Uniwersytet Kardynała Stefana
Wyszyńskiego
Wydział Matematyczno – Przyrodniczy

Gra Policjanci i Złodzieje

Piotr Jaworski

Warszawa, 2020

Spis treści

1. Wstęp

Poniżej przedstawiam dokumentację projektu gry „Policjanci i Złodzieje”. Miniona gra jest grą planszową, turową i komputerową. Plansza ma kształt kwadratu i jest złożona z 400 małych kafelków, białe, reprezentują drogę, po której można się poruszać, a brązowe budynki, które stanowią przeszkodę uniemożliwiającą przejście przez to pole. Na planszy znajdują się dwa rodzaje pionków, policjanci reprezentowani przez głowy z kapeluszami policyjnymi lub detektywistycznymi i złodzieje reprezentowani przez głowy z czarnymi czapkami. W każdej rozgrywce jest zawsze dwóch policjantów(Policjant1 i Policjant2) oraz jeden złodziej. Celem policjantów jest złapanie złodzieja, a złodziej ma za zadanie jak najdłużej uciekać przed policją. Dostępne są dwa tryby gry:

1. Police Team – gracz porusza się dwoma policjantami, a komputer dokonuje ruchów złodzieja
2. Thief Team – gracz porusza się złodziejem, komputer policjantami

W trakcie gry ruchy pionków wykonywane są na zmianę w tej samej kolejności przez całą rozgrywkę, a każdy ruch może zostać wykonany w czterech kierunkach góra, dół, lewo, prawo i tylko na kafelek drogi graniczący z kafelkiem obecnie zajmowanym przez dokonujący ruch pionek.

Program napisany jest w języku C# [1] z wykorzystaniem bibliotek Windowsa [2] w środowisku Microsoft Visual Studio 2013 [4] zainstalowanego na platformie Windows 10 [5].

2. Podstawy teoretyczne

2.1 Opis planszy gry

Plansza ma kształt kwadratu i zbudowana jest z 400 jednakowych pól o kształcie czworokątów foremnych. Pola występują w dwóch kolorach białym oznaczającym drogę i brązowym oznaczającym budynki. Pionki mogą się poruszać tylko po polach oznaczających drogę. Ruch polega na przesunięciu się pionka na jedno z pustych (nie zajmowanych przez żaden inny pionek), białych pól bezpośrednio graniczących z jego obecnym polem w linii pionowej lub poziomej (nie dozwolone są ruchy po skosie).

2.2 Gra policjantami

W trybie gry policjantami gracz porusza się dwoma pionkami policjant1 (policeman1) i policjant2 (policeman2), a jego celem jest doprowadzenie policjantów do złodzieja i zablokowaniu mu drogi do dalszej ucieczki. Na samym początku pionki policjantów i złodzieja ustawiane są na pozycjach startowych. Pierwszy dokonuje ruchu policjant1, a zatem gracz i następnie po nim policjant2 czyli również gracz. Po wykonaniu ruchu gracza obydwoma jego pionkami, następuje ruch złodzieja wykonywany przez komputer i dokonywany jest w taki sposób aby pionek złodzieja oddalił się od bliższego policjanta. Powyższa kolejność wykonywania ruchów jest powielana przez cała rozgrywkę.

2.3 Gra złodziejem

W trybie gry złodziejem gracz porusza się jednym pionkiem o nazwie złodziej (thief), a jego celem jest ucieczka przed goniącymi go policjantami przez 150 ruchów (1 ruch to przesunięcie się dowolnego pionka o jedną pozycję zgodnie z zasadami gry). Również jak w przypadku trybu gry policjantami na samym początku pionki policjantów i złodzieja ustawiane są na pozycjach startowych. Pierwszy dokonuje ruchu złodziej, a zatem gracz, a następnie po nim komputer porusza się policjantem1 i potem policjantem2. Powyższa kolejność wykonywania ruchów jest powielana przez całą rozgrywkę. Policjanci dokonują takich ruchów, które ich najbardziej zbliżą do złodzieja, jednocześnie unikając sytuacji, kiedy obaj policjanci poruszaliby się jedna drogą.

2.4 Opis algorytmów

2.4.1 Algorytm inicjalizacji planszy

Dane wejściowe:

- Wymiary planszy – static 20 x 20
- Tryb gry – 0 lub 1
- Wygląd pionków
- Ciąg liczbowy zapisany w pliku tekstowym odwzorowujący kształt planszy z cyframi określającymi przeznaczenie pola (0 – droga, 1 – budynek, 2 – policjant1, 3 – złodziej, 4 – policjant2

Wynik działania algorytmu:

- Dwuwymiarowa tablica typu int zawierająca informacje o rozplanowaniu budynków i dróg, a także o pozycjach wszystkich pionków (ustawionych na pozycjach startowych)
- Dwuwymiarowa, pomocnicza tablica typu int zawierająca informacje o rozplanowaniu budynków i dróg
- Dwuwymiarowa tablica typu PictureBox wyświetlająca obrazy odwzorowujące faktyczny stan planszy
- Obraz reprezentujący wygląd pionka rozpoczynającego rozgrywkę oraz jego nazwę

Lista kroków:

(1) Start algorytmu
(2) Ustaw obraz reprezentujący wygląd pionka rozpoczynającego rozgrywkę oraz jego nazwę w zależności od wybranego przez użytkownika trybu gry i wyglądu pionków
(3) Wczytaj jedną cyfrę z ciągu liczbowego zapisanego w pliku tekstowym
   a) Jeśli wczytana liczba = 0
      (a.1) Wstaw do dwuwymiarowej tablicy typu int 0
      (a.2) Wstaw do dwuwymiarowej, pomocniczej tablicy typu int 0
      (a.3) Wstaw do dwuwymiarowej tablicy typu PictureBox obraz drogi
   b) Jeśli wczytana liczba = 1
      (a.1) Wstaw do dwuwymiarowej tablicy typu int 1
      (a.2) Wstaw do dwuwymiarowej, pomocniczej tablicy typu int 1
      (a.3) Wstaw do dwuwymiarowej tablicy typu PictureBox obraz budynku
   c) Jeśli wczytana liczba = 2
      (a.1) Wstaw do dwuwymiarowej tablicy typu int 2
      (a.2) Wstaw do dwuwymiarowej, pomocniczej tablicy typu int 0
      (a.3) Wstaw do dwuwymiarowej tablicy typu PictureBox obraz policjanta1 zależnego od wyboru wyglądu pionków dokonanego wcześniej przez użytkownika
   d) Jeśli wczytana liczba = 3
      (a.1) Wstaw do dwuwymiarowej tablicy typu int 3
      (a.2) Wstaw do dwuwymiarowej, pomocniczej tablicy typu int 0
      (a.3) Wstaw do dwuwymiarowej tablicy typu PictureBox obraz złodzieja zależnego od wyboru wyglądu pionków dokonanego wcześniej przez użytkownika
   e) Jeśli wczytana liczba = 4
      (a.1) Wstaw do dwuwymiarowej tablicy typu int 4
      (a.2) Wstaw do dwuwymiarowej, pomocniczej tablicy typu int 0
      (a.3) Wstaw do dwuwymiarowej tablicy typu PictureBox obraz policjanta2 zależnego od wyboru wyglądu pionków dokonanego wcześniej przez użytkownika
(4) Ustaw lokalizacje obrazów dodanych do tablicy typu PictureBox wyliczonych na podstawie rozmiaru planszy i obrazów oraz kolejności dodawanego obrazu
(5) Wróć do punktu (3) dopóki nie sprawdzisz wszystkich cyfr z pliku tekstowego
(6) Koniec algorytmu

2.4.2 Algorytm dokonywania ruchu złodzieja, w trybie gry policjantami

Algorytm odpowiedzialny za znajdywanie najlepszego ruchu dla złodzieja wykorzystuje algorytm BFS (breadth-first search), który polega na przeszukiwaniu grafu (w tym przypadku ścieżek stworzonych z pól drogi na planszy) z punktu startowego we wszystkich kierunkach jednocześnie z tą sama prędkością w każdą stronę do momentu znalezienia szukanego wierzchołka grafu. W przypadku ruchu złodzieja szukanym wierzchołkiem grafu jest pozycja bliższego policjanta. W przypadku znalezienia szukanego wierzchołka złodziej dokonuje ruchu w przeciwnym kierunku niż kierunek najbardziej przybliżający go do znalezionego celu w celu zwiększenia dystansu od bliższego policjanta. Zanim jednak algorytm będzie mógł to zrobić, musi najpierw określić, który policjant znajduje się bliżej złodzieja. Z tego powodu algorytm dokonywania ruchu złodzieja, w trybie gry policjantami podzielony jest na dwa mniejsze algorytmy.

1. Pierwszy z nich odpowiedzialny jest tylko za ustalenie, który z policjantów znajduje się bliżej złodzieja i działa w następujący sposób:

    (1) Start algorytmu
    (2) Ustawienie pomocniczej, dwuwymiarowej tablicy typu int do stanu początkowego (tablica przechowuje tylko informacje o tym czy dane pole jest 0 - drogą czy 1 - budynkiem)
    (3) Stworzenie kolejki (zmiennej typu Queue) przechowującej zmienne typu pojedynczych pól planszy
    (4) Umieszczenie na początku kolejki pola, na którym obecnie znajduje się złodziej
    (5) Zaznaczenie pola, na którym znajduje się złodziej w pomocniczej, dwuwymiarowej tablicy typu int za pomocą wartości -1
    (6) Sprawdzenie czy najstarsza dodana zmienna do kolejki jest polem zajmowanym przez, któregoś z policjantów
        a) Jeśli (6) prawdziwe, to jeśli policjantem zajmującym badane pole jest policjant1
            i. Ustawienie wartości pola w pomocniczej, dwuwymiarowej tablicy typu int przechowującego informacje o pozycji złodzieja (wartość (-1)) na 0 – droga
            ii. Wywołanie drugiej części algorytmu z policjantem1 jako polem szukanym
            iii. Zwrócenie informacji uzyskanej dzięki wywołaniu drugiej części algorytmu o tym, gdzie ma się ruszyć złodziej do głównego kodu programu, koniec algorytmu
        b) Jeśli (6) prawdziwe, to jeśli policjantem zajmującym badane pole jest policjant2
            i. Ustawienie wartości pola w pomocniczej, dwuwymiarowej tablicy typu int przechowującego informacje o pozycji złodzieja (wartość (-1)) na 0 – droga
            ii. Wywołanie drugiej części algorytmu z policcjantem2 jako polem szukanym
            iii. Zwrócenie informacji uzyskanej dzięki wywołaniu drugiej części algorytmu o tym, gdzie ma się ruszyć złodziej do głównego kodu programu

(7) Usunięcie sprawdzanego w punkcie (6) elementu z kolejki
(8) Sprawdzenie czy pola otaczające usuwany w punkcie (7) element nie wykraczają poza rozmiar tablicy przechowującej pola planszy oraz czy są drogą
    a) Jeśli sprawdzane pole spełnia oba warunki z punktu (8)
        i. Ustawienie wartości pola w pomocniczej, dwuwymiarowej tablicy typu int o współrzędnych badanego pola na 2
        ii. Dodanie badanego pola do kolejki
(9) Jeśli kolejka nie jest pusta wróć do instrukcji (6)
(10) Koniec algorytmu

2. Działanie drugiego algorytmu (drugiej części algorytmu ruchu złodzieja):

Dane wejściowe:
- Pozycja złodzieja
- Pozycja bliższego policjanta

Wynik działania algorytmu:
- Kierunek w jakim ma się ruszyć złodziej

(1) Start algorytmu
(2) Ustawienie pomocniczej, dwuwymiarowej tablicy typu int do stanu początkowego (tablica przechowuje tylko informacje o tym czy dane pole jest 0 - drogą czy 1 - budynkiem)
(3) Stworzenie kolejki (zmiennej typu Queue) przechowującej zmienne typu pojedynczych pól planszy
(4) Umieszczenie na początku kolejki pola, na którym obecnie znajduje się złodziej
(5) Zaznaczenie pola, na którym znajduje się złodziej w pomocniczej, dwuwymiarowej tablicy typu int za pomocą wartości -1
(6) Stworzenie zmiennych pomocniczych
(7) Sprawdzenie czy najstarsza dodana zmienna do kolejki jest polem zajmowanym przez bliższego policjanta
    a) Jeśli warunek z punktu (7) jest prawdziwy
        i. Dodanie do zmiennej pomocniczej (tablica przechowująca pola graniczące z polem zajmowanym przez złodzieja, po których można dostać się do bliższego policjanta, pola są przechowywane w kolejności od rozpoczynającego najkrótszą drogę do celu do rozpoczynającego najdłuższą) informacji o polu graniczącym z polem zajmowanym przez złodzieja, które musi wybrać złodziej, aby dotrzeć do bliższego policjanta
        ii. Zwiększenie o jeden zmiennej pomocniczej przechowującej informację o ilości pól graniczących z polem zajmowanym przez złodzieja rozpoczynających drogę do bliższego policjanta
        iii. Jeśli ilość pól graniczących z polem złodzieja pozwalających dotrzeć do bliższego policjanta jest równa ilości pól oznaczających drogę graniczących z polem, na którym znajduje się złodziej zwróć pole na które ma się ruszyć złodziej i zakończ algorytm

iv. Jeśli warunek z punktu (iii) jest fałszywy wyczyść kolejkę, ustaw pomocniczą, dwuwymiarową tablicę typu int do stanu początkowego, dodaj do kolejki pozycje złodzieja

b) Jeśli warunek z punktu (7) jest fałszywy
   i. Usuń najstarszy element z kolejki

(8) Sprawdzenie czy otaczające usunięty element pola nie wykraczają poza tablicę planszy i są drogą
   a) Jeśli warunek z punktu (8) jest prawdziwy i badany kierunek nie znajduje się w zmiennej pomocniczej przechowującej informacje czy sprawdzany kierunek pozwala dotrzeć do bliższego policjanta, algorytm sprawdza w którym kierunku względem pola, na którym znajduje się złodziej znajduje się badane pole i zaznacza ten kierunek w dwuwymiarowej, pomocniczej tablicy typu int w miejscu o współrzędnych badanego pola, badany element dodawany jest do kolejki

(9) Jeśli kolejka nie jest pusta wróć do punktu (7)

(10) Koniec algorytmu

2.4.3 Algorytm dokonywania ruchu policjantów, w trybie gry złodziejem

Algorytm wykonywania ruchu policjantami, w trybie gry złodziejem również tak jak w przypadku algorytmu dokonywania ruchu złodziejem wykorzystuje algorytm BFS (breadth-first search), co oznacza, że graf (w tym przypadku ścieżki stworzone z pól drogi na planszy) jest przeszukiwany we wszystkich kierunkach jednocześnie z tą sama prędkością od miejsca startowego (w tym przypadku pozycji policjanta wykonującego ruch) do momentu znalezienia szukanego wierzchołka grafu (w tym przypadku pozycji złodzieja). Zanim jednak algorytm zaczyna szukać najlepszego ruchu dla policjanta najpierw musi upewnić się, że obaj policjanci nie stoją w jednej ulicy jednocześnie. Gdyby tak się stało, że obaj policjanci zajmują pozycje w tej samej uliczce, a pomiędzy nimi nie znajduje się złodziej, a ruch najbardziej przybliżający policjanta wykonującego ruch byłby ruchem przybliżającym go również do drugiego policjanta. W takiej sytuacji policjant wykonujący ruch musi wycofać się z zajmowanej przez drugiego policjanta drogi w inną ulice. Z tego powodu algorytm dokonywania ruchu policjantów, w trybie gry złodziejem został podzielony na dwa mniejsze algorytmy.

1. Działanie algorytmu sprawdzającego czy dwóch policjantów stoi w jednej ulicy:

Dane wejściowe:

- Pozycja policjanta wykonującego ruch
- Pozycja drugiego policjanta

Wynik działania algorytmu:

- Zaznaczenie specjalnych ścian blokujących wykonanie ruchu w stronę drugiego policjanta w pomocniczej, dwuwymiarowej tablicy typu int wartością 9, w przypadku gdy dwóch policjantów stoi w jednej ulicy

(1) Start algorytmu

(2) Ustawienie zmiennych pomocniczych typu bool odpowiedzialnych za przechowywanie informacji o tym czy pomiędzy dwoma policjantami znajduje się ściana lub złodziej na wartość false (czyli na start zakładamy, że między dwoma policjantami nie ma ścian ani złodzieja)

(3) Sprawdzenie czy obaj policjanci znajdują się w jednym rzędzie

a) Sprawdzenie, czy między policjantami znajduje się złodziej lub pole budynku, jeśli tak zaznaczenie tych informacji w zmiennych pomocniczych typy bool

(4) Sprawdzenie czy obaj policjanci znajdują się w jednej kolumnie
   a) Sprawdzenie, czy między policjantami znajduje się złodziej lub pole budynku, jeśli tak zaznaczenie tych informacji w zmiennych pomocniczych typy bool

(5) Jeśli warunek z punktu (3) lub (4) został spełniony i pomiędzy policjantami znajduje się pole budynku lub złodziej
   a) Zaznaczenie w pomocniczej, dwuwymiarowej tablicy typu int ściany specjalnej ustawionej w miejscu pomiędzy dwoma policjantami na polu graniczącym z polem, na którym znajduje się policjant wykonujący ruch wartością 9

(6) Koniec algorytmu

2. Działanie algorytmu znajdującego najlepszy ruch dla policjanta:

Dane wejściowe:

- Pozycja policjanta wykonującego ruch
- Pozycja złodzieja

Wynik działania algorytmu:

- Kierunek w jakim ma się ruszyć policjant

(1) Start algorytmu

(2) Ustawienie pomocniczej, dwuwymiarowej tablicy typu int do stanu początkowego, ale z zachowaniem specjalnych ścian powstałych w wyniku działania algorytmu sprawdzającego czy dwóch policjantów stoi w jednej uliczce (tablica przechowuje tylko informacje o tym czy dane pole jest 0 - drogą czy 1 – budynkiem lub 9 – ściana specjalna)

(3) Stworzenie kolejki (zmiennej typu Queue) przechowującej zmienne typu pojedynczych pól planszy

(4) Umieszczenie na początku kolejki pola, na którym obecnie znajduje się policjant wykonujący ruch

(5) Zaznaczenie pola, na którym znajduje się policjant obecnie wykonujący ruch w pomocniczej, dwuwymiarowej tablicy typu int za pomocą wartości -1

(6) Stworzenie zmiennych pomocniczych

(7) Sprawdzenie czy najstarsza dodana zmienna do kolejki jest polem zajmowanym przez złodzieja
   a) Jeśli warunek z punktu (7) jest prawdziwy
      i. Dodanie do zmiennej pomocniczej (tablica przechowująca pola graniczące z polem zajmowanym przez policjanta wykonującego ruch, po których można dostać się do pola zajmowanego przez złodzieja, pola są przechowywane w kolejności od rozpoczynającego najkrótszą drogę do celu do rozpoczynającego najdłuższą) informacji o polu graniczącym z polem zajmowanym przez policjanta wykonującego ruch, które musi wybrać, aby dotrzeć do złodzieja

ii. Zwiększenie o jeden zmiennej pomocniczej przechowującej informację o ilości pól graniczących z polem zajmowanym przez policjanta wykonującego ruch rozpoczynających drogę do bliższego policjanta
iii. Jeśli ilość pól graniczących z polem policjanta wykonującego ruch pozwalających dotrzeć do złodzieja jest równa ilości pól oznaczających drogę graniczących z polem, na którym znajduje się policjant wykonujący ruch zwróć pole na które ma się ruszyć policjant i zakończ algorytm
iv. Jeśli warunek z punktu (iii) jest fałszywy wyczyść kolejkę, ustaw pomocniczą, dwuwymiarową tablicę typu int do stanu początkowego z zachowaniem pozycji specjalnych ścian, dodaj do kolejki pozycje policjanta wykonującego ruch

b) Jeśli warunek z punktu (7) jest fałszywy
i. Usuń najstarszy element z kolejki

(8) Sprawdzenie czy otaczające usunięty element pola nie wykraczają poza tablicę planszy i są drogą
a) Jeśli warunek z punktu (8) jest prawdziwy algorytm sprawdza w którym kierunku względem pola, na którym znajduje się policjant wykonujący ruch znajduje się badane pole i zaznacza ten kierunek w dwuwymiarowej, pomocniczej tablicy typu int w miejscu o współrzędnych badanego pola, badany element dodawany jest do kolejki

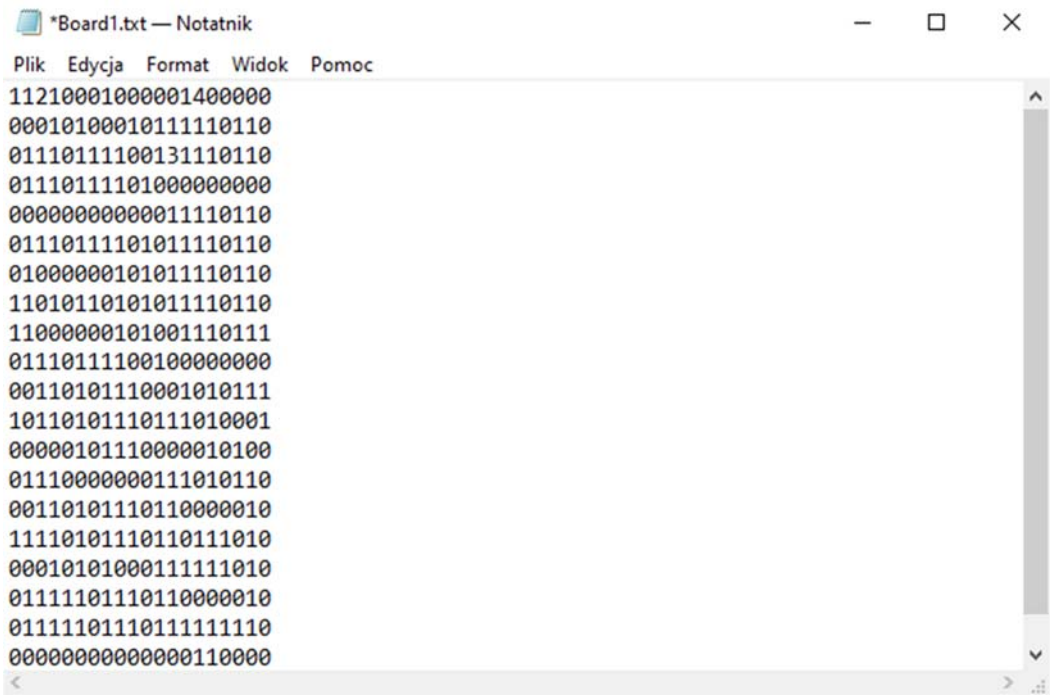
(9) Jeśli kolejka nie jest pusta wróć do punktu (7)
(10) Koniec algorytmu

3. Opis programu

3.1 Struktura programu

3.1.1 Plik wejściowy

Plikiem wejściowym mojego projektu jest plik tekstowym zawierający ciąg cyfr opisujących wygląd planszy do gry oraz pozycje startowe pionków. Cyfry 0 oznaczają pola drogi, cyfry 1 pola budynków, cyfra 2 pozycję policjanta1, 3 złodzieja i 4 policjanta2


*Board1.txt — Notatnik
Plik Edycja Format Widok Pomoc
1121000010000014000000
0001010001011111110110
0111011111001311101110
0111011110100000000000
0000000000001111011O
0111011110101111011O
0100000010101111011O
1101011010101111011O
1100000010100111011l
0111011110010000000O
0011010111000101011l
1011010111011101000l
0000010111000001010O
0111000000111010110
0011010111011000001O
1111010111011011101O
0001010100011111101O
0111110111011000001O
0111110111011111111O
0000000000000011000O

3.1.2 Klasa Cell

Klasa, która przechowuje informacje o pojedynczym polu planszy. Nie posiada żadnych metod tylko cztery pola i jeden konstruktor:

- `public int RowNumber { get; set; }` – numer wiersza komórki
- `public int ColumnNumber { get; set; }` – numer kolumny komórki
- `public bool IsBulding { get; set; }` – informacja czy komórka jest budynkiem czy drogą
- `public int CurrentState { get; set; }` – informacja czy jakiś pionek zajmuje tą komórkę
- `public Cell(int x, int y){RowNumber = x; ColumnNumber = y;}` – konstruktor dwuargumentowy ustalający położenie komórki

3.1.3 Klasa Board

Klasa, która reprezentuje plansze do gry. Posiada dwa pola i jeden konstruktor:

- `public int Size { get; set; }` – rozmiar planszy
- `public Cell[,] Table { get; set; }` – dwuwymiarowa tablica przechowująca pola typu Cell reprezentująca plansze do gry
- Konstruktor jednoargumentowy tworzący nową plansze

```
17      // constructor
18      public Board (int s)
19      {
20          // initial size of the board
21          Size = s;
22
23          // create a new 2D arry of type cell
24          Table = new Cell[Size, Size];
25
26          // fill the 2D arry with new Cells with unique x and y coordinates
27          for (int i = 0; i < Size; i++)
28          {
29              for (int j = 0; j < Size; j++)
30              {
31                  Table[i, j] = new Cell(i, j);
32              }
33          }
34      }
```

3.1.4 Klasa Character

Klasa, która reprezentuje pionki policjantów i złodziei. Posiada cztery pola, jeden konstruktor i jedną metodę:

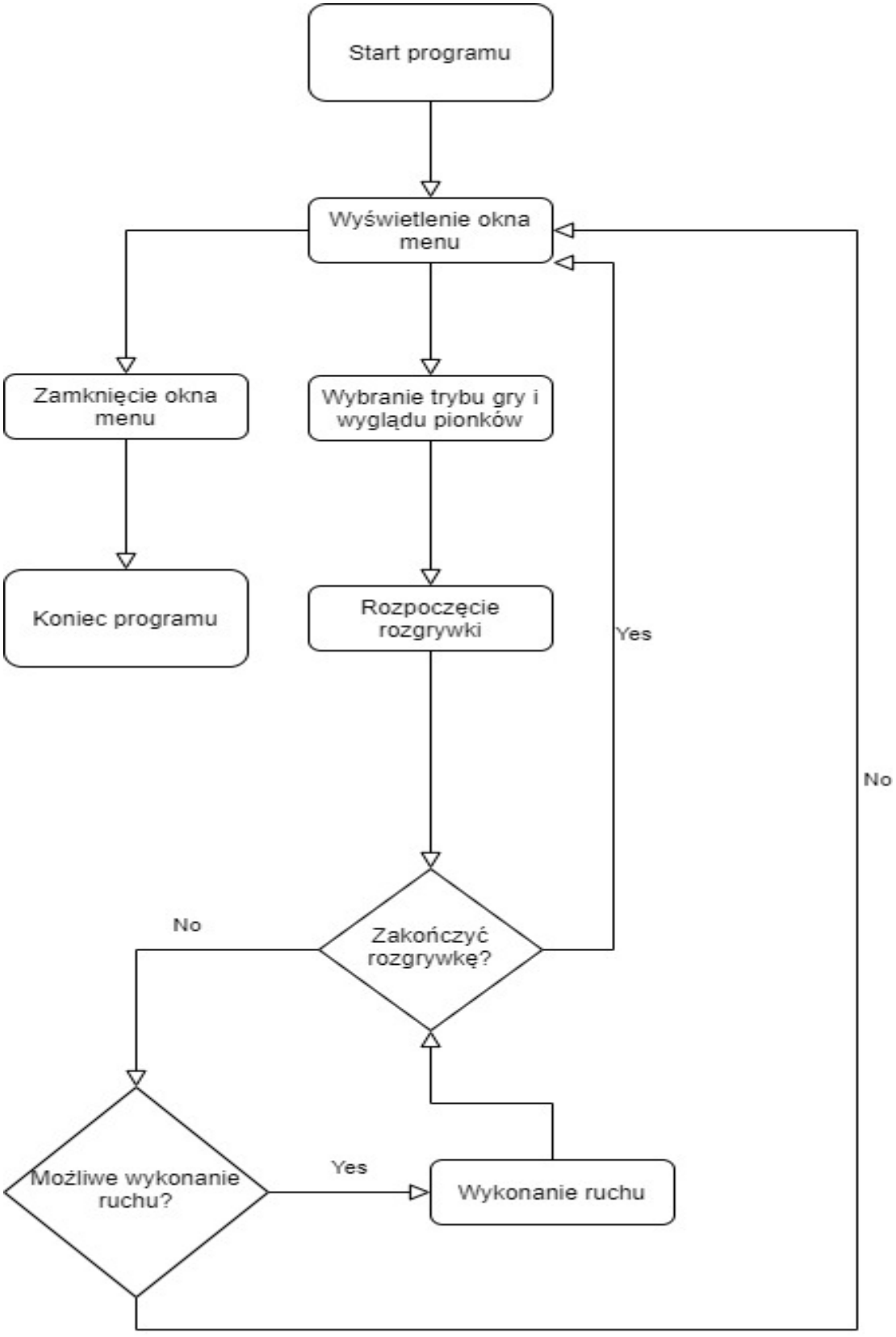
- `public int Team { get; set; }` – pole informujące o tym czy pionek jest policjantem czy złodziejem
- `public int CharacterNumber { get; set; }` – pole informujące o wyglądzie pionka
- `public int PositionX { get; set; }` – numer kolumny, w której znajduje się pionek

- `public int PositionY { get; set; }` – numer wiersza, w którym znajduje się pionek
- `public Character(int t, int a, int x, int y, int p){ Team = t; CharacterNumber = a; PositionX = x; PositionY = y;}` – konstruktor pięcioargumentowy tworzący pionek
- Metoda Move zmieniająca położenie pionka na planszy w kierunku zależnym od wartości zmiennej podanej w argumencie wywołania funkcji, jeśli ruch jest udany metoda zwraca true, jeśli ruch jest niemożliwy do wykonania metoda zwraca false

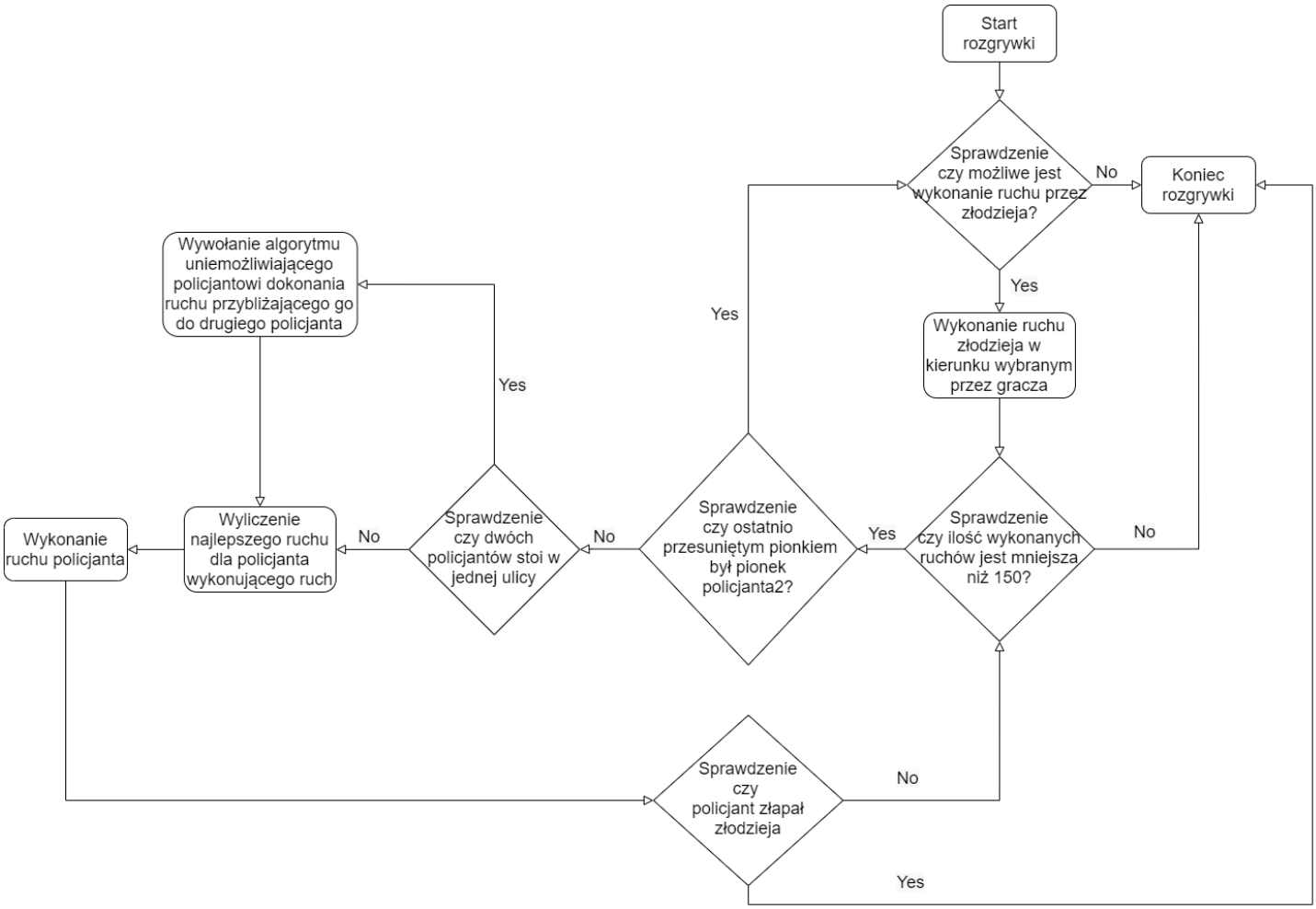
```
// method that changing position of the character
public bool Move(int direction, Board myBoard)
{
    // move up
    if (direction == 1)
    {
        if (PositionX != (myBoard.Size - myBoard.Size))
        {
            if (myBoard.Table[PositionX - 1, PositionY].IsBulding == false)
            {
                myBoard.Table[PositionX - 1, PositionY].CurrentState = CharacterNumber;
                myBoard.Table[PositionX, PositionY].CurrentState = 0;
                PositionX = PositionX - 1;
                return true;
            }
        }
    }

    // move down
    else if (direction == 2)
    {
        if (PositionX != (myBoard.Size - 1))
        {
            if (myBoard.Table[PositionX + 1, PositionY].IsBulding == false)
            {
                myBoard.Table[PositionX + 1, PositionY].CurrentState = CharacterNumber;
                myBoard.Table[PositionX, PositionY].CurrentState = 0;
                PositionX = PositionX + 1;
                return true;
            }
        }
    }

    // move left
    else if (direction == 3)
    {
        if (PositionY != (myBoard.Size - myBoard.Size))
        {
            if (myBoard.Table[PositionX, PositionY - 1].IsBulding == false)
            {
                myBoard.Table[PositionX, PositionY - 1].CurrentState = CharacterNumber;
                myBoard.Table[PositionX, PositionY].CurrentState = 0;
                PositionY = PositionY - 1;
                return true;
            }
        }
    }

    // move right
    else if (direction == 4)
    {
        if (PositionY != (myBoard.Size - 1))
        {
            if (myBoard.Table[PositionX, PositionY + 1].IsBulding == false)
            {
                myBoard.Table[PositionX, PositionY + 1].CurrentState = CharacterNumber;
                myBoard.Table[PositionX, PositionY].CurrentState = 0;
                PositionY = PositionY + 1;
                return true;
            }
        }
    }
    return false;
}
```

3.2 Schematy blokowe aplikacji

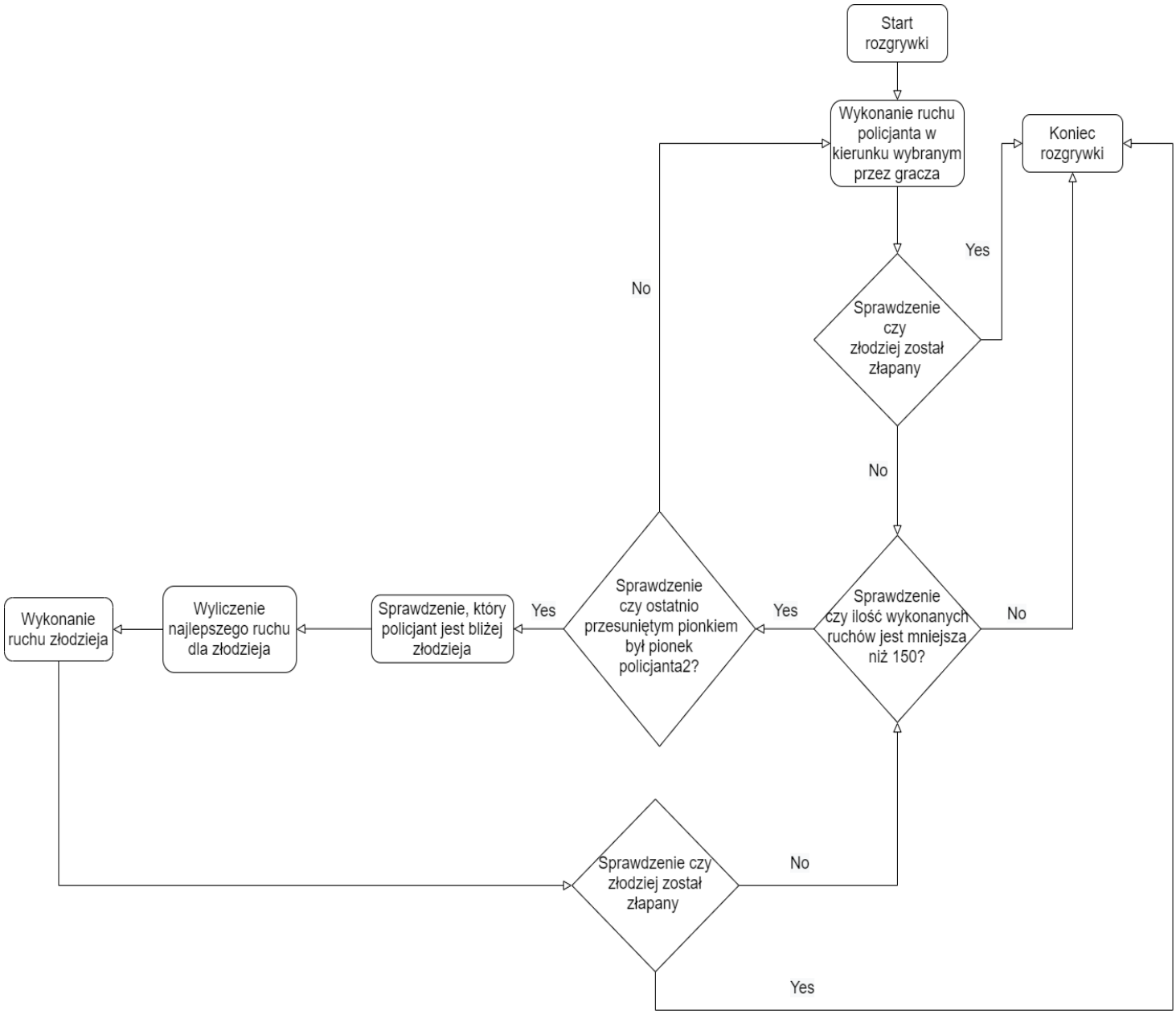
3.2.1 Schemat blokowy ogólnego działania programu



3.2.2 Schemat blokowy trybu gry złodziejem

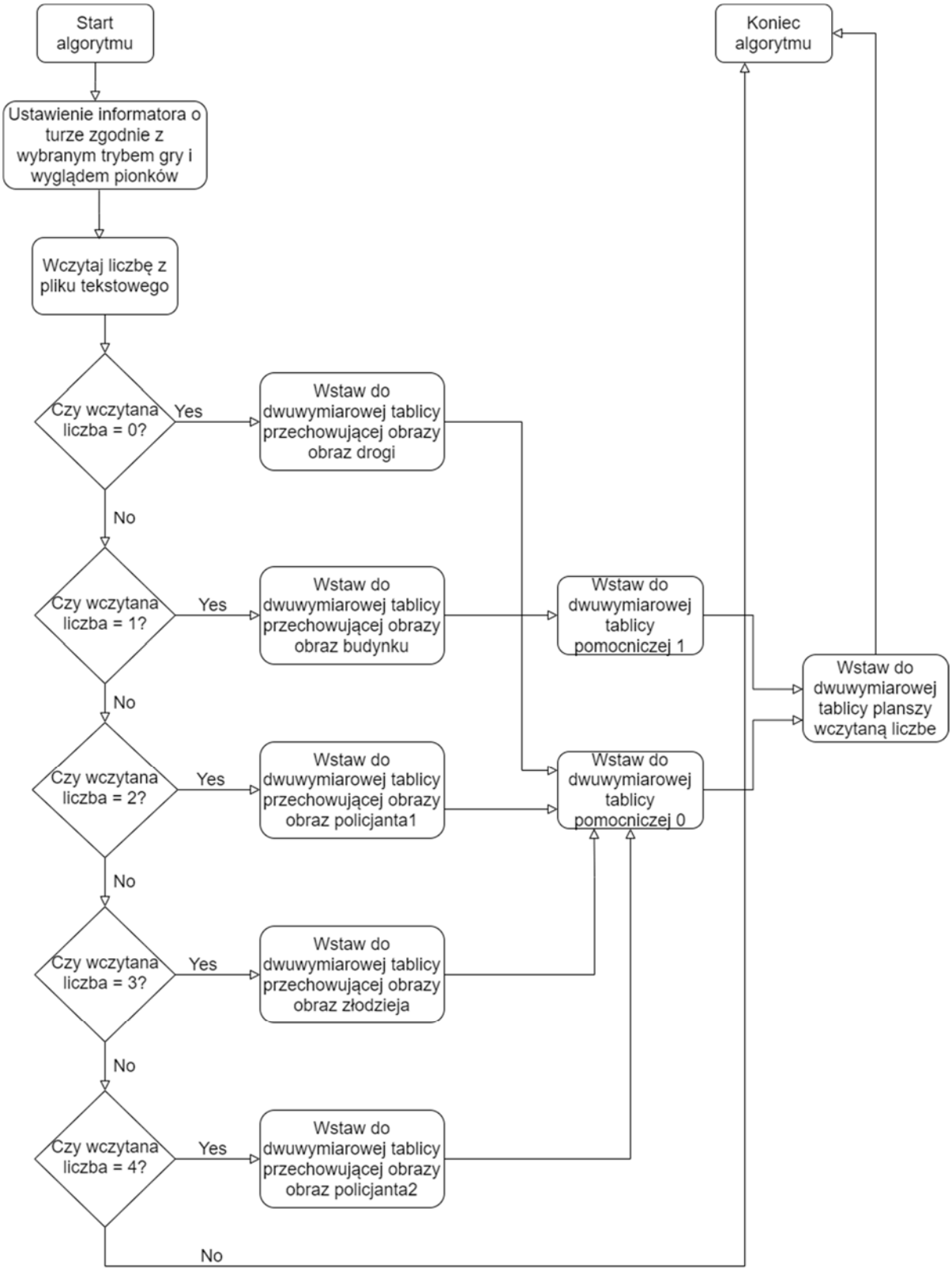


3.2.3 Schemat blokowy trybu gry policjantami



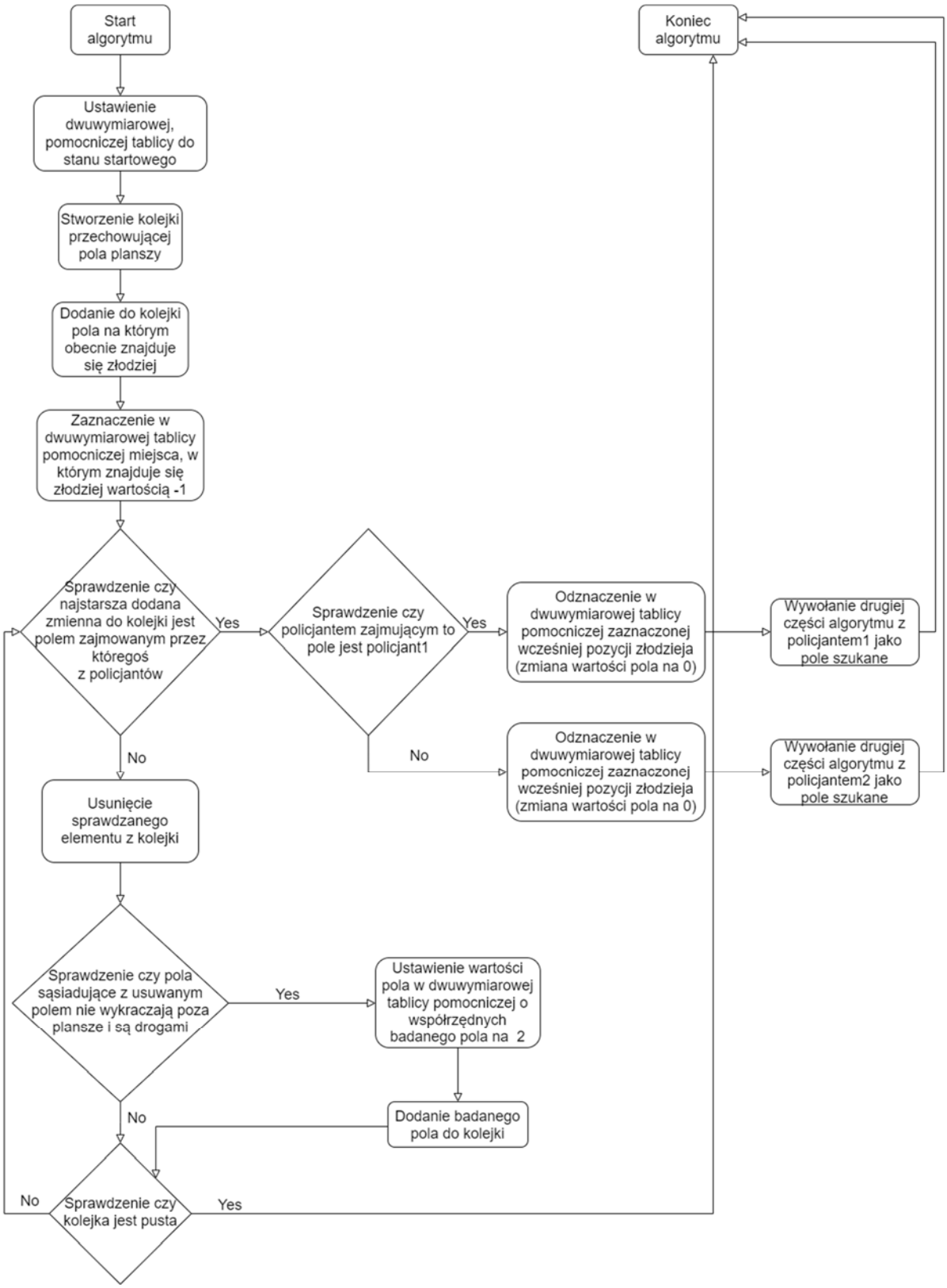
3.3 Schematy blokowe algorytmów

3.3.1 Schemat blokowy algorytmu inicjalizacji planszy

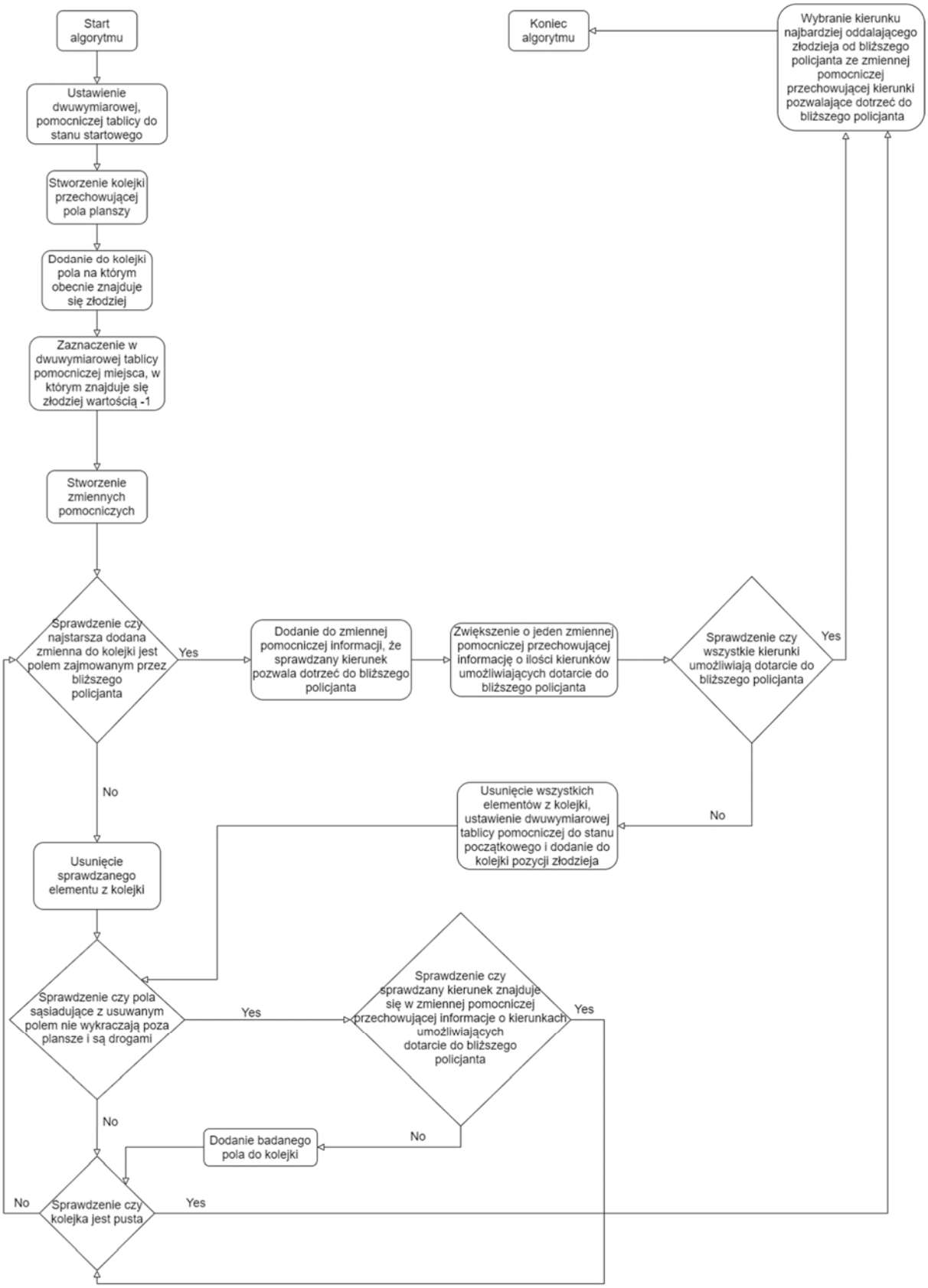


3.3.2 Schemat blokowy algorytmu dokonywania ruchu złodzieja, w trybie gry policjantami

1. Pierwsza część algorytmu

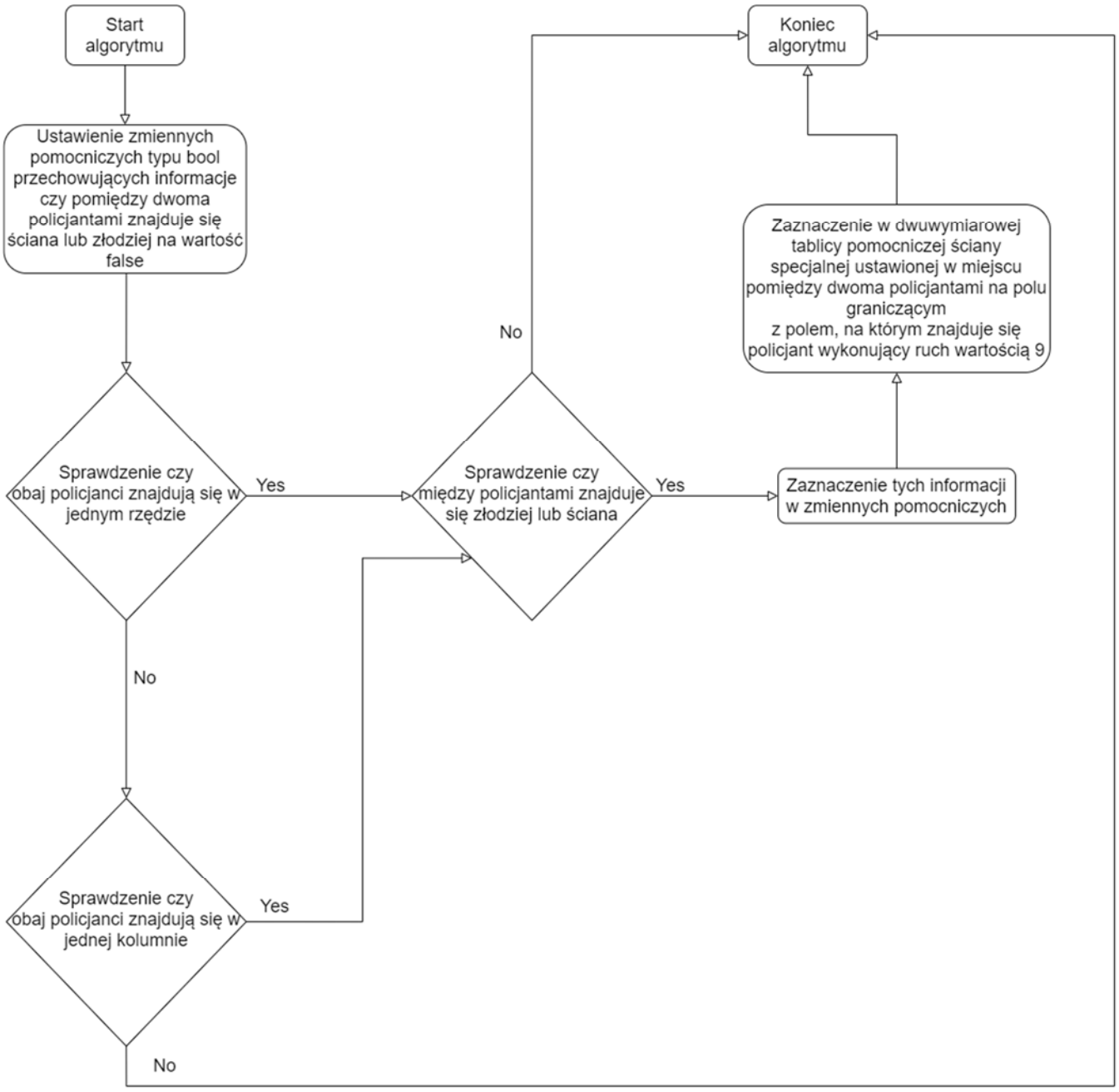


2. Druga część algorytmu

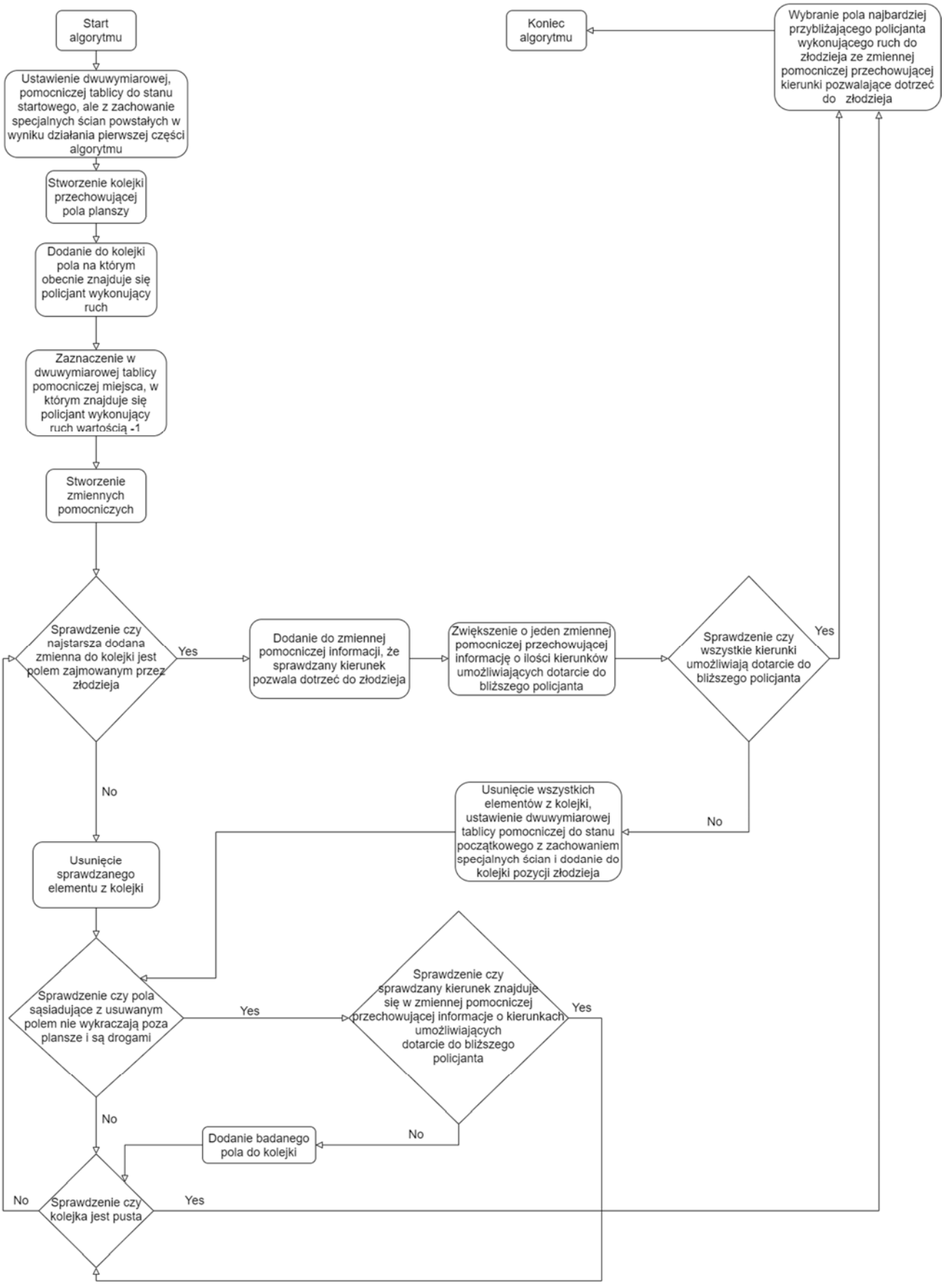


3.3.3 Schemat blokowy algorytmu dokonywania ruchu policjantów, w trybie gry złodziejem

1. Pierwsza część algorytmu ruchu policjantów (Sprawdzanie czy dwóch policjantów znajduje się w jednej uliczce)



2. Druga część algorytmu ruchu policjantów (znajdowanie najlepszego ruchu dla policjanta)



3.4 Kod źródłowy wybranych elementów programu

3.4.1 Klasa Cell

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PiZBoardModel
{
    public class Cell
    {
        // the number of row
        public int RowNumber { get; set; }

        // the number of column
        public int ColumnNumber { get; set; }

        // is the cell a building or a road
        public bool IsBulding { get; set; }

        // is there any charakter staying on the cell
        public int CurrentState { get; set; }

        public Cell(int x, int y)
        {
            RowNumber = x;
            ColumnNumber = y;
        }
    }
}
```

Objaśnienia

Klasa Cell reprezentuje pojedynczą komórkę tablicy dwuwymiarowej pełniącej funkcji planszy, na której toczy się rozgrywka. Obiekty tej klasy są pojedynczymi polami mapy, posiadają swoje współrzędne i informacje o tym czy są drogą czy budyniem oraz czy są aktualnie zajmowane przez jakiś pionek.

3.4.2 Klasa Board

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace PiZBoardModel
8   {
9       public class Board
10      {
11          // the size of the board usually 20x20
12          public int Size { get; set; }
13
14          // 2D array of type cell
15          public Cell[,] Table { get; set; }
16
17          // constructor
18          public Board (int s)
19          {
20              // initial size of the board
21              Size = s;
22
23              // create a new 2D arry of type cell
24              Table = new Cell[Size, Size];
25
26              // fill the 2D arry with new Cells with unique x and y coordinates
27              for (int i = 0; i < Size; i++)
28              {
29                  for (int j = 0; j < Size; j++)
30                  {
31                      Table[i, j] = new Cell(i, j);
32                  }
33              }
34          }
35      }
36  }
```

Objaśnienia

Klasa Board reprezentuje plansze złożoną z pojedynczych pól typu Cell, na której prowadzona jest cała rozgrywka. Tworzona za pomocą tej klasy tablica dwuwymiarowa ma zawsze rozmiar 20 x 20 i zawiera informacje o aktualnych położeniach wszystkich pionków oraz rozplanowaniu położenia budynków i dróg.

3.4.3 Klasa Character

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


namespace PiZBoardModel
{
    public class Character
    {
        public int Team { get; set; }
        public int CharacterNumber { get; set; }
        public int PositionX { get; set; }
        public int PositionY { get; set; }
        public Character(int t, int a, int x, int y, int p)
        {
            Team = t;
            CharacterNumber = a;
            PositionX = x;
            PositionY = y;
        }

        // method that changing position of the character
        public bool Move(int direction, Board myBoard)
        {
            // move up
            if (direction == 1)
            {
                if (PositionX != (myBoard.Size - myBoard.Size))
                {
                    if (myBoard.Table[PositionX - 1, PositionY].IsBulding == false)
                    {
                        myBoard.Table[PositionX - 1, PositionY].CurrentState = CharacterNumber;
                        myBoard.Table[PositionX, PositionY].CurrentState = 0;
                        PositionX = PositionX - 1;
                        return true;
                    }
                }
            }

            // move down
            else if (direction == 2)
            {
                if (PositionX != (myBoard.Size - 1))
                {
                    if (myBoard.Table[PositionX + 1, PositionY].IsBulding == false)
                    {
                        myBoard.Table[PositionX + 1, PositionY].CurrentState = CharacterNumber;
                        myBoard.Table[PositionX, PositionY].CurrentState = 0;
                        PositionX = PositionX + 1;
                        return true;
                    }
                }
            }

            // move left
            else if (direction == 3)
            {
                if (PositionY != (myBoard.Size - myBoard.Size))
                {
                    if (myBoard.Table[PositionX, PositionY - 1].IsBulding == false)
                    {
                        myBoard.Table[PositionX, PositionY - 1].CurrentState = CharacterNumber;
                        myBoard.Table[PositionX, PositionY].CurrentState = 0;
                        PositionY = PositionY - 1;
                        return true;
                    }
                }
            }

            // move right
            else if (direction == 4)
            {
                if (PositionY != (myBoard.Size - 1))
                {
                    if (myBoard.Table[PositionX, PositionY + 1].IsBulding == false)
                    {
                        myBoard.Table[PositionX, PositionY + 1].CurrentState = CharacterNumber;
                        myBoard.Table[PositionX, PositionY].CurrentState = 0;
                        PositionY = PositionY + 1;
                        return true;
                    }
                }
            }
            return false;
        }
    }
}

```

Objaśnienia

Klasa Charakter reprezentuje pionek, którym gracze poruszają się po planszy w trakcie prowadzenia rozgrywki. Obiekty tej klasy posiadają pola definiujące czy są pionkiem policjanta czy złodzieja, wygląd pionka wybrany wcześniej przez użytkownika, współrzędne definiujące położenie pionka w aktualnym momencie na mapie oraz metodę Move umożliwiającą przesuwanie pionków po planszy w wybranym przez użytkownika lub algorytm wykonujący ruchy pionków kierunku.

3.4.4 Algorytm inicjalizacji planszy

```
55      private void fillImageBoard()
56      {
57          if(gameMode == 1)
58          {
59              label4.Text = "Thief";
60              if (thief.CharacterNumber == 1)
61              {
62                  pictureBox2.Image = Properties.Resources.Adam_złodziej;
63              }
64              else
65              {
66                  pictureBox2.Image = Properties.Resources.Jaca_złodziej;
67              }
68          }
69          else if(gameMode == 2)
70          {
71              label4.Text = "Policeman1";
72              if (policeman1.CharacterNumber == 1)
73              {
74                  pictureBox2.Image = Properties.Resources.Policjant_Filip;
75              }
76              else
77              {
78                  pictureBox2.Image = Properties.Resources.Policjant_Piotrek;
79              }
80          }
81          int pictureSize = panel1.Width / myBoard.Size;
82          panel1.Height = panel1.Width;
83          int counter = 0;
84          for (int i = 0; i < myBoard.Size; i++)
85          {
86              for (int j = 0; j < myBoard.Size; j++)
87              {
88                  imageBoard[i, j] = new PictureBox();
89                  imageBoard[i, j].Height = pictureSize;
90                  imageBoard[i, j].Width = pictureSize;
91                  if(Properties.Resources.Board1[counter] == '0')
92                  {
93                      imageBoard[i, j].Image = Properties.Resources.Droga_kwadrat_biały;
94                      myBoard.Table[i, j].IsBulding = false;
95                      myBoard.Table[i, j].CurrentState = 0;
96
97                      // ancillary board
98                      anciBoard.Table[i, j].CurrentState = 0;
99                      anciBoard.Table[i, j].IsBulding = false;
100                 }
101                 else if (Properties.Resources.Board1[counter] == '1')
102                 {
103                     imageBoard[i, j].Image = Properties.Resources.Budynek_kwadrat_ciemny;
104                     myBoard.Table[i, j].IsBulding = true;
105                     myBoard.Table[i, j].CurrentState = 1;
106
```

```
                    // ancillary board
                    anciBoard.Table[i, j].CurrentState = 1;
                    anciBoard.Table[i, j].IsBulding = true;
                }
                else if (Properties.Resources.Board1[counter] == '3')
                {
                    myBoard.Table[i, j].IsBulding = false;
                    myBoard.Table[i, j].CurrentState = 3;

                    // ancillary board
                    anciBoard.Table[i, j].CurrentState = 0;
                    anciBoard.Table[i, j].IsBulding = false;
                    if (thief.CharacterNumber == 1)
                        imageBoard[i, j].Image = Properties.Resources.Adam_złodziej_z_tłemb;
                    else
                        imageBoard[i, j].Image = Properties.Resources.Jaca_złodziej_z_tłemb;
                }
                else if (Properties.Resources.Board1[counter] == '2')
                {
                    myBoard.Table[i, j].IsBulding = false;
                    myBoard.Table[i, j].CurrentState = 2;

                    // ancillary board
                    anciBoard.Table[i, j].CurrentState = 0;
                    anciBoard.Table[i, j].IsBulding = false;
                    if (policeman1.CharacterNumber == 1)
                        imageBoard[i, j].Image = Properties.Resources.Policjant_Filip_z_tłemb;
                    else
                        imageBoard[i, j].Image = Properties.Resources.Policjant_Piotrek_z_tłemb_40x40;
                }
                else if (Properties.Resources.Board1[counter] == '4')
                {
                    myBoard.Table[i, j].IsBulding = false;
                    myBoard.Table[i, j].CurrentState = 4;

                    // ancillary board
                    anciBoard.Table[i, j].CurrentState = 0;
                    anciBoard.Table[i, j].IsBulding = false;
                    if (policeman2.CharacterNumber == 1)
                        imageBoard[i, j].Image = Properties.Resources.Policjant_Korczak_z_tłemb;
                    else
                        imageBoard[i, j].Image = Properties.Resources.Policjant_Karol_z_tłemb;
                }
                imageBoard[i, j].SizeMode = PictureBoxSizeMode.Zoom;
                panel1.Controls.Add(imageBoard[i, j]);
                imageBoard[i, j].Location = new Point(j * pictureSize, i * pictureSize);
                counter++;
            }
            counter += 2;
        }
    }
```

Objaśnienia

Algorytm inicjalizacji planszy najpierw w zależności od trybu gry i wyglądu pionków ustalonych przez użytkownika tworzy informator, który wskazuje, który pionek ma wykonać jako pierwszy ruch. Następnie ustala wymiary planszy i w zależności od wczytanych danych z pliku tekstowego tworzy pola dróg i budynków ustalając ich położenie, a następnie umieszcza na pozycjach startowych wszystkie pionki biorące udział w rozgrywce oraz zapełnia odpowiednimi danymi tablice dwuwymiarową pomocniczą.

3.4.5 Funkcja sprawdzająca koniec gry

```
596         //Function taht checks if thief or policemen win
597         private void WinChecker()
598         {
599             if((policeman1.PositionX == thief.PositionX && policeman1.PositionY == thief.PositionY) || (policeman2.PositionX == thief.PositionX && policeman2.PositionY == thief.PositionY))
600             {
601                 this.Close();
602                 Form3 VILAS = new Form3(0);
603                 VILAS.Show();
604             }
605             else if (tourCounter == 150)
606             {
607                 this.Close();
608                 Form3 VILAS = new Form3(1);
609                 VILAS.Show();
610             }
611         }
```

Objaśnienia

Funkcja sprawdza czy któryś z policjantów złapał złodzieja (czyli wszedł na pole przez niego zajmowane) i jeśli tak zamyka formularz rozgrywki i otwiera formularz z informacją o tym, że drużyna policjantów zwyciężyła. Jeśli żadnemu policjantowi nie udało się złapać złodzieja funkcja sprawdza czy ilość tur jest równa 150 (czyli czy złodziejowi udało się uciec przed policjantami) i jeśli tak jest zamyka formularz rozgrywki i otwiera formularz z informacją o tym, że drużyna złodzieja zwyciężyła.

3.4.6 Algorytm dokonywania ruchu złodzieja, w trybie gry policjantami

1. Pierwsza część algorytmu dokonującego ruchu złodzieja (sprawdzanie, który policjant jest bliżej złodzieja)

```
528         //function that's checking distance between thief and policeman1 and thief and policeman2 and decides wich one of policeman is closer to thief
529         //and then with help of BestMoveFinder find and return best move for thief
530
531         private int ThiefMoveFinder()
532         {
533             anciBoardCleaner();
534             int[] move = new int[2] { 0, 0 };
535             Queue<Cell> q = new Queue<Cell>();
536             Cell s = new Cell(thief.PositionX, thief.PositionY);
537             q.Enqueue(s);
538             anciBoard.Table[thief.PositionX, thief.PositionY].CurrentState = -1;
539             while(q.Count != 0)
540             {
541                 Cell curr = q.Peek();
542                 if(curr.RowNumber == policeman1.PositionX && curr.ColumnNumber == policeman1.PositionY)
543                 {
544                     anciBoard.Table[thief.PositionX, thief.PositionY].CurrentState = 0;
545                     move = BestMoveFinder(thief, policeman1);
546                     return move[1];
547                 }
548                 else if(curr.RowNumber == policeman2.PositionX && curr.ColumnNumber == policeman2.PositionY)
549                 {
550                     anciBoard.Table[thief.PositionX, thief.PositionY].CurrentState = 0;
551                     move = BestMoveFinder(thief, policeman2);
552                     return move[1];
553                 }
554                 q.Dequeue();
555                 for (int i = 0; i < 4; i++)
556                 {
557                     int row = new int();
558                     int col = new int();
559                     if (i == 0)
560                     {
561                         row = curr.RowNumber - 1;
562                         col = curr.ColumnNumber;
563                     }
564                     else if (i == 1)
565                     {
566                         row = curr.RowNumber;
567                         col = curr.ColumnNumber + 1;
568                     }
569                     else if (i == 2)
570                     {
571                         row = curr.RowNumber + 1;
572                         col = curr.ColumnNumber;
573                     }
574                     else if (i == 3)
575                     {
576                         row = curr.RowNumber;
577                         col = curr.ColumnNumber - 1;
578                     }
579                     if (isValid(row, col) && anciBoard.Table[row, col].CurrentState == 0)
```

```
                    if (isValid(row, col) && anciBoard.Table[row, col].CurrentState == 0)
                    {
                        anciBoard.Table[row, col].CurrentState = 2;
                        Cell adjcell = new Cell(row, col);
                        q.Enqueue(adjcell);
                    }
                }
            }
            return move[1];
        }

```

Objaśnienia

Powyższy algorytm przeszukuje we wszystkich kierunkach z tą samą prędkością plansze gry zaczynając od pola zajmowanego przez złodzieja do momentu aż znajdzie pierwszą ścieżkę łączącą złodzieja z jednym z policjantów i następnie wywołuje drugą część algorytmu dokonującego ruchu złodzieja przekazując do niego informacje o tym, do którego policjanta została znaleziona ścieżka jako pierwsza.

2. Druga część algorytmu dokonującego ruchu złodzieja (znalezienie najlepszego ruchu dla złodzieja)

```
    /* function that's finding the best move for policeman and thief by using BFS algorithm and returns array that stores in first cell best move for policeman
    and in second best momve for thief */
    private int[] BestMoveFinder(Character start, Character destination)
    {
        anciBoardCleaner();
        int[] bestAndWorstMove = new int[2];
        Queue<Cell> q = new Queue<Cell>();
        Cell s = new Cell(start.PositionX, start.PositionY);
        q.Enqueue(s);
        anciBoard.Table[start.PositionX, start.PositionY].CurrentState = -1;
        int toFirstadjcell = 0;
        int[] checkedCell = new int[4] { 0, 0, 0, 0 };
        int temp = 0;
        int reachedDest = 0;
        while(q.Count != 0)
        {
            Cell curr = q.Peek();
            if (curr.RowNumber == destination.PositionX && curr.ColumnNumber == destination.PositionY)
            {
                checkedCell[reachedDest] = anciBoard.Table[curr.RowNumber, curr.ColumnNumber].CurrentState;
                reachedDest++;
                if (reachedDest == toFirstadjcell)
                {
                    bestAndWorstMove[0] = checkedCell[0];
                    bestAndWorstMove[1] = checkedCell[reachedDest - 1];
                    anciBoard.Table[start.PositionX, start.PositionY].CurrentState = 0;
                    return bestAndWorstMove;
                }
                else
                {
                    q.Clear();
                    anciBoardCleaner();
                    q.Enqueue(s);
                    curr = q.Peek();
                }
            }
            q.Dequeue();
            for(int i = 0; i < 4; i++)
            {
                int row = new int();
                int col = new int();
                if (i == 0)
                {
                    row = curr.RowNumber - 1;
                    col = curr.ColumnNumber;
                }
                else if(i == 1)
                {
                    row = curr.RowNumber;
                    col = curr.ColumnNumber + 1;
                }
                else if (i == 2)
```

```
                {
                    row = curr.RowNumber + 1;
                    col = curr.ColumnNumber;
                }
                else if (i == 3)
                {
                    row = curr.RowNumber;
                    col = curr.ColumnNumber - 1;
                }
                if (isValid(row, col) && anciBoard.Table[row, col].CurrentState == 0)
                {
                    if(anciBoard.Table[curr.RowNumber, curr.ColumnNumber].CurrentState == -1)
                    {
                        if(i == 0)
                        {
                            if(reachedDest > 0)
                            {
                                if(!hasReached(2, checkedCell))
                                {
                                    anciBoard.Table[row, col].CurrentState = 2;
                                }
                                else
                                {
                                    continue;
                                }
                            }
                            else
                            {
                                anciBoard.Table[row, col].CurrentState = 2;
                                if(temp == 0)
                                   toFirstadjcell++;
                            }
                        }
                        else if(i == 1)
                        {
                            if (reachedDest > 0)
                            {
                                if (!hasReached(3, checkedCell))
                                {
                                    anciBoard.Table[row, col].CurrentState = 3;
                                }
                                else
                                {
                                    continue;
                                }
                            }
                            else
                            {
                                anciBoard.Table[row, col].CurrentState = 3;
                                if (temp == 0)
                                    toFirstadjcell++;
                            }
```

```
                }
                else if (i == 2)
                {
                    if (reachedDest > 0)
                    {
                        if (!hasReached(4, checkedCell))
                        {
                            anciBoard.Table[row, col].CurrentState = 4;
                        }
                        else
                        {
                            continue;
                        }
                    }
                    else
                    {
                        anciBoard.Table[row, col].CurrentState = 4;
                        if (temp == 0)
                            toFirstadjcell++;
                    }
                }
                else if (i == 3)
                {
                    if (reachedDest > 0)
                    {
                        if (!hasReached(5, checkedCell))
                        {
                            anciBoard.Table[row, col].CurrentState = 5;
                        }
                        else
                        {
                            continue;
                        }
                    }
                    else
                    {
                        anciBoard.Table[row, col].CurrentState = 5;
                        if (temp == 0)
                            toFirstadjcell++;
                    }
                }
            }
            else if(anciBoard.Table[curr.RowNumber, curr.ColumnNumber].CurrentState == 2)
            {
                anciBoard.Table[row, col].CurrentState = 2;
            }
            else if (anciBoard.Table[curr.RowNumber, curr.ColumnNumber].CurrentState == 3)
            {
                anciBoard.Table[row, col].CurrentState = 3;
            }
            else if (anciBoard.Table[curr.RowNumber, curr.ColumnNumber].CurrentState == 4)
            {
                anciBoard.Table[row, col].CurrentState = 4;
            }
            else if (anciBoard.Table[curr.RowNumber, curr.ColumnNumber].CurrentState == 5)
            {
                anciBoard.Table[row, col].CurrentState = 5;
            }
            Cell adjcell = new Cell(row, col);
            q.Enqueue(adjcell);
        }
    }
    temp++;
}
anciBoardCleaner();
int temp2 = 0;
Stack<int> stack = new Stack<int>();
Random rng = new Random();
bestAndWorstMove[0] = checkedCell[0];
if (reachedDest > 0)
{
    if(reachedDest > 1)
    {
        bestAndWorstMove[1] = checkedCell[reachedDest - 1];
    }
    else if(toFirstadjcell > 1)
    {
        if (isValid(start.PositionX - 1, start.PositionY) && anciBoard.Table[start.PositionX - 1, start.PositionY].CurrentState == 0 && checkedCell[0] != 2 && checkedCell[1] != 2 && checkedCell[2] != 2 && checkedCell[3] != 2)
        {
            stack.Push(2);
            temp2++;
        }
        if (isValid(start.PositionX, start.PositionY + 1) && anciBoard.Table[start.PositionX, start.PositionY + 1].CurrentState == 0 && checkedCell[0] != 3 && checkedCell[1] != 3 && checkedCell[2] != 3 && checkedCell[3] != 3)
        {
            stack.Push(3);
            temp2++;
        }
        if (isValid(start.PositionX + 1, start.PositionY) && anciBoard.Table[start.PositionX + 1, start.PositionY].CurrentState == 0 && checkedCell[0] != 4 && checkedCell[1] != 4 && checkedCell[2] != 4 && checkedCell[3] != 4)
        {
            stack.Push(4);
            temp2++;
        }
        if (isValid(start.PositionX, start.PositionY - 1) && anciBoard.Table[start.PositionX, start.PositionY - 1].CurrentState == 0 && checkedCell[0] != 5 && checkedCell[1] != 5 && checkedCell[2] != 5 && checkedCell[3] != 5)
        {
            stack.Push(5);
            temp2++;
        }
        temp2 = rng.Next(0, temp2);
        for (int i = 0; i < temp2; i++)
        {
            stack.Pop();
        }
        bestAndWorstMove[1] = stack.Pop();
    }
```

```
399             else
400             {
401                 bestAndWorstMove[1] = checkedCell[0];
402             }
403         }
404         else
405         {
406             if (isValid(start.PositionX - 1, start.PositionY) && anciBoard.Table[start.PositionX - 1, start.PositionY].CurrentState == 0)
407             {
408                 stack.Push(2);
409                 temp2++;
410             }
411             if (isValid(start.PositionX, start.PositionY + 1) && anciBoard.Table[start.PositionX, start.PositionY + 1].CurrentState == 0)
412             {
413                 stack.Push(3);
414                 temp2++;
415             }
416             if (isValid(start.PositionX + 1, start.PositionY) && anciBoard.Table[start.PositionX + 1, start.PositionY].CurrentState == 0)
417             {
418                 stack.Push(4);
419                 temp2++;
420             }
421             if (isValid(start.PositionX, start.PositionY - 1) && anciBoard.Table[start.PositionX, start.PositionY - 1].CurrentState == 0)
422             {
423                 stack.Push(5);
424                 temp2++;
425             }
426             temp2 = rng.Next(0, temp2);
427             for(int i = 0; i < temp2; i++)
428             {
429                 stack.Pop();
430             }
431             bestAndWorstMove[0] = stack.Pop();
432             bestAndWorstMove[1] = bestAndWorstMove[0];
433         }
434         anciBoard.Table[start.PositionX, start.PositionY].CurrentState = 0;
435         return bestAndWorstMove;
436     }
```

Objaśnienia

Powyższy algorytm wykorzystywany jest podczas wykonywania ruchu przez złodzieja, w trybie gry policjantami oraz przez policjantów, w trybie gry złodziejem. W obu przypadkach algorytm stanowi drugą część całego algorytmu zajmującego się dokonywaniem ruchów przez komputer. Powyższy algorytm w przypadku obliczania najlepszego ruchu dla złodzieja przyjmuje jako miejsce startowe pozycje złodzieja, a jako miejsce docelowe pozycje bliższego policjanta znalezionego przez pierwszą część algorytmu dokonywania ruchu złodzieja. Swoje działanie zaczyna od dodania do kolejki pozycji złodzieja, sprawdzenia czy policjant znajduje się w tym samym miejscu i jeśli nie, sprawdzeniu sąsiadujących pól z dopiero co sprawdzonym polem w taki sam sposób zaznaczając, w którym kierunku przeszukiwana jest ścieżka w danym momencie. Jeśli, któraś ścieżka pozwoli doprowadzić złodzieja do policjanta zapisywany jest kierunek jaki trzeba obrać, aby podążać tą ścieżką, po czym cały proces przeszukiwania mapy zostaje powielony z tą różnicą, że znaleziony wcześniej kierunek nie jest sprawdzany ponownie. Algorytm przeszukuje mapę do momentu aż znajdzie tyle ścieżek doprowadzających złodzieja do policjanta ile jest pól drogi graniczących z polem złodzieja lub do momentu opróżnienia się kolejki. Jeżeli żaden kierunek nie doprowadza złodzieja do policjanta ruch złodzieja jest losowany. Jeżeli ścieżek doprowadzających złodzieja do policjanta jest więcej niż 1 złodziej wybiera kierunek, który znalazł policjanta najpóźniej. Jeśli tylko jedna ścieżka doprowadza złodzieja do policjanta, a pól graniczących z polem złodzieja jest więcej niż 1, to ruch złodzieja jest losowany ze wszystkich graniczących z polem złodzieja pól drogi, ponieważ nie da się jednoznacznie stwierdzić, który kierunek jest najlepszy dla złodzieja.

3.4.7 Algorytm dokonywania ruchu policjantów, w trybie gry złodziejem

1. Pierwsza część algorytmu ruchu policjantów (sprawdzanie czy policjanci stoją w jednej uliczce)

```
438     //function that's check if 2 policemen are in the same road
439     //if they are, the function makes the wall in anciBoard.Table on the perpendicular way between policemen
440     //in the cell adjacent to the moving policeman
441
442     private void PolicemanChecker (Character movingP, Character P2)
443     {
444         int temp = 0;
445         int temp2 = 0;
446         int temp3 = 0;
447         bool isWall = false;
448         bool isThief = false;
449         if(movingP.PositionX == P2.PositionX)
450         {
451             if (movingP.PositionY > P2.PositionY)
452             {
453                 temp3 = -1;
454                 temp2 = movingP.PositionY;
455                 temp = P2.PositionY + 1;
456             }
457             else
458             {
459                 temp3 = 1;
460                 temp2 = P2.PositionY;
461                 temp = movingP.PositionY + 1;
462             }
463             for(int i = temp; i < temp2; i++)
464             {
465                 if (myBoard.Table[movingP.PositionX, i].CurrentState == 1)
466                 {
467                     isWall = true;
468                 }
469                 else if(myBoard.Table[movingP.PositionX, i].CurrentState == 3)
470                 {
471                     isThief = true;
472                 }
473             }
474         }
475         else if(movingP.PositionY == P2.PositionY)
476         {
477             if (movingP.PositionX > P2.PositionX)
478             {
479                 temp3 = -1;
480                 temp2 = movingP.PositionX;
481                 temp = P2.PositionX + 1;
482             }
483             else
484             {
485                 temp3 = 1;
486                 temp2 = P2.PositionX;
487                 temp = movingP.PositionX + 1;
488             }
489             for (int i = temp; i < temp2; i++)
490             {
491                 if (myBoard.Table[i, movingP.PositionY].CurrentState == 1)
492                 {
493                     isWall = true;
494                 }
495                 else if (myBoard.Table[i, movingP.PositionY].CurrentState == 3)
496                 {
497                     isThief = true;
498                 }
499             }
500         }
501         if((movingP.PositionX == P2.PositionX || movingP.PositionY == P2.PositionY) && isThief == false && isWall == false)
502         {
503             if(movingP.PositionX == P2.PositionX)
504             {
505                 anciBoard.Table[movingP.PositionX, movingP.PositionY + temp3].CurrentState = 9;
506             }
507             else if (movingP.PositionY == P2.PositionY)
508             {
509                 anciBoard.Table[movingP.PositionX + temp3, movingP.PositionY].CurrentState = 9;
510             }
511         }
512     }
```

Objaśnienia

Algorytm sprawdza czy dwóch policjantów znajduje się w jednej kolumnie lub wierszu i jeśli tak jest, sprawdza czy pomiędzy policjantami występują tylko pola drogi nie zajmowane przez złodzieja. Jeśli dojdzie do opisanej wyżej sytuacji algorytm stawia specjalną ścianę pomiędzy policjantami w polu graniczącym z policjantem wykonującym ruch.

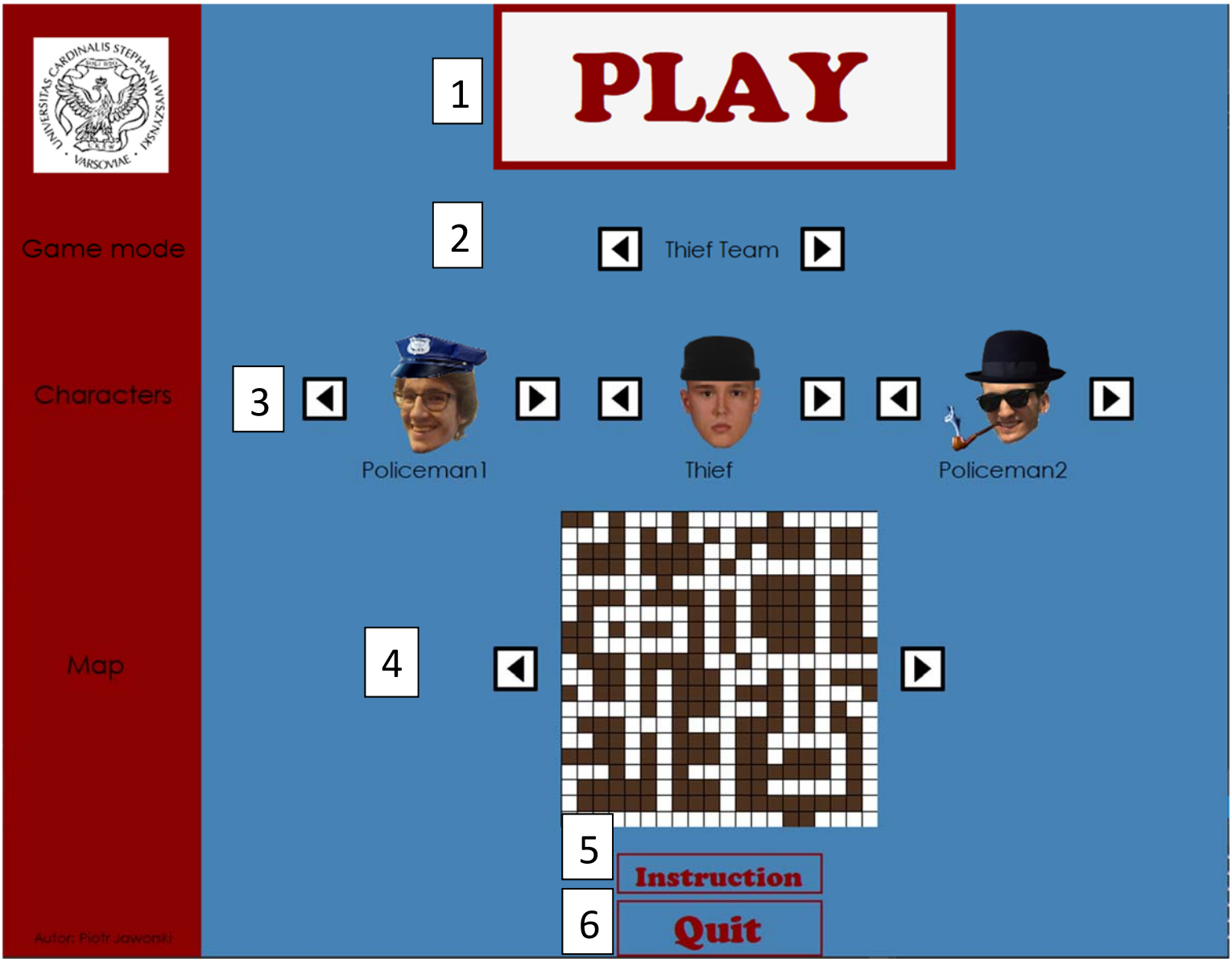
2. Druga część algorytmu ruchu policjantów (znalezienie ruchu najbardziej przybliżającego policjanta do złodzieja)

Objaśnienia

Kod tego algorytmu został zamieszczony w drugiej części algorytmu dokonywania ruchu złodzieja, ponieważ w obu przypadkach wykorzystywana jest ta sama funkcja. Algorytm działa w ten sam sposób jak w przypadku znajdowania najlepszego ruchu dla złodzieja z tą różnicą, że polem startowym poszukiwań najkrótszych ścieżek jest pole zajmowane przez policjanta wykonującego ruch, a docelowym pole zajmowane przez złodzieja. Zwracanym ruchem jest zawsze pole rozpoczynające ścieżkę, która jako pierwsza znalazła złodzieja.

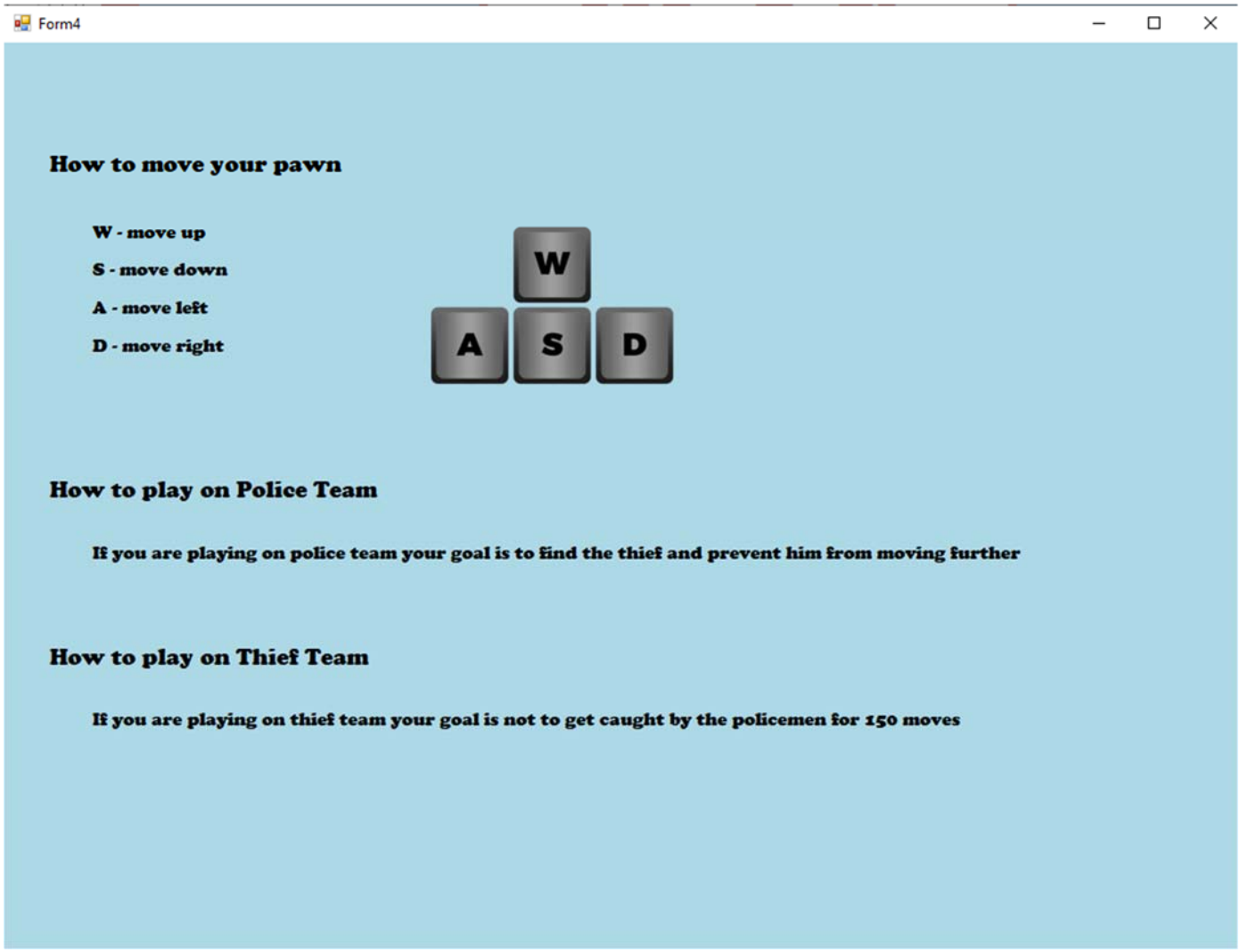
3.5 Interfejs aplikacji

3.5.1 Menu startowe



1. Przycisk Play rozpoczynający rozgrywkę, powoduje otwarcie nowego formularza z planszą
2. Wybór trybu gry
3. Wybór wyglądu pionków
4. Wygląd mapy (nie ma możliwości wyboru wyglądu mapy)
5. Przycisk otwierający nowy formularz z instrukcją do gry
6. Przycisk zamykający menu startowe

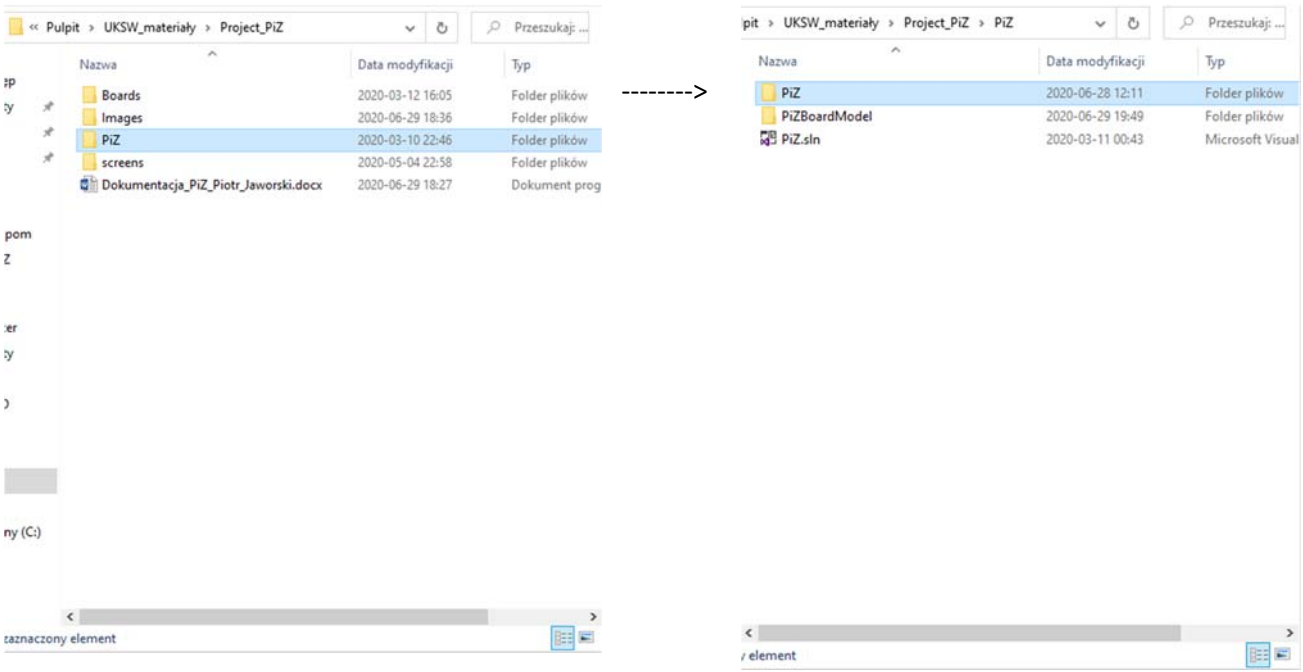
3.5.2 Instrukcja wyjaśniająca zasady gry i sterowanie pionkami

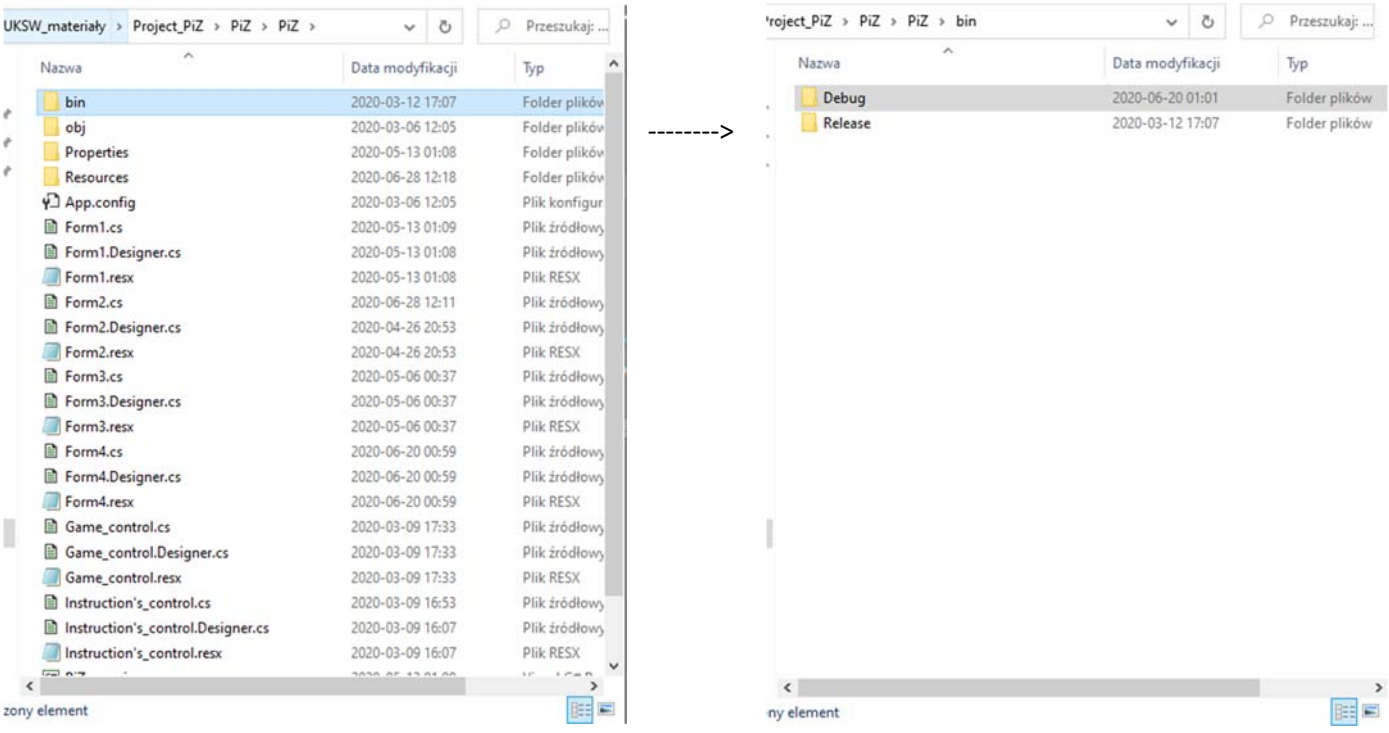


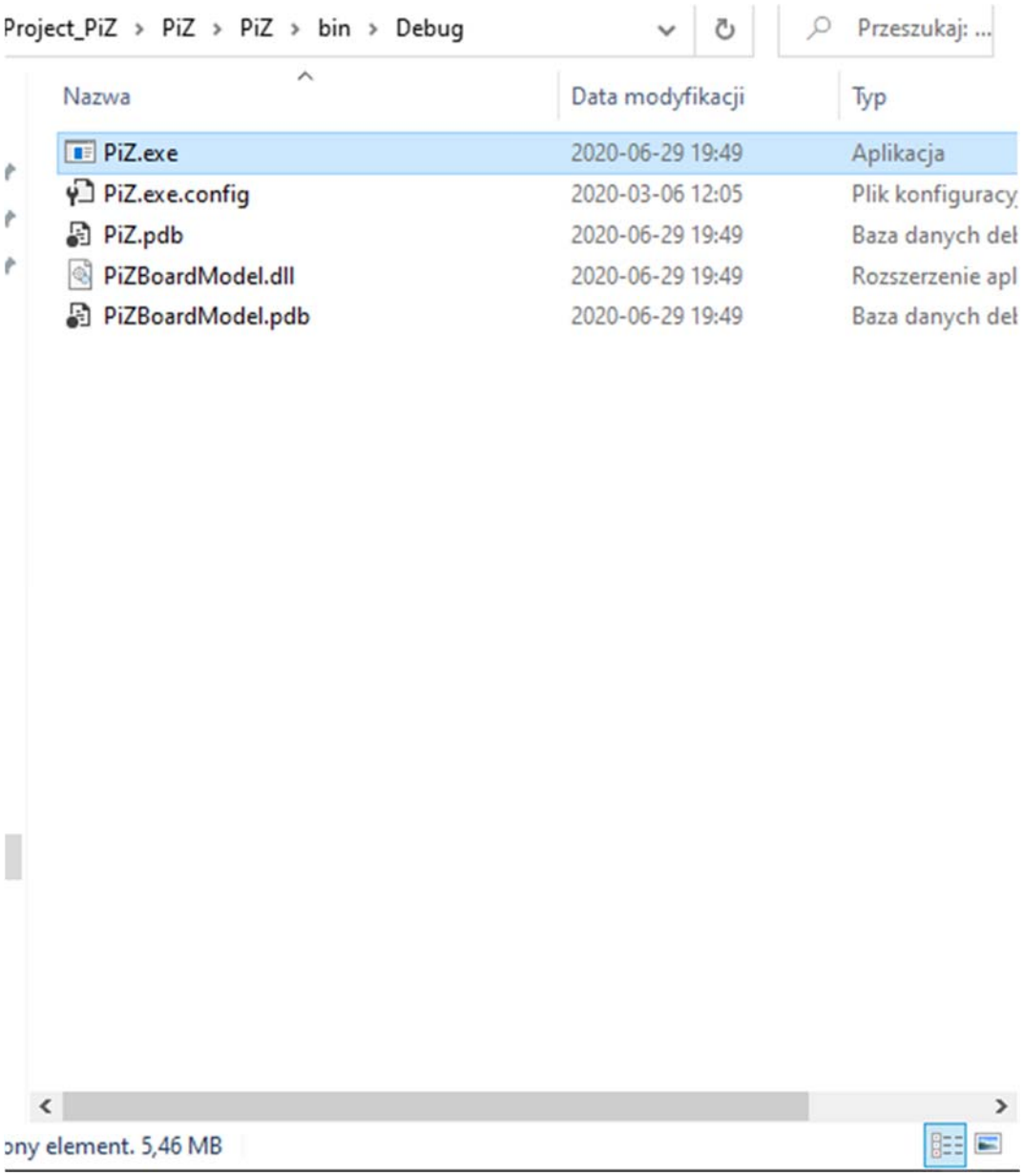
4. Instrukcja obsługi

4.1 Uruchamianie aplikacji

Aby uruchomić aplikacje potrzebny jest zainstalowany Microsoft Windows 7/8/10, a żeby skompilować kod programu potrzebny jest Microsoft Visual Studio 2013. Po otwarciu folderu projektu o nazwie „Project_PiZ” wyświetli nam się takie okno. Należy wtedy wejść w PiZ -> PiZ -> bin -> Debug -> PiZ.exe

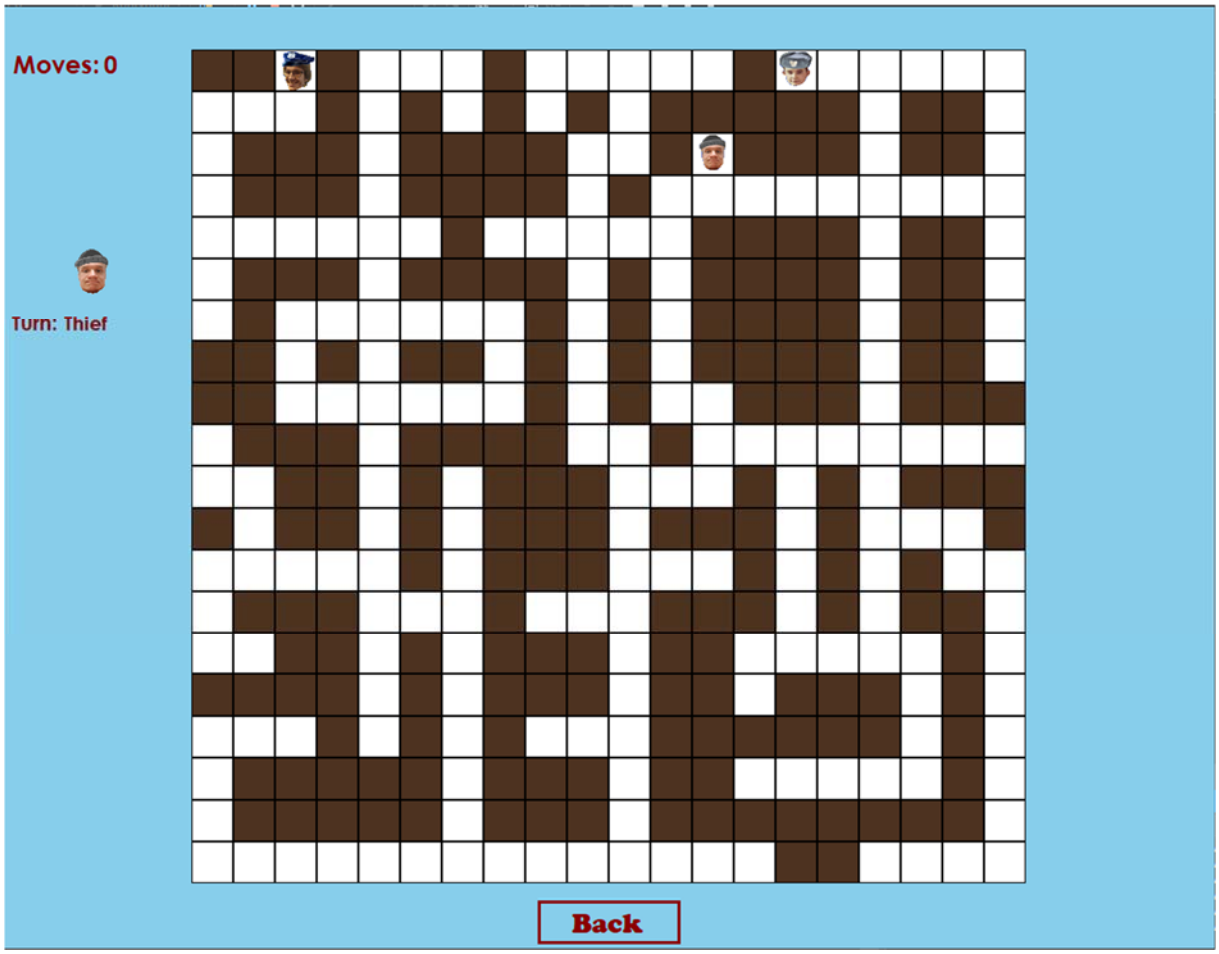




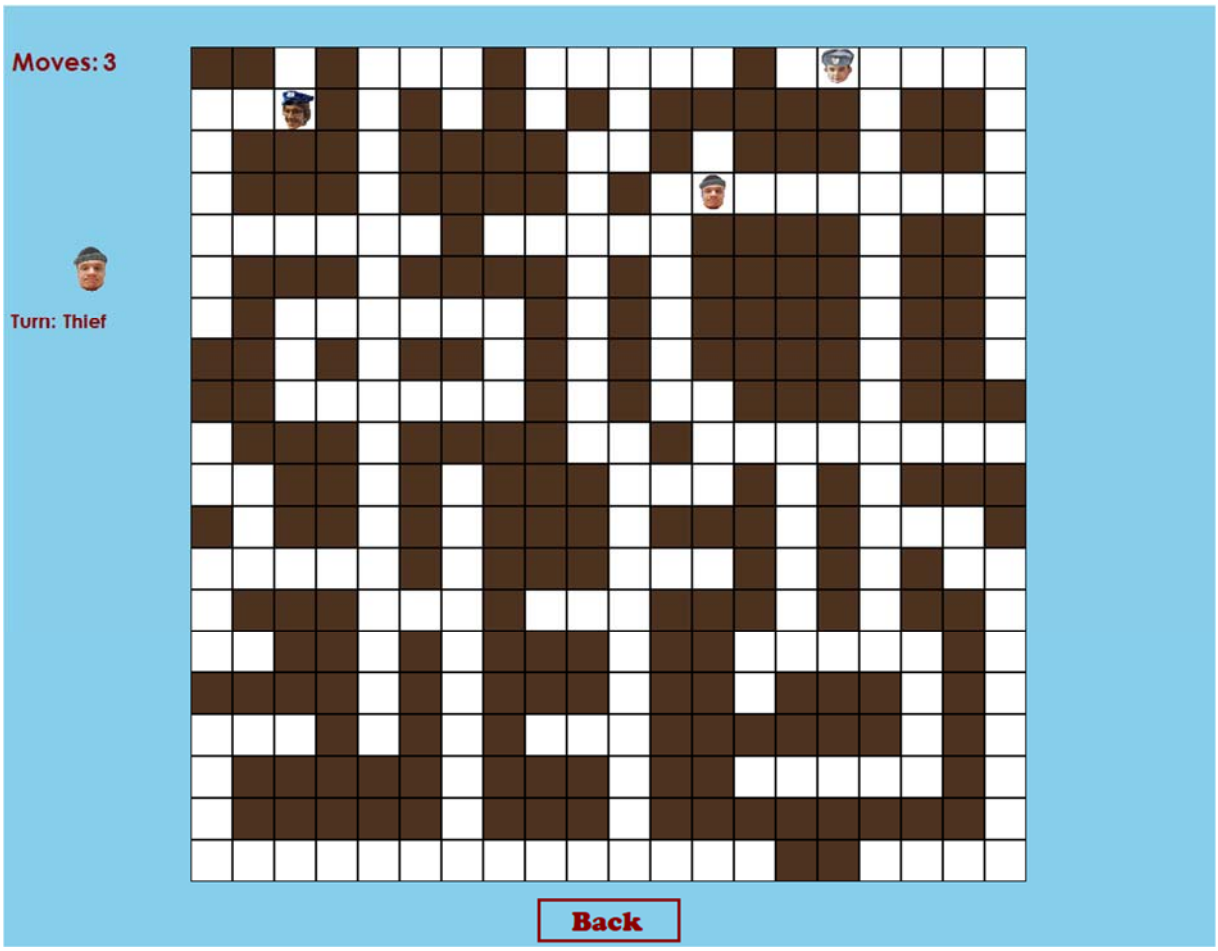

Project_PiZ › PiZ › PiZ › bin › Debug
Przeszukaj: ...
Nazwa
Data modyfikacji
Typ
PiZ.exe
2020-06-29 19:49
Aplikacja
PiZ.exe.config
2020-03-06 12:05
Plik konfiguracy
PiZ.pdb
2020-06-29 19:49
Baza danych dek
PiZBoardModel.dll
2020-06-29 19:49
Rozszerzenie apl
PiZBoardModel.pdb
2020-06-29 19:49
Baza danych dek
ony element. 5,46 MB

4.2 Przykładowy przebieg działania programu

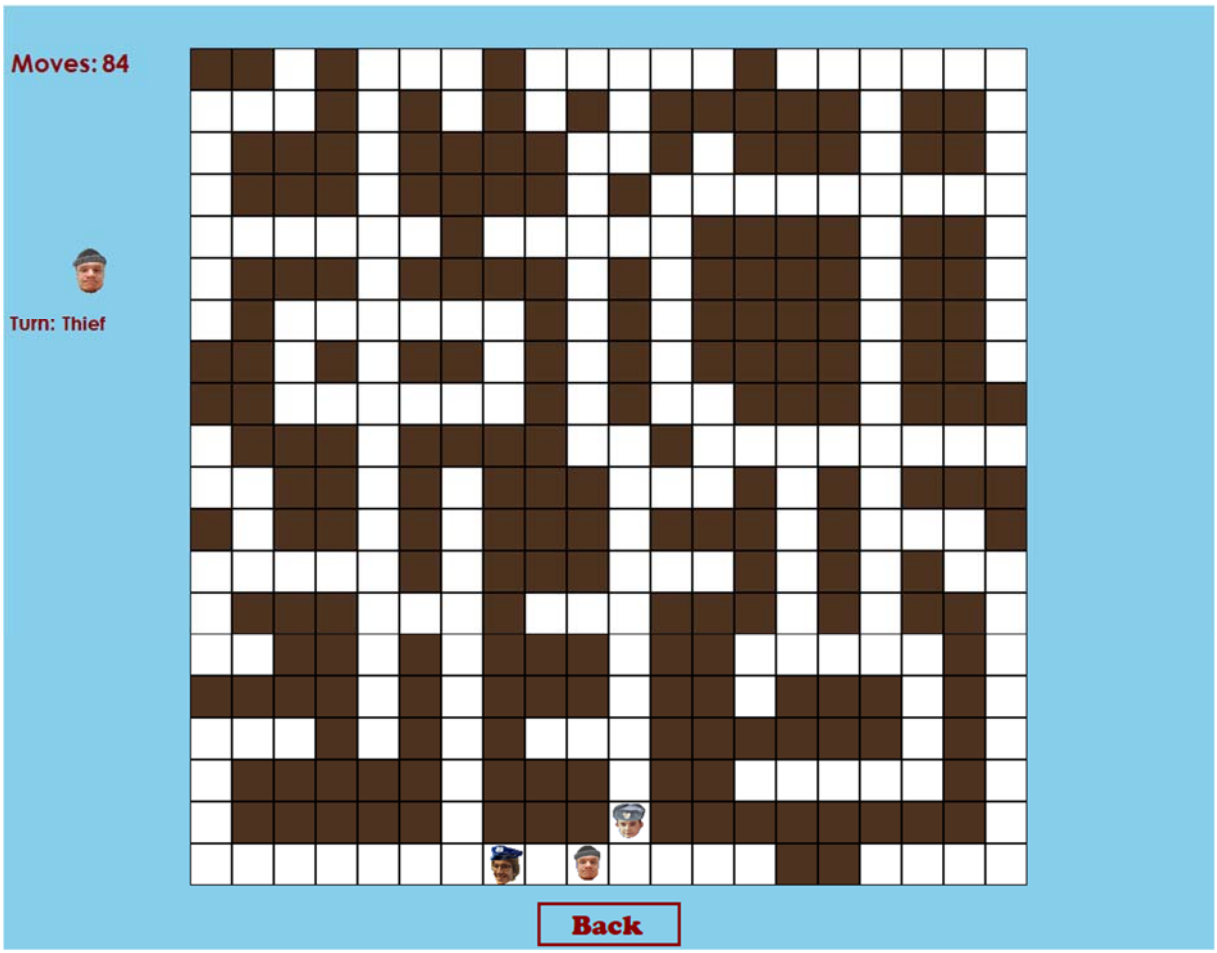
Pozycja startowa wszystkich pionków. W pierwszym wierszu planszy znajdują się dwaj policjanci, od lewej policjant1, policjant2, natomiast w trzecim wierszu znajduje się złodziej. Pierwszy ruch wykonywać będzie złodziej, a zatem gracz, tak jak jest to pokazane po lewej stronie w informatorze tur.



Okno gry po wykonaniu ruchu przez gracza złodziejem o jedno pole w dół i automatycznych ruchach policjantów

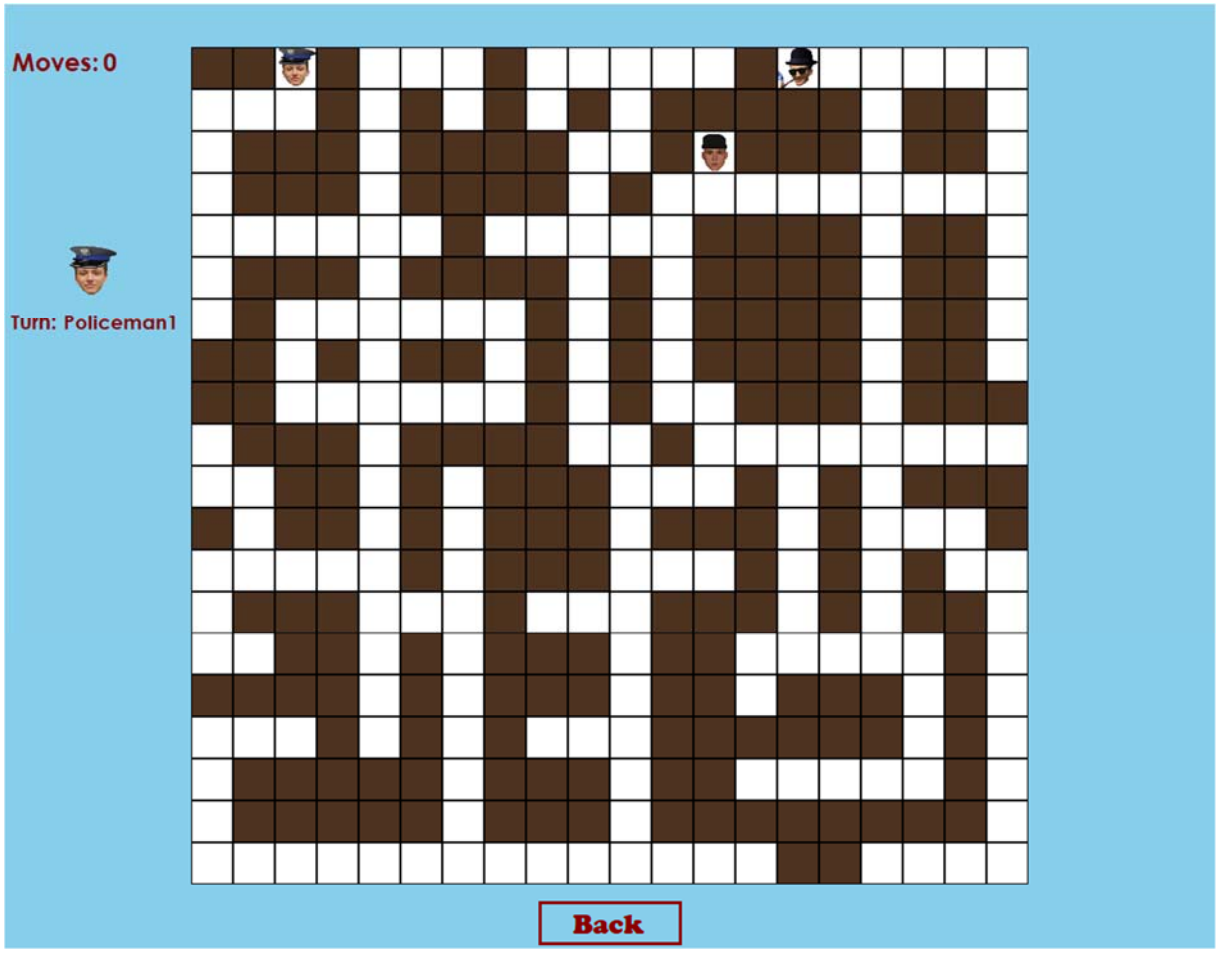


Okno gry po 84 ruchach, gdzie złodziej po wykonaniu swojego następnego ruchu zostanie złapany przez policjantów

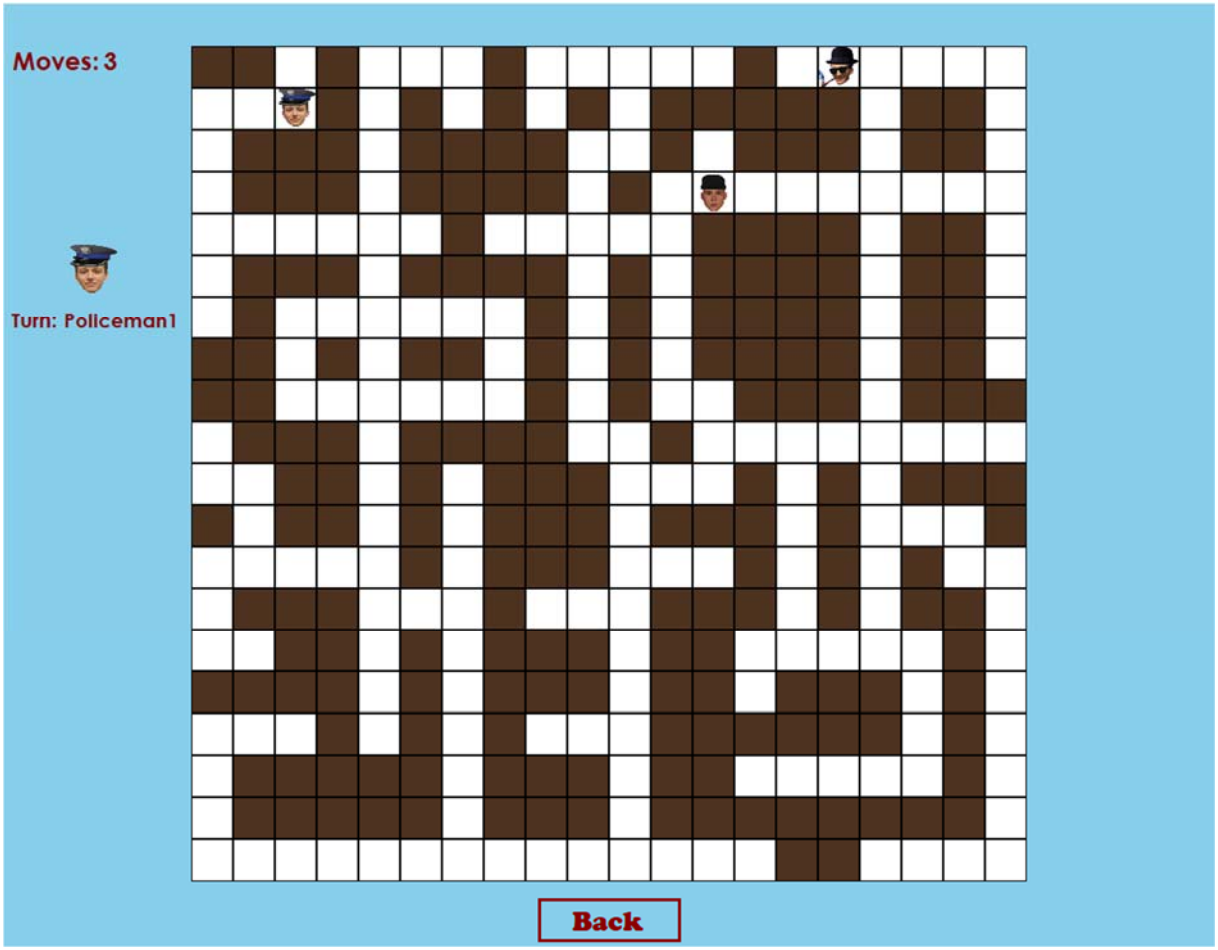


Po wygraniu policjantów wyświetli nam się takie okno, a po zamknięciu go wrócimy do okna menu startowego

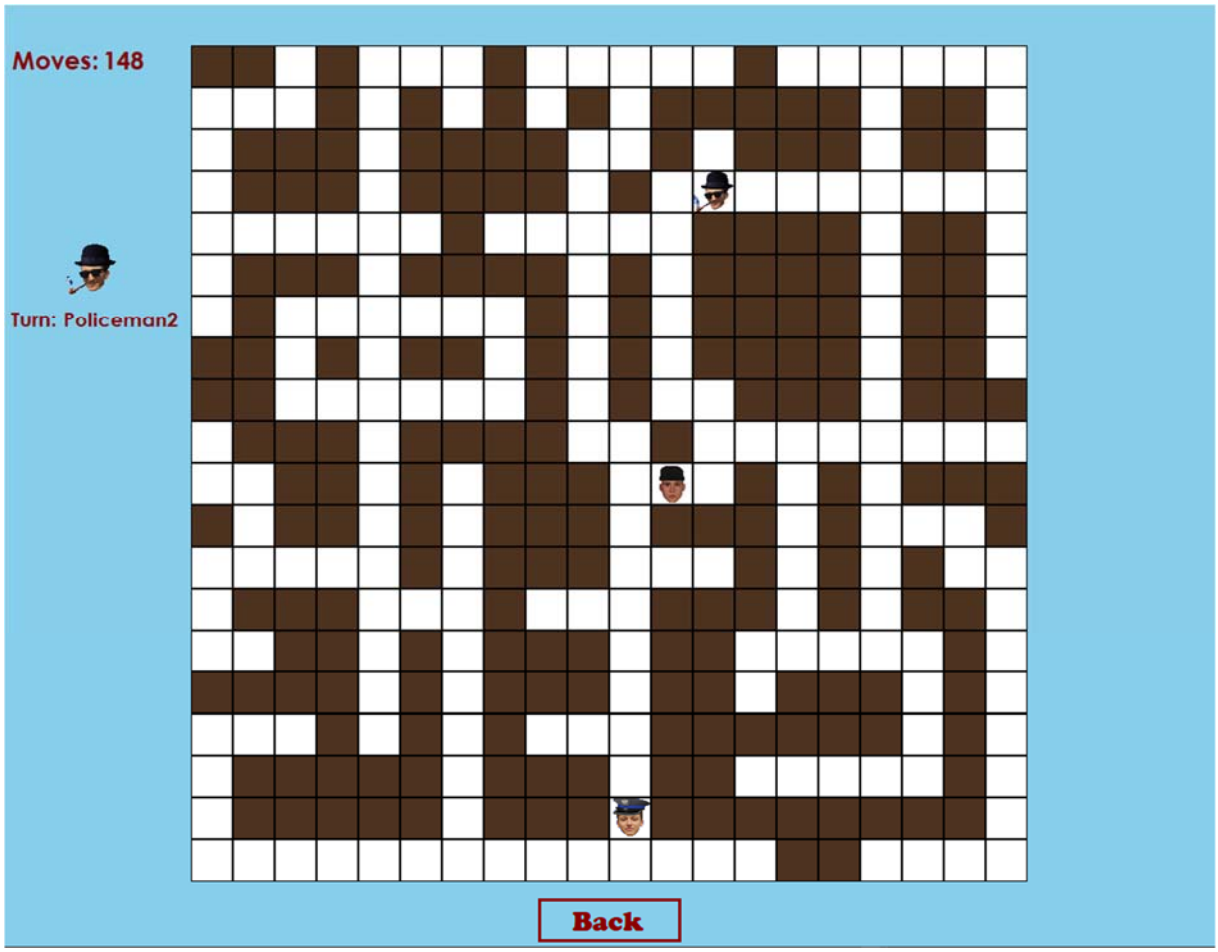
Pozycja startowa wszystkich pionków. W pierwszym wierszu planszy znajdują się dwaj policjanci, od lewej policjant1, policjant2, natomiast w trzecim wierszu znajduje się złodziej. Pierwszy ruch wykonywać będzie policjant1, a zatem gracz, tak jak jest to pokazane po lewej stronie w informatorze tur.



Okno gry po wykonaniu ruchu przez gracza policjantem1 o jedno pole w dół i policjantem2 o jedno pole w prawo i automatycznym ruchu złodzieja.



Okno gry po 148 ruchach. Po następnym ruchu policjanta2 złodziej wygra, ponieważ wykona 150 ruch ,po którym nie zostanie złapany przez, żadnego policjanta.



Po wygraniu złodzieja wyświetli nam się takie okno, a po zamknięciu go wrócimy z powrotem do okna menu

5. Bibliografia

[1] – Źródło: https://www.geeksforgeeks.org/shortest-path-in-a-binary-maze/

[2] – Źródło: https://visualstudio.microsoft.com/pl/

[3] – Źródło: https://www.microsoft.com/pl-pl/

6. Zawartość dysku CD

Na dysku znajdują się: folder „Boards”, w którym znajduje się plik tekstowy ciągiem cyfrowym opisującym rozplanowanie planszy, folder „Images”, w którym znajdują się obrazu użyte w programie, folder „PiZ”, w którym znajdują się pliki gry w tym plik .exe potrzebny do uruchomienia gry, folder „screens”, w którym znajdują się zrzuty ekranu użyte w trakcie raportowania postępu w pracy nad projektem oraz dokumentacja całego projektu w pliku .pdf