnym jest utrata danych ,które przechowujemy na naszych urządzeniach, ponieważ mogą się dostać do niepowołanych osób w przypadku kradzieży komputera, lub kiedy nasze urządzenie zostanie zainfekowane przez złośliwe oprogramowanie. Dlatego bardzo ważne jest żeby każdy użytkownik ,który posiada poufne informacje szyfrował je. Dzięki temu mamy zabezpieczenie ,że tylko odpowiednie osoby będą wstanie odczytać informacje a w przypadku gdy zostaną one skradzione przez jakąś osobę nie zostaną odczytane ani w żadnym stopniu wykorzystane bez naszej zgody.

## 1.2 Cel i założenia projektu

Celem projektu jest stworzenie aplikacji ,która będzie skutecznie i w jak najkrótszym czasie szyfrowała i odszyfrowywała pliki graficzne. Bardzo ważne jest żeby chronić zdjęcia tak samo jak inne informacje, ponieważ powszechnie wiadomo ,że fotografie często zawierają ogromne ilości informacji jak na przykład zapisane hasła ,dane osobowe oraz informacje o naszych znajomych, rodzinie. W przypadku przedostania się zdjęcia do niepowołanej osoby nie będzie ona wstanie dowiedzieć się co jest zapisane na obrazie oraz odszyfrować go nie znając klucza. Dzięki temu użytkownicy będą mogli bezpiecznie przechowywać swoje zdjęcia na komputerze oraz przesyłać je między sobą bez obawy o wykorzystanie ich przez niepowołane osoby. Założeniem projektu jest użycie techniki szyfrowania obrazów kolorowych poprzez transpozycję pikseli między kanałami RGB wykorzystując ruchy skoczka szachowego i mapy chaotycznej do generowania reguł dla każdego piksela w obrazie. [1]

## 1.3 Uzasadnienie wyboru algorytmu

Wybrałem ten algorytm ,ponieważ wykorzystuje on ruchy skoczka szachowego z gry w szachy co jest ciekawym sposobem szyfrowania oraz używa on generatora opartego na chaosie do losowanie bitów pseudolosowych z ,których tworzone są reguły do przemieszczania się konika szachowego posiada on dobre cechy losowości .W wyniku tego otrzymujemy efektywnie zaszyfrowany obraz ,ponieważ mamy niski współczynnik korelacji sąsiednich pikseli .

## 1.4 Zawartość rozdziałów

(TO DO)

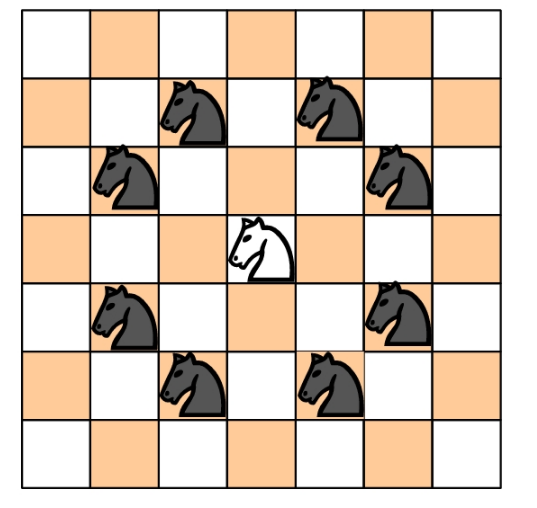
# 2. Algorytm szyfrowania plików graficznych

## 2.1 Wprowadzenie

Jest to algorytm służący do szyfrowania plików graficznych ,który wykorzystuje technikę scramblingu do transpozycji pikseli, między kanałami RGB. Używane są do tego ruchy skoczka szachowego oraz model skramblera opartego na chaosie. [1]

## 2.1 Opis ruchów skoczka szachowego

Konik szachowy jest to figura w popularnej grze szachy, która porusza się w kształcie litery “L” Przemieszcza się pionowo o dwa pola a następnie poziomo o jedno pole lub poziomo o dwa pola a następnie pionowo o jedno pole, można to zrobić w dowolnym kierunku. Maksymalna liczba możliwych ruchów konika szachowego wynosi 8. Możliwe ruchy zostały przedstawione na rysunku poniżej. [1]



**Rysunek 1 Ruchy skoczka szachowego**

Żródło: https://www.hindawi.com/journals/mpe/2014/932875/fig1/

## 

## 2.2 Projekt scramblera

Scrambler jest to algorytm, który służy do randomizacji sekwencji bitowych. Jego głownym celem jest osiągniecie jak najbardziej prawdziwie losowej sekwencji bitów. Do realizacji tego zadania został stworzony nowy chaotyczny dyskretny system zbudowany na funkcjach trygonometrycznych, którego zamiarem jest użycie go w aplikacjach kryptograficznych. Ten system posiada chaotyczne zachowanie dla dużej przestrzeni wartości parametrów i bardzo dobre właściwości statystyczne. Ponadto jest on używany w połączeniu z operacją binarną do projektowania nowego modelu generatora bitów pseudolosowych (PRBG). Proponowany model systemu został zdefiniowany następująco:

h - przedstawia ograniczoną rzeczywistą mapę , która ogranicza przestrzeń fazową do zamkniętego przedziału dzięki temu układ dynamiczny ma dobre właściwości chaotyczne.

f - przedstawia okresową rzeczywistą mapę, która została wybrana tak , żeby zapewnić dużą przestrzeń fazową.

Na podstawie modelu został zaprojektowany jednowymiarowy dyskretny system dynamiczny, który jest zdefiniowany następująco:

arctg - został wybrany , żeby ograniczyć przestrzeń fazową do bliskiego przedziału , w którym układ dynamiczny ma dobre właściwości chaotyczne.

ctg - został wybrany , żeby zapewnić dużą przestrzeń fazową

r - parameter kontrolny

x - parametr początkowy

Są to wartości na podstawie ,których generowane będą liczby rzeczywiste.

(Poziom wrażliwości parametru kontrolnego i parametru początkowego. wartości w jakim przedziale się znajdują plus wytłumaczenie dlaczego TO DO )

W prezentowanym systemie zostały użyte dwie chaotyczne mapy w celu zapewnienia większego bezpieczeństwa, gdyż użycie jednej mapy chaotycznej może ujawnić pewne informacje o ich podstawowym układzie chaotycznym. ( TO DO jakie informacje ?)

Zostały one zaprojektowane na podstawie przedstawionego wcześniej modelu.

Generator liczb pseudolosowych otrzymujemy poprzez zastosowanie operacji binarnej do chaotycznych map. Podana jako wzór.

gdzie

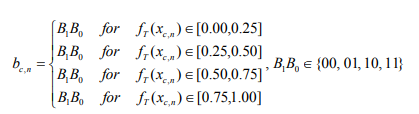
Dzięki czemu otrzymujemy połączone mapy chaotyczne przedstawione poniżej:

Gdzie wartości są wartościami początkowymi a są parametrami kontrolnymi.

Wartości uzyskuje się za pomocą rekurencji przypisując wynik równania.

, dla wszystkich liczb .

Jako wynik uzyskujemy liczby rzeczywiste , które zostają poddane metodzie dyskretyzacji ,żeby otrzymać generator liczb pseudolosowych. [1][2]



Jeżeli wartość Y znajduje się w przedziale od 0 do 0.25 to zapisujemy do pierwszego i drugiego pliku wartość 0

Jeżeli wartość Y znajduje w przedziale od 0.25 do 0.50 to zapisujemy do pierwszego pliku wartość 0 a do drugiego wartość 1

Jeżeli wartość Y znajduje się w przedziale od 0.50 do 0.75 to zapisujemy do pierwszego pliku wartość 1 a do drugiego wartość 0

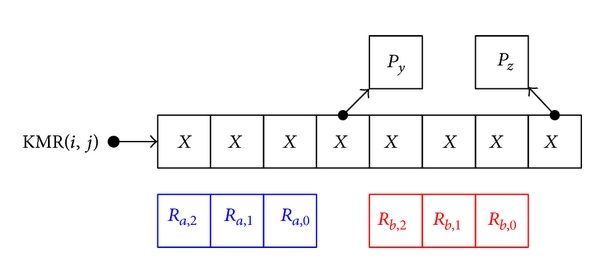
Jeżeli wartość Y znajduje się w przedziale od 0.75 do 1.00 to zapisujemy do pierwszego i drugiego pliku wartość 1

Odpowiednio od przekazanego wyniku z równania Y otrzymujemy dwa bity, z których jeden jest zapisywany do pierwszego pliku a drugi do drugiego pliku tak jak to zostało przedstawione powyżej. W rezultacie otrzymujemy dwa pliki z wygenerowanymi bitami. [3]

## 2.3 Opis algorytmu

Algorytm rozpoczynamy od wczytania zdjęcia w celu pobrania jego wymiarów oraz wartości pikseli. Odpowiednio od pobrania wysokości i szerokości obrazu generujemy dwa pliki, w których zawarte są sekwencje zer i jedynek przy użyciu scramblera opisanego w podpunkcie 2.2. Na podstawie wielkości obrazu generujemy odpowiednią liczbę bitów. Jeżeli zdjęcie ma wymiary 512 x 512 to w każdym pliku powinno się znajdować 6 291 456 bitów, ponieważ dla każdego koloru (RGB) będziemy potrzebować 8 bitów. Te sekwencje potrzebne będa do ustalenia ilości przemieszczeń skoczka szachowego, jego kierunku ruchu oraz transpozycji wartości koloru. Tworzymy dwie tablice trójwymiarowe KMR i RNS w, których będziemy przechowywać reguły. Wielkość tych tablic jest taka sama do jakiej zapisaliśmy wartości pikseli obrazu dla przykładu jeżeli posiadamy zdjęcie o wymiarach 512 na 256 to wtedy tworzyme tablice o wymiarach 512 x 256 x 3 dla każdego koloru RGB (czerwonego, zielonego, niebieskiego) i do każdej komórki zapisujemy po 8 bitów. Z pierwszego pliku zapisujemy wartości do tablicy KMR a z drugiego do tablicy RNS.

Dla tablicy KMR robimy to następująco



**Rysunek 2 Generowanie reguł z tablicy KMR**

Żródło: https://www.hindawi.com/journals/mpe/2014/932875/fig3/

Reguły jakie otrzymujemy z tablic KMR są następujące: Ra, Rb, Py, Pz

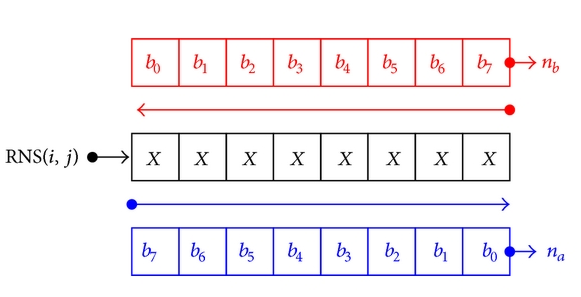
Pz jest pobierana z pierwszego bitu

Rb jest pobierana od 2 do 4 bitu

Py jest pobierana z 5 bitu

Ra jest pobierana od 6 do 8 bitu

Natomiast dla tablicy RNS



**Rysunek 3 generowanie reguł z tablic RNS**

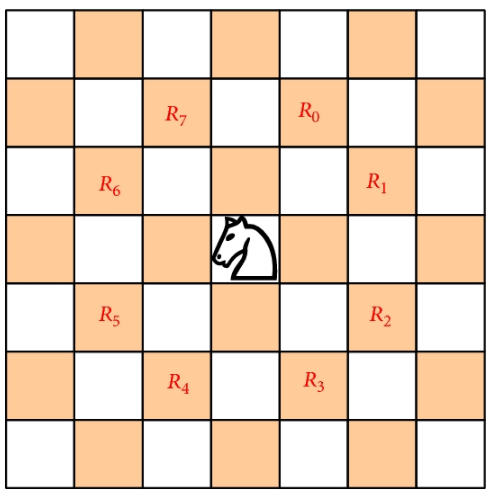
Żródło: https://www.hindawi.com/journals/mpe/2014/932875/fig3/

Reguły jakie otrzymujemy z tablic RNS są następujące: Na, Nb

Na jest pobierana od pierwszego do ostatniego bitu

Nb jest pobierana od ostatniego do pierwszego bitu

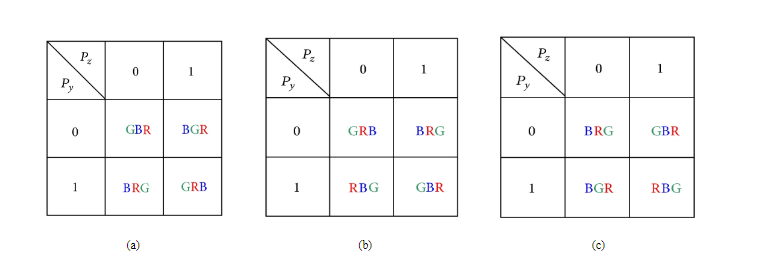
Zmienne **Ra** i **Rb** wskazują nam kierunek ruchu skoczka szachowego natomiast ze zmiennej **Na i Nb** dostajemy liczbę ile razy ruch ma zostać wykonany w danym kierunku. Jeżeli ruch skoczka osiągnie granice obrazu wtedy wychodzi z drugiej strony zachowując przy tym naturalny kształt ruchu.



**Rysunek 4 Kierunek ruchu skoczka**

Żródło: https://www.hindawi.com/journals/mpe/2014/932875/fig1/

Przenoszenie wartości między kanałami kolorów czyli transpozycje dla danego piksela. wykonujemy przy użyciu zmiennych **Py** i **Pz** , które przyjmują wartości zero lub jeden i przenoszą wartości tak jak na przedstawionych zdjęciach.



**Rysunek 5 Zasady transpozycji pikseli**

Żródło: https://www.hindawi.com/journals/mpe/2014/932875/fig2/

a - przedstawia kanał koloru czerwonego.

b - przedstawia kanał koloru zielonego.

c - przedstawia kanał koloru niebieskiego.

Dla przykładu jeżeli aktualnie szyfrujemy piksel wartości koloru czerwonego i wylosowaliśmy Py = 0 i Pz = 1. W takim wypadku do wartości koloru czerwonego z obecnego piksela wpisujemy wartość koloru niebieskiego z trzeciego piksela i odwrotnie. Natomiast wartość drugiego piksela pozostaje bez zmian. Czynności, które zostały przedstawione są wykonywane tak samo przy szyfrowaniu oraz odszyfrowywaniu obrazu. [1]

## 2.3.1 Szyfrowanie

Przy szyfrowaniu wprowadzamy wartości kontrolne i startowe do scramblera i na podstawie ich posiadamy wygenerowane dwa pliku z sekwencja bitów, z których zostały stworzone tablice gdzie znajdują się reguły. Szyfracje rozpoczynamy od początku obrazu i pierwszego koloru (czerwonego, zielonego, niebieskiego). Od piksela, na którym się aktualnie znajdujemy wyszukujemy za pomocą reguł dwa kolejne piksele pierwszy przy użyciu zmiennej **Ra** i **Na** natomiast drugi od pozycji znalezionego piksela przy pomocy zmiennej **Rb** i **Nb**. W tym momencie posiadamy współrzędne trzech pikseli obecnie na ,którym się znajdujemy oraz dwa ,które otrzymaliśmy przy pomocy wygenerowanych reguł. Ostatnim krokiem jaki musimy wykonać jest przenoszenie wartości kolorów między znalezionymi pikselami do tego używamy zmiennej **Py** i **Pz**. Czynności te powtarzamy dla każdego piksela i jego kanałów kolorów. Jako wynik powinniśmy otrzymać zaszyfrowany obraz. [1]

## 2.3.2 Deszyfrowanie

Przy deszyfracji wprowadzamy poprawny klucz , czyli wartości kontrolne do scramblera, jakie zostały użyte do szyfrowania obrazu oraz zaszyfrowane zdjęcie. Deszyfracje zaczynamy od końca obrazu i ostatniego koloru czyli (niebieski, zielony ,czerwony). Znajdujemy za pomocą reguł dwa piksele jak w przypadku szyfrowania do zamiany wartości kolorów, tylko ,że pierwszy piksel otrzymujemy najpierw przez zmienna **Rb** z pomocą zmiennej **Nb** a trzeci piksel przez zmienna **Ra** przy użyciu zmiennej **Na**. Ostatnią różnicą jaką musimy zmienić przy odszyfrowaniu jest zmienić kierunek zmiennej **Rb** i **Ra** na przeciwny. Czyli jak uzyskaliśmy kierunek R3 to w odszyfrowaniu wykonujemy ruch kierunkiem R7 , który jest przeciwny. Czynności te powtarzamy dla każdego piksela i jego kanałów kolorów. Jako wynik powinniśmy otrzymać obraz sprzed szyfrowania. [1]

# 3. Algorytm przeszukiwania szachownicy

## 3.1 Problem skoczka szachowego

Jest to problem matematyczny, który polega na znalezieniu takiej trasy na szachownicy tak aby skoczek odwiedził każde pole dokładnie jeden raz, używając dozwolonych ruchów. Wyróżnione są dwa rodzaje rozwiązań przy znajdowaniu ścieżki.

* Zamknięte
* Otwarte

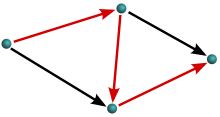
Rozwiązanie zamknięte polega na tym, że skoczek musi odwiedzić wszystkie pola na szachownicy, a ścieżka kończy się w miejscu z ,którego ma możliwość dotarcia do pozycji startowej jednym ruchem. Pozwala to na przechodzenie planszy nieskończoną ilość razy.

Rozwiązanie otwarte polega na tym ,że skoczek odwiedza każde pole, ale ścieżka kończy się na pozycji z, której nie jest w stanie za pomocą jednego ruchu dostać się na pole startowe. [5][6]

## 3.2 Wymagania dotyczące wymiarów szachownicy

Szachownica jest to plansza w kształcie kwadratu. Klasycznie ma wymiary 8x8 czyli składa się z 8 rzędów i 8 kolumn. Poniżej zostaną przedstawione dowody dla których plansz istnieje ścieżka dla rozwiązania zamkniętego oraz otwartego. [5]

Aby przedstawić dowody rozwiązań wpierw należy opisać pojęcie grafu Hamiltona. Jest to termin wywodzący się z teorii grafów. Związany jest z poszukiwaniem tak zwanej ścieżki Hamiltona - drogi przechodzącej przez wszystkie wierzchołki tylko jeden raz. Rozróżnia się szczególny przypadek ścieżki zamkniętej - zawierający pełny cykl Hamiltonowski. Rozwiązanie problemu skoczka sprowadza się do przedstawienia pola szachownicy w postaci grafu oraz znalezienia wyżej opisanej ścieżki.

**Rysunek 6 Graf hamiltonowski**

Żródło: https://pl.wikipedia.org/wiki/Graf\_hamiltonowski

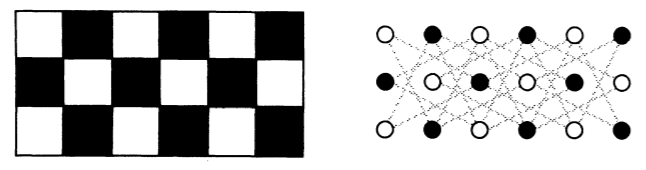
Przed zdefiniowaniem problemu najpierw należy poczynić pewne założenia. Przyjmijmy kwadratową tablice o wymiarach m x n, gdzie m to liczba komun a n to liczba wierszy podzielona na komórki o oznaczeniach i oraz j licząc od górnej lewej krawędzi. Dozwolony ruch skoczka szachowego to przesunięcie go o 2 komórki horyzontalnie lub wertykalnie a następnie o 1 komórkę w kierunku prostopadłym do poprzedniego w obrębie dostępnych pól co można zapisać w postaci i +- 2, j+-1 lub i+- 1 j +- 2. Stąd nakreśla się pytanie: dla jakich wymiarów szachownicy skoczek może wykonywać kolejne ruchy tak aby zajął każde pole tylko raz oraz żeby zakończył podróż w miejscu jej startu.

Jest to problem konika szachowego w wariancie zamkniętym

### 3.2.1 Rozmiary planszy do rozwiązania Zamkniętego

Jak wcześniej wspomniano problem rozwiązuje się sprowadzając szachownice do postaci grafu. Pola szachownicy o zmiennych kolorach białym i czarnym zamieniono na wierzchołki, z których białe oznaczają pola parzyste a czarne nieparzyste Jak mozna zauwazyc krawędzie łączą wierzchołki o przeciwnych kolorach - jest to przypadek bidiagramu. Stąd można wyciągnąć pierwszy wniosek:

* liczba pól białych jest równa liczbie pól czarnych, stąd liczba krawędzi jak i ścieżek musi być parzysta



W tym wariancie istnieje szereg wymagań które musi spełnić szachownica.  
Poniżej przedstawiono warunki, z których spełnienie minimum jednego wyklucza pozytywne rozwiązanie.

Dla szachownicy o wymiarach m x n (gdzie m <= n)

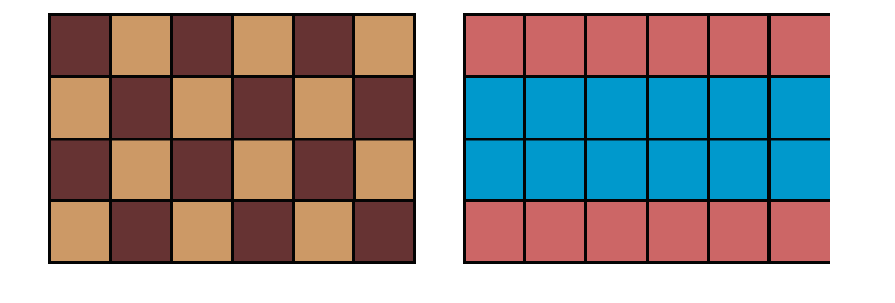
1. wymiary m oraz n są nieparzyste
2. m = 1, 2 lub 4
3. m = 3 oraz n = 4, 6, 8

Dowod 1.

Wymiary tablicy muszą być parzyste jak przedstawiono we wniosku pierwszym. Iloczyn liczb nieparzystych daje liczbe nieparzysta. Dlatego aby konik szachowy wrócił na swoje miejsce liczba pól musi być parzysta.

Dowod 2.

Dla m = 1, 2 problem jest oczywisty. Wymiary szachownicy nie zapewniają minimalnej przestrzeni dla ruchu skoczka. Sytuacja jest nieco bardziej skomplikowana dla m = 4. Do zobrazowania problemu należy spojrzeć na dwa obrazki



**Rysunek 7 Dla tablic 4 x n nie istnieje rozwiązanie zamknięte**

Pierwszy przedstawia szachownicę o wymiarach 4 na n. Na drugim pola podzielono na 2 kolory. Liczba pól jest jednakowa dla obu kolorów. Ponadto pola czerwone dostępne są tylko z pól niebieskich co implikuje konieczność zmiany kolorów pola z każdym skokiem Ruch z pola niebieskiego na niebieskie jest więc nie dozwolony. Drugi wniosek można sformułować porównując kolejne pola skoczka na obu rysunkach zaczynając z pola (1,1) skacząc na pole (3,2) przeskakujemy z pola czarnego/czerwonego na pole białe/niebieskie kolejnym skoku skaczemy na pole czarne/czerwone i tak dalej. Wynika z tego że pole (2,2) o kolorze białym/czerwonym nigdy nie zostanie pokryte. Empirycznie dowód można przedstawić analizując ruch skoczka na podstawie rysunku

To Do

. Zaczynając podróż od pola (1,1), zeby dostac sie na pole (2,2) skoczek musiałby skoczyć z pola pomarańczowe na pomarańczowego co jest niedozwolone.

Na rysunku przedstawiono problem niedozwolonego ruchu:

1. błąd skoku z pola niebieskiego na niebieskie
2. skok na pole (2,2) przy wymaganiach związanych z podziałem na pola niebieskiego i na czerwone wymaga dodatkowego pola

To do

Dowod 3

Aby udowodnić podpunkt c) należy posłużyć się pewnymi założeniami. Wariant 3 x 4 został wykluczony w poprzednim dowodzie. Dla każdego cyklu hamiltonowskiego przy usunięciu dowolnego zestawu k wierzchołków pozostaje maksymalnie k połączonych komponentów. Dla grafu 3 x 6 przy usunięciu wierzchołków (1,3) i (3,3) pozostają 3 połączone komponenty.

(To do zdjecie)

Dla grafu 3 x 8 można wyróżnić 8 wierzchołków z dwoma połączeniami co daje łącznie 16 krawędzi które muszą znaleźć się w ścieżce. Dla ułatwienia rozważań zbudowany został nowy graf składający się z wierzchołków wolnych oraz tych kończących pewne krawędzie. Cykl Hamiltona z oryginalnego grafu musi zawierać się w grafie pochodnym, co jak widać jest sprzecznością, co zatem dowodzi prawdziwości tezy.



Wszystkie inne wymiary szachownic spełniają wymagania i zawierają ścieżkę Hamiltona. Kluczem jest znalezienie metody która pozwala tworzyć małe drogi, które byłyby częścią głównej ścieżki.[5][6]

TODO : przykłady + dowody co musi zawierać

### 3.2.2 Rozmiary planszy do rozwiązania Otwartego

W tym rozwiązaniu użyjemy grafów jako dowody dla ,jakich szachownic istnieje rozwiązanie. Cull and de Curtins udowodnił, że we wszystkich szachownic o wymiarach m x n szachownicy gdzie 5 <= m <= n. istnieje rozwiązanie problemu. Natomiast Van Rees przedstawił, że dla plansz o wymiarach 3 x n gdzie n = 4 lub n >= 7 również istnieje otwarta ścieżka. Pozwala nam to na sformułowanie twierdzenia dla jakich plansz nie ma rozwiązania jeżeli jakiś z podanych warunków jest spełniony .[10]

Twierdzenie

1) m = 1 lub 2

2) m = 3 i n = 3, 5 ,6

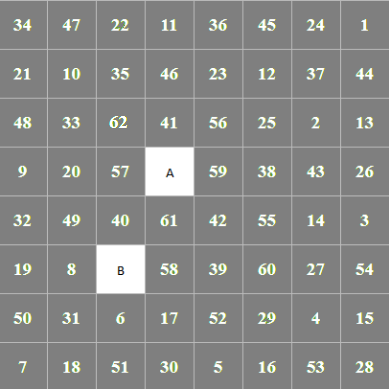
3) m = 4 i n = 4

Dowody i większy opis TO DO

## 3.3 Metody w znajdowaniu ścieżki

Euler przedstawia rozwiązanie problemu przeszukiwania szachownicy dla planszy o wymiarach 8x8, co daje nam szachownice składającą się z 64 pól. Trasa jaką musi pokonać skoczek składa się z 63 ruchów, co skutkuje odwiedzeniem każdego pola tylko jeden raz. Liczba ścieżek jaką może poruszać się skoczek żeby osiągnąć oczekiwany rezultat jest skończona.Ta metoda jest skuteczna kiedy startujemy z dowolnego rogu planszy.

Metoda otwarta wymaga od nas ustawienia konika na polu startowym czyli w dowolnym rogu szachownicy. Następnie używając reguł skoczka szachowego staramy się wykonać jak najwięcej możliwych ruchów pamiętając że nie możemy odwiedzić tego samego pola więcej niż jeden raz. Przy każdym przemieszczeniu się konika musimy zapisywać aktualny numer ruchu a pozostałe pola ,które nie zostały odwiedzone przez nas oznaczamy literami. Wykonujemy te operacje żebyśmy byli w stanie odtworzyć jego ruchy oraz określić, które pola nie zostały odwiedzone. Na rysunku 3 przedstawiona została trasa skoczka, który wykonał 61 ruchów i pozostały dwa nieosiągalne pola.



**Rysunek Niekompletna trasa rozwiązanie otwarte**

W momencie kiedy nie możemy wykonać ruchu musimy określić, które pola muszą zostać użyte aby umożliwić nam dotarcie do miejsc, które nie zostały wcześniej odwiedzone są one oznaczone literami A i B. Następnie musimy określić do jakich pól skoczek jest w stanie przejść z ostatniej pozycji. W wyniku dostajemy numery miejsc takich jak:

21, 47, 11, 23, 59, 61, 49 oraz 9.

Teraz musimy podzielić drogę od startu do podanych pól oraz sprawdzić czy umożliwi nam to dojście do pola A lub B.

W rezultacie otrzymujemy takie drogi.

1. 1 -> 21 oraz 62 -> 22
2. 1 -> 47 oraz 62 -> 48
3. 1 -> 11 oraz 62 -> 12
4. 1 -> 23 oraz 62 -> 24 Pole 23 prowadzi do miejsca A
5. 1 -> 59 oraz 62 -> 60
6. 1 -> 49 oraz 62 -> 50 Pole 50 prowadzi do miejsca B
7. 1 -> 9 oraz 62 -> 10

Z czego wybieramy dowolną drogę ,która przechodzi przez punkt ,który jest nie odwiedzony.

1 -> 49 oraz 62 -> 50 -> B

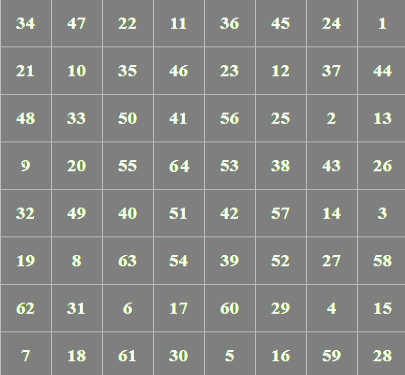
Pole B umożliwia nam przejście na następujące miejsca.

18, 50, 32, 20, A, 42, 52, 30.

Mamy możliwość przejścia z pola B bezpośrednio do pola A.

1-> 49 oraz 62 -> 50 -> B -> A

Teraz mamy kompletną trasę ,która prezentuje się na rysunku 4.



**Rysunek Kompletna trasa rozwiązanie otwarte**

Metoda zamknięta, warunkiem użycia tej metody jest posiadanie trasy ,która obejmuje wszystkie pola szachownicy.Dla przykładu użyjemy wyniku z metody zamkniętej rysunek 4.

Posiadamy drogę od 1 - > 64 pola.

ostatnie miejsce prowadzi nas do pól takich jak

35,23,25,57,39,63,49,33,

1. 1 -> 35 oraz 64 -> 36
2. 1 -> 23 oraz 64 -> 24
3. 1 -> 25 oraz 64 -> 26
4. 1 -> 57 oraz 64 -> 58
5. 1 -> 39 oraz 64 -> 40
6. 1 -> 49 oraz 64 -> 50
7. 1 -> 33 oraz 64 -> 34

Nie sprawdzamy pola numer 63 ,ponieważ nie ma wpływu na wynik. Wykryte drogi odwracamy.

1. 36 -> 64 oraz 35 -> 1
2. 24 -> 64 oraz 23 -> 1
3. 26 -> 64 oraz 25 -> 1
4. 58 -> 64 oraz 57 -> 1
5. 40 -> 64 oraz 39 -> 1
6. 50 -> 64 oraz 49 -> 1
7. 34 -> 64 oraz 33 -> 1

Teraz musimy zobaczyć gdzie możemy się dostać z pierwszego pola.

Pole 1 prowadzi nas do miejsc 2 i 12

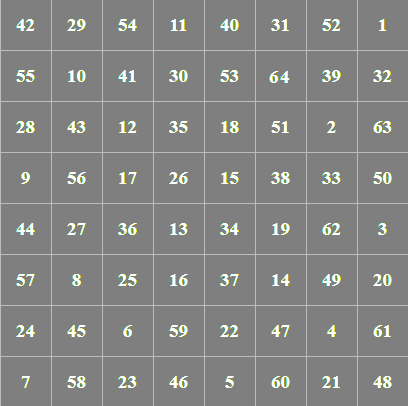
W wyniku dostajemy takie transformacje

1. 36 -> 64 oraz 31 -> 12 oraz 1 -> 11
2. 24 -> 64 oraz 23 -> 12 oraz 1 -> 11
3. 26 -> 64 oraz 25 -> 12 oraz 1 -> 11
4. 58 -> 64 oraz 57 -> 12 oraz 1 -> 11
5. 40 -> 64 oraz 39 -> 12 oraz 1 -> 11
6. 50 -> 64 oraz 49 -> 12 oraz 1 -> 11
7. 34 -> 64 oraz 33 -> 12 oraz 1 -> 11

Możemy zauważyć ,że w 6 trasie pole 50 łączy się z polem 11 możemy więc jedyne co zostało do zrobienia to przekształcenie tej transformacji na:

1 -> 11 oraz 50 -> 64 oraz 49 -> 12

Z każdej trasy ,którą sprawdzaliśmy można osiągnąć wynik różnica polega na ilości transformacji. Wynik tej metody przedstawiony jest na rysunku 5. Możemy teraz zauważyć ,że jesteśmy wstanie z ostatniego pola przejść bezpośrednio na początek.[9]



**Rysunek kompletna trasa rozwiązanie zamknięte**

Metoda wykorzystująca podział planszy..

(opis…………)

TO DO

## 3.4 Wykorzystanie w Kryptografii

(To DO)

## 3.5 Algorytmy rozwiązujące problem skoczka szachowego

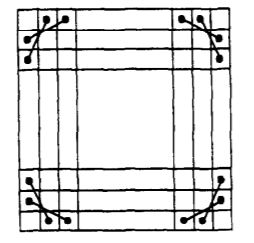
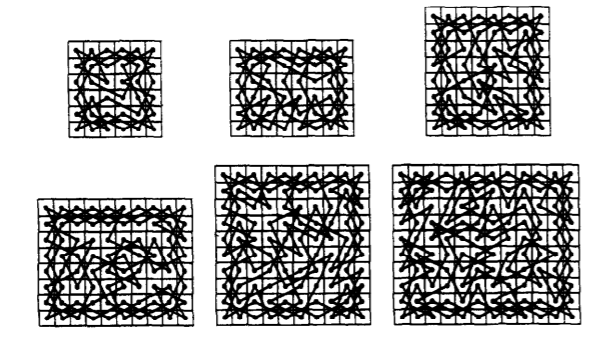
### 3.5.1 Brute-force

Metoda “siłowa”, która nie przyjmuje skomplikowanego algorytmu do znalezienia ścieżki. Kolejne ruchy są kontynuowane aż do momentu kiedy zabraknie legalnego ruchu. Następnie przy kolejnej próbie iteracji testowana jest kolejna ścieżka aż do wyczerpania wszystkich możliwości. Jest to najmniej efektywny algorytm, szczególnie dla dużych rozmiarów szachownicy, choć czasem wykorzystywany jeżeli inne algorytmy nie mogą być zastosowane. [10]

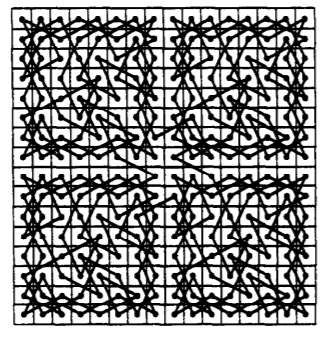
### 3.5.2 Divide and conquer

Jest to algorytm , który korzysta z pewnych obserwacji: istnieją powtarzalne fragmenty ścieżki na tablicy szachowej. Dla tablicy n x n, n x n + 2 gdzie n >= 6 i n parzystych istnieje ścieżka obliczona w czasie O(n^2).

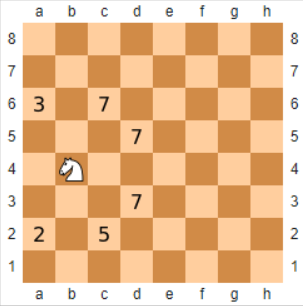
Kluczowy jest ten ruch skoczka w narożnikach, co przedstawia poniższy rysunek:



Przykładowo dla pola o wymiarach 16 x 16 problem można rozwiązać stosując technikę znajdowania ścieżki z pola 8 x 8.



### 3.5.3 Warnsdorff’s rule

Metoda ta polega na przesuwaniu skoczka na pole o najmniejszej ilości możliwych ruchów. 

Metoda ta jest skuteczna dla szachownicy 8x8 oraz dla innych niewielkich rozmiarów. W przypadku “remisów” wybierane jest losowe pole, co przy tych wymiarach nie stanowi problemu. Jednak dla wielkich tablic skuteczność tej metody spada, dlatego też doczekała się usprawnień. Padberry stosowal metode Warnsdorffa rowniez dla pozostalych podzbiorow na planszy co pozwoliło mu na uzyskanie dobrych wyników dla m = 78. Innym uprawnieniami były zmiany rozstrzygania remisów zgodnie z wybranym algorytmem (Pohl) co zwieksza skutecznosc oryginalnego algorytmu.

### 3.5.4 Neural network solution

Wszystkie ruchu skoczka przedstawiona sa jako neurony - o stanie 1 aktywne lub 0 nieaktywne. Aktywne neurony to te, ktore stanowia czesc rozwiązania. Przy uruchomieniu sieci neuronowej każdy neuron może zmienić swój stan na podstawie sąsiadujących neuronów zgodnie ze wzorem: [ To DO ]

Metoda ta traci skuteczność wraz ze wzrostem rozmiaru, korzystne wyniki otrzymuje się dla m okolo 25. Ponadto po zakończeniu algorytmu moze okazac sie ze wynik nie jest zamkniętym rozwiązaniem problemu skoczka.

### 3.5.5 Analiza algorytmów - wybór najbardziej optymalnego.

Metoda brute force może być przyjęta wyłącznie dla plansz o niewielkich rozmiarach, inaczej staje się mało efektywna. Prawdopodobieństwo szybkiego znalezienia rozwiązania maleje wraz ze wzrostem rozmiarów pola.

Metoda divide and conquer to prosta a jednocześnie skuteczna metoda (czasowo linearna) dla określonych rozmiarów szachownicy.

Ograniczeniem tego algorytmu jest wymiar tablicy - skuteczny jest on tylko dla określonych wymiarów (w przybliżeniu kwadratowych). Korzysta ona z .

To Do

# 4. Przedstawienie wykorzystanych technologii

## 4.1 Język C++

Język C++ powstał na początku lat osiemdziesiątych w oddziale badawczym Bell Labs przez Bjarne Stroustrupa. Pierwsza jego wersja pojawiła się w 1979 r. jako rozszerzenie języka C o obiektowe mechanizmy abstrakcji danych i silną statyczną kontrolę typów. Został stworzony aby ułatwić pisanie programów zachowując przy tym zgodność z językiem C. Jest jednym z najbardziej popularnych języków na świecie gdyż został zaprojektowany do ogólnego przeznaczenia między innymi gier komputerowych, mikrokontrolerów, przeróżnych aplikacji komercyjnych oraz do programowania systemowego czyli wykorzystując bezpośrednio zasoby sprzętowe jak na przykład sterowniki urządzeń, maszyny wirtualne, systemy operacyjne, środowiska programistyczne oraz biblioteki bazowe. Pozwala on na implementację programów przy użyciu stylów programowania proceduralnego, obiektowego ogólnego i abstrakcje danych. Ogólną właściwością tego języka jest wieloparadygmatowość oznacza to ,że można w nim stosować jednocześnie różne style programowania jak również programować na poziomie asemblera. Umożliwia bezpośrednie zarządzanie wolną pamięcią , zakłada statyczną kontrolę typów oraz posiada elementy dynamicznej kontroli typów. Programy napisane w tym języku mogą być wieloplatformowe bez żadnych modyfikacji ,ale tylko wtedy gdy przestrzegane są odpowiednie zasady , ponieważ niektóre biblioteki są dla specyficznych systemów. Głównie ten język znajduje zastosowanie tam ,gdzie liczy się niskie zużycie zasobów oraz wydajność. [11][12]

## 4.2 Język C#

Język C# powstał w latach 1998 - 2001 jego głównym twórcą był architekt Andersa Hejlsberg’a oraz jego zespół podczas prac nad środowiskiem Microsoft .NET Framework. Jest to nowoczesny obiektowy język programowania do ogólnego przeznaczenia ,którego zaletą jest możliwość szybkiego pisania aplikacji, gdyż twórcy tego języka zachowali balans między prostotą a wydajnością. Posiada wiele wyróżniających się cech obiektowości od innych języków programowania takich jak jednolity system typów, interfejs, własności, metody i zdarzenia. Dysponuje on statyczną kontrolą typów ,która polega na sprawdzaniu wykorzystanych typów danych podczas kompilacji programu, dzięki temu pozwala to na wyeliminowanie błędów przed rozpoczęciem programu. Ważna zaletą tego języku jest zarządzanie pamięcia ,ponieważ jest ona automatycznie zarządzana przez system wykonawczy. Środowisko uruchomieniowe CLR zawiera moduł usuwania nieużywanych obiektów. Jest on wykonywany w trakcie działania programu w celu odzyskania zajmowanej pamięci ,dzięki czemu programista nie musi własnoręcznie zwalniać pamięci jak jest w języku C++. Programowanie w języku C# pozwala na tworzenie aplikacji nie tylko na system Windows ,ale też na różne platformy urządzeń przenośnych oraz tworzyć aplikacje internetowe , które mogą działać na platformie Microsoft Net. Framework lub Microsoft .Net Core. [13]

## 4.3 Środowisko programistyczne

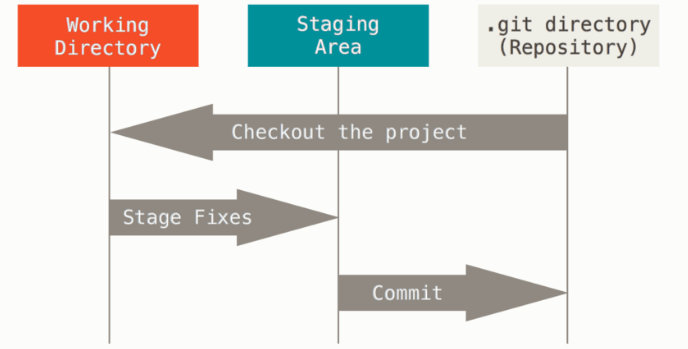
Do stworzenia aplikacji zostało użyte środowisko programistyczne Microsoft Visual Studio 2017 wersja Community. Jest to wersja bezpłatna, która umożliwia tworzenie oprogramowania konsolowego oraz z graficznym interfejsem użytkownika.

Wybrałem to narzędzie, ponieważ pozwoliło mi to na implementacje algorytmu w języku C++ oraz stworzenie graficznego interfejsu użytkownika w języku C#, dodatkowo posiada mechanizmy refaktoryzacji kodu, zintegrowany debugger co usprawnia znajdowanie błędów w pisanym programie, designer ,który pozwala na łatwe i szybkie tworzenie graficznego interfejsu użytkownika gdyż ma dostęp do elementów graficznych Microsoft Windows oraz pisania testów jednostkowych dzięki, którym programista ma możliwość sprawdzenia poprawności napisanego kodu. Środowisko to jest bardzo popularne co również ułatwia pracę, ponieważ w sieci możemy znaleźć bardzo duże ilości kursów oraz pomocy od innych użytkowników w przypadku ustawień projektu. [14]

## 4.4 System kontroli wersji

Do stworzenia aplikacji zostało użyte darmowe narzędzie służące do kontroli wersji oprogramowania Git. Używane jest często w komercyjnych projektach przez programistów ,którzy pracują nad wspólnym kodem źródłowym. Sprawdza się również bardzo dobrze dla pojedynczego użytkownika gdyż umożliwia śledzenie wprowadzanych zmian oraz pozwala w jakimkolwiek momencie na ich przywrócenie do wcześniejszej wersji projektu. Git posiada trzy stany ,w których mogą się znajdować pliki:

* Katalog Git - Jest to najważniejsza część projektu, ponieważ znajduje się tam baza danych projektu, posiadająca wszystkie jego wersje. Ten katalog jest kopiowany podczas klonowania repozytorium.
* Katalog roboczy - W tej sekcji są dokonywane wszelkie zmiany kodu, które możemy zobaczyć lub przywrócić do stanu odpowiadającemu ostatniej rewizji.
* Przechowalnia - Tutaj zostają wrzucone pliki z katalogu roboczego ,które chcemy zapamiętać w repozytorium. Dzięki niej można utrwalić tylko wybrane zmiany.



Rysunek Katalog git, Katalog roboczy, Przechowalnia

Źródlo: https://git-scm.com/book/pl/v2/Pierwsze-kroki-Podstawy-Git

Git umożliwia pracę nad tym samym kodem źródłowym przy użyciu rozgałęzień. Jest to powszechnie stosowana metoda ,w której zachodzi sytuacja gdy wielu programistów pracuje na tym samym pliku źródłowym. Różnice między wersjami zostają wtedy rozwiązane przez wykorzystanie narzędzia do scalania kodu. W wypadku kiedy pracuje tylko jeden użytkownik to narzędzie jest również bardzo przydatne, ponieważ zabezpieczy przed niepożądaną utratą danych oraz pozwoli na przywrócenie dowolnej wersji. [15]

## 4.5 Visual Paradigm

Jest to potężne narzędzie wieloplatformowe, które jest łatwe w użyciu do projektowania i zarządzania systemami informatycznymi. Pozwala na tworzenie wszelkiego rodzaju diagramów uml. Ułatwia współpracę z większością wiodących IDE w tym Microsoft Visual Studio. Pozwala na modelowanie procesów biznesowych oraz szybkie tworzenie dokumentacji i specyfikacji systemu. [16]

## 4.6 Matlab

Jest to język wysokiego poziomu o wysokiej wydajności do obliczeń technicznych. Zawiera on instrukcje przepływu sterowania, funkcje, struktury danych , dane wejściowe/wyjściowe oraz umożliwia pisanie programów zorientowanych obiektowo. Jest to również środowisko programistyczne przeznaczone do numerycznych obliczeń oraz wizualizacji danych. [17]

# 5. Projekt Aplikacji

## 5.1 Wprowadzenie

## 5.2 Interfejs graficzny użytkownika

## 5.3 Model wymagań

### 5.3.1 Wymagania funkcjonalne

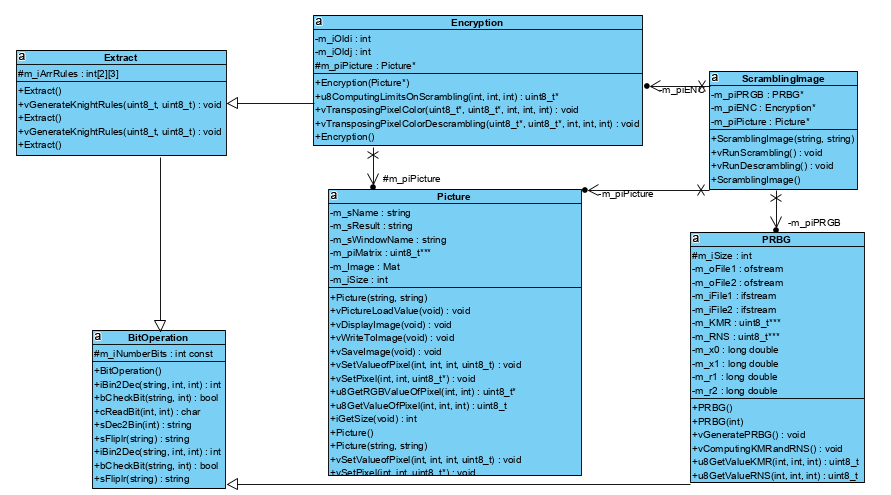
### 5.3.1.1 Diagram przypadków użycia

### 5.3.1.2 Diagram sekwencji

### 5.3.1.3 Diagram aktywności

### 5.3.2 Model projektowy

### 5.3.2.1 Diagram Klas



### 5.3.2.2 Diagram ERD

## 5.4 Testy

# 6. Opis Implementacji

## 6.1 Wstęp

Algorytm do szyfrowania obrazów został zaimplementowany w języku programowania C++ wykorzystując paradygmat programowania obiektowego. Zostały użyte standardowe biblioteki tego języka oraz dodatkowo biblioteka OpenCV, w której zawarte są zestawy funkcji umożliwiające pracę do obróbki obrazów. Natomiast aplikacja oraz interfejs użytkownika został zaimplementowany w języku C#, która wykonuje algorytm poprzez dołączenie pliku DLL z implementacją algorytmu.

## 6.2 Opis klas

## 6.3 Szczegółowy opis metod

# 7. Analiza

PODSUMOWANIE

(TO DO)

BIBLIOGRAFIA

[1] Adrian-Viorel Diaconu, Alexandru Costea and Marius-Aurel Costea. “Color Image Scrambling Technique Based on Transposition of Pixels between RGB Channels Using Knight’s Moving Rules and Digital Chaotic Map” (Published 23 April 2014)

[2] A.-C. Dăscălescu, R. E. Boriga, and A.-V. Diaconu, “Study of a new chaotic dynamical system and its usage in a novel pseudorandom bit generator,” Mathematical Problems in Engineering, vol. 2013, Article ID 769108, 10 pages, 2013.

[3] A.-V. Diaconu, “Multiple bitstreams generation using chaotic sequences,” The Annals of the University Dunarea De Jos of Galati Fascicle III, vol. 35, no. 1, pp. 37–42, 2012

[4] Graf Hamiltonowski. https://pl.wikipedia.org/wiki/Graf\_hamiltonowski

[5] Ben Hill, Kevin Tostado “ Knight’s Tours”, December 18, 2004

[6] Schwenk, Allen J. ”Which Rectangular Chessboards Have a Knight’s

Tour?”. Mathematics. 64 (1991): 325-332.

[7] Joe DeMaio, Bindia Mathew “Which Chessboards have a Closed Knight’s Tour within the Rectangular Prism?”.

[8] G.L.Chia, Siew-HuiOng “Generalized knight's tours on rectangular chessboards“ Discrete Applied Mathematics, Volume 150, Issues 1–3, 1 September 2005, Pages 80-98'

[9] L. Euler, ”A Solution to a curious problem which seems not subjected to any analysis”. Acad. Sci. Berlin 15 (1759), 310–337.

[10] Problem skoczka szachowego, https://en.wikipedia.org/wiki/Knight%27s\_tour

[11] Stephen Prata, “Szkoła Programowania Język C++ Wydanie V”

[12] Bjarne Stroustrup, “Język C++ Kompendium Wiedzy Wydanie IV”

[13] Joseph Albahari, Ben Albahari, “C# 6.0 w pigułce Wydanie VI”

[14] Microsoft Visual Studio IDE, https://docs.microsoft.com/pl-pl/visualstudio/ide/visual-studio-ide?view=vs-2017

[15] Włodzimierz Gajda, “Git Rozproszony system kontroli wersji”, Helion, Gliwice 2013

[16] Visual Paradigm, https://www.visual-paradigm.com/support/documents/vpuserguide/12/13/5963\_visualparadi.html

[17] Matlab, https://www.mathworks.com/products/matlab.html

SPIS ILUSTRACJI

Rysunek 1. Ruchy skoczka szachowego

Rysunek 2. Generowanie reguł z tablicy KMR

Rysunek 3. Generowanie reguł z tablic RNS

Rysunek 4. Kierunek ruchu skoczka

Rysunek 5. Zasady transpozycji pikseli

Rysunek 6. Graf Hamiltonowski

Rysunek 7 Dla tablic 4 x n nie istnieje rozwiązanie zamknięte