POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Automatyka i Robotyka (AIR)
SPECJALNOŚĆ: Robotyka (ARR)

PRACA DYPLOMOWA INŻYNIERSKA

Badanie własności algorytmów poszukiwania ścieżki w labiryncie dla robota klasy micromouse

Maze solving algorithms analysis for micromouse mobile robots

AUTOR:
Dawid Perdek

PROWADZĄCY PRACĘ:
dr inż. Robert Muszyński, I-6

OCENA PRACY:

WROCŁAW 14 grudnia 2015

Ludziom, którzy swoim nieróbstwem
motywowali mnie do cięższej pracy.

Spis treści

Rozdział 1

Wstęp

Labirynty są znane ludzkości od tysięcy lat, a ich główną rolą było utrudnianie dostępu do ważnych, pilnie strzeżonych miejsc, które ulokowane były zazwyczaj w centralnej części budowli. Najczęściej kojarzone są z grobowcami egipskich faraonów, posadzkami średniowiecznych kościołów oraz z mitologią, a symbolika tego elementu znana jest szeroko na całym świecie [20]. Przykład labiryntu przedstawiono na rysunku 1.1. Obecnie jednak na labirynty patrzy się głównie przez pryzmat nauki — jako na ciekawy obiekt umożliwiający testowanie algorytmów przeszukiwania zbioru możliwych rozwiązań i odnajdywania najkrótszej ścieżki. W ciągu kilku ostatnich dekad dynamicznie rozwijała się technika, a jej coraz ważniejszym działem stawała się robotyka. Kluczową gałęzią robotyki zawsze była robotyka przemysłowa, lecz wraz z biegiem czasu coraz popularniejsza staje się robotyka amatorska i to właśnie wśród ludzi nią się zajmujących wciąż aktualny i ważny jest problem poszukiwania ścieżki w labiryncie. Jedną z najciekawszych i najbardziej wymagających konkurencji na zawodach robotów jest konkurencja MicroMouse polegająca właśnie na znalezieniu ścieżki do środka labiryntu przez w pełni autonomicznego robota mobilnego. Zdecydowana większość konstrukcji ma bardzo zbliżone kształty i możliwości jezdne, więc o sukcesie w takiej rywalizacji decyduje przede wszystkim dobór algorytmu. Ścieżka do mety może być optymalizowana ze względu na jak najmniejszą liczbę przejechanych pól lub przez wzgląd na jak najmniejszą liczbę zakrętów. Istotnym elementem strategii jest również takie skonstruowanie robota, by zapewnić mu możliwość poruszania się między komórkami labiryntu pod kątem 45°, zamiast wykonywania wszystkich skrętów pod kątem prostym.

Rysunek 1.1: Przykładowy labirynt [20]

1.1 Cel pracy

Celem pracy jest ustalenie, który z popularnych algorytmów proponowanych do poszukiwania ścieżki w labiryncie jest najbardziej odpowiedni dla robota mobilnego klasy MicroMouse. Wyniki mogą być przydatne osobom chcącym zbudować własną konstrukcję tego typu oraz zajmującym się szerzej pojętym problemem poszukiwania drogi w labiryncie. W ramach pracy zachowane zostaną wszelkie wymogi i zasady obowiązujące na najważniejszych krajowych zawodach robotów w kategorii MicroMouse. Jednocześnie przy opisach poszczególnych algorytmów i w ich schematach blokowych w dalszych rozdziałach pominięte zostaną kwestie lokalizacji robota, akwizycji, przechowywania i interpretacji danych oraz realizacji ruchu. Obowiązuje założenie, że robot nie umie przemieszczać się po przekątnych.

Układ pracy jest następujący. W rozdziale 2 omówiono najważniejsze cechy robotów MicroMouse oraz zasady obowiązujące w konkurencji MicroMouse. W rozdziale 3 przedstawiono badane algorytmy, natomiast wyniki badań umieszczono w rozdziale 4. Rozdział 5 poświęcono wnioskom płynącym z badań, odkrytym w ich trakcie problemom oraz możliwościom dalszego rozwoju pracy.

Rozdział 2

Zawody robotów mobilnych klasy MicroMouse

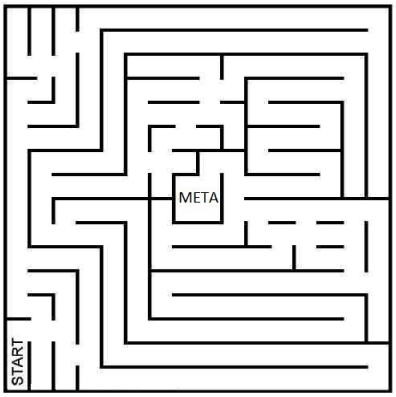
Zawody robotów są w Polsce organizowane od około dekady, a w ostatnich latach na popularności zyskuje konkurencja MicroMouse, mająca swoje początki w Japonii w późnych latach 70. XX wieku [19]. Na liście większych krajowych imprez tego typu znajdują się takie wydarzenia jak warszawski *Robomaticon* [11], organizowana we Wrocławiu *Robotic Arena* [12], krakowski *Robocomp* [2] czy odbywający się w Poznaniu *Festiwal Robotyki Cyberbot* [3]. Na wszystkich wymienionych zawodach rozgrywana jest konkurencja MicroMouse, a analizując obowiązujące tam regulaminy można stwierdzić, że poza drobiazgami są one do siebie bardzo zbliżone, jeśli nie identyczne [8, 7, 9, 10].

Dalej przedstawiono bardziej szczegółową charakterystykę konkurencji MicroMouse oraz krótki opis wymagań wobec robota i zadań, które musi umieć on zrealizować, by z powodzeniem walczyć o zwycięstwo w zawodach.

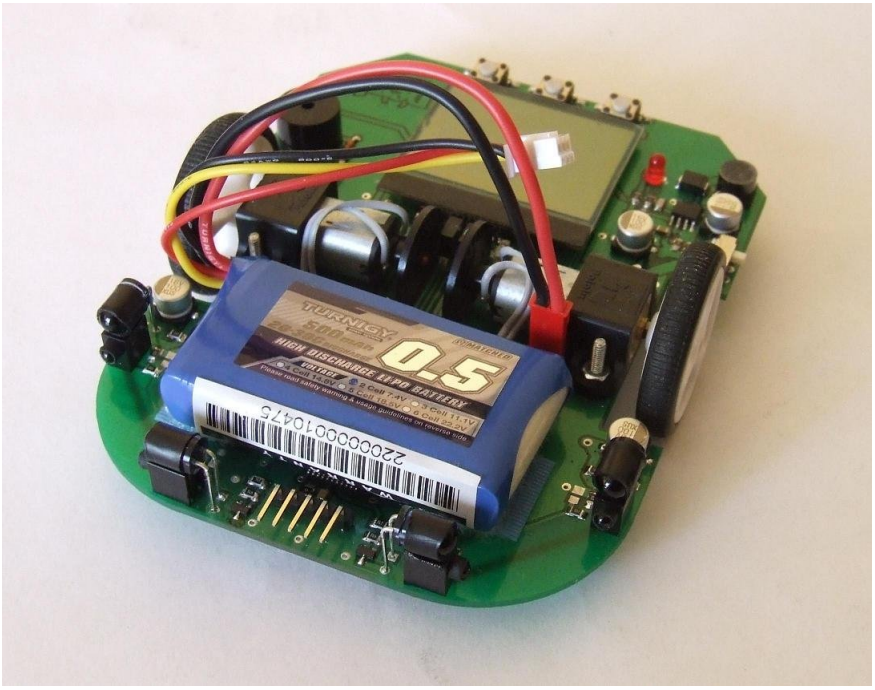
2.1 Opis konkurencji MicroMouse

Konkurencja rozgrywana jest w labiryncie o kształcie kwadratu o boku długości 288 cm. Pojedyncza komórka ma wymiary 180x180 mm, co przy ściankach grubości 12 mm zostawia kwadrat o boku 168 mm, wewnątrz którego może poruszać się robot. Wszystkie ścianki labiryntu mają wysokość 50 mm i są pomalowane na biało [8, 7, 9, 10]. Schemat przykładowego labiryntu wykorzystywanego w trakcie zawodów pokazano na rysunku 2.1.

Konkurencja rozgrywana jest jednoetapowo — każdemu uczestnikowi przysługuje 10 minut na wykonanie dowolnej liczby przejazdów. Spośród wszystkich czasów przejazdu od pola startowego do środka labiryntu pod uwagę brany będzie najlepszy. Zawody wygrywa konstruktor, którego robot pokona tę trasę w najkrótszym czasie. Podczas trwania rozgrywki komunikacja z robotem jest zabroniona, z wyjątkiem zdalnego startowania i zatrzymywania [8, 7, 9, 10].



Rysunek 2.1: Przykładowy labirynt wykorzystywany w zawodach MicroMouse (na podstawie [6])



Rysunek 2.2: Robot MicroMouse — Devil [1]

2.2 Wymagania stawiane robotowi

Od strony formalnej kryteria kwalifikacji robota do udziału w konkurencji mówią, że musi to być autorska konstrukcja, o wymiarach pozwalających poruszać się w labiryncie, przy czym waga i wysokość są nieograniczone. Konieczne jest, by urządzenie potrafiło przemieszczać się w pełni autonomicznie, tylko po podłodze labiryntu oraz bez ingerencji w jego strukturę [8, 7, 9, 10].

Od strony technicznej robot musi być w stanie sprostać podstawowym zadaniom, jakie czekają na niego w labiryncie, takim jak:

- obrót o zadany kąt,
- przemieszczenie o zadany dystans w określonym kierunku,
- określanie swojej lokalnej orientacji i pozycji (w obrębie komórki),
- określanie swojego globalnego położenia (w obrębie całego labiryntu),
- tworzenie w pamięci mapy labiryntu i jej bieżące analizowanie,
- wyznaczenie najkrótszej ścieżki.

Na rysunku 2.2 przedstawiono przykładową konstrukcję robota klasy MicroMouse.

Rozdział 3

Algorytmy poszukiwania ścieżki

W pracy przeanalizowano najbardziej popularne algorytmy stosowane w robotach klasy MicroMouse, wybrane na podstawie artykułów i tematów na forach internetowych [15], raportów z budowy takich konstrukcji [4] oraz doświadczeń autora. Najczęściej wykorzystywanym, i jednocześnie osiągającym najwięcej sukcesów, algorytmem jest algorytm zalewający (tzw. *flood fill*) [15, 18]. Często wykorzystywane są również *metoda lewej/prawej ręki* [18] oraz algorytmy grafowe [14], np. *algorytm Dijkstry*, *przeszukiwanie wszerz* czy *przeszukiwanie w głąb*. Zdecydowana większość algorytmów skupia się na poszukiwaniu ścieżki najkrótszej, podczas gdy niekoniecznie jest ona najszybsza. Dla robotów klasy MicroMouse równie istotna co długość ścieżki jest liczba zakrętów jaką muszą pokonać, dlatego też wybór ścieżki potencjalnie najszybszej może być oparty o kryterium minimalizacji liczby zakrętów.

W bieżącym rozdziale przedstawione zostaną zasady działania poszczególnych algorytmów, ich schematy blokowe oraz fragmenty przejazdu przez labirynt z rysunku 2.1. Zawarte w rozdziale rysunki przedstawiające fragmenty przejazdu, pochodzą z aplikacji stworzonej na potrzeby testowania działania badanych algorytmów. Komórki traktowane przez robota jako docelowa ścieżka oznaczane są zielonymi kwadratami, natomiast mniejsze, czerwone kwadraty oznaczają komórki odwiedzone w trakcie poszukiwania rozwiązania, jednakże odrzucone. Szczegółowy opis aplikacji zawarty jest w dodatku A. Zakłada się, że czytelnik zna podstawy teorii grafów i rozumie pojęcia takie jak *relaksacja*, *graf skierowany*, *graf nieskierowany*, *waga krawędzi* i *cykl w grafie*, ponadto zna również terminy *algorytm zachłanny*, *algorytm dokładny* i *algorytm heurystyczny*.

3.1 Algorytm losowy

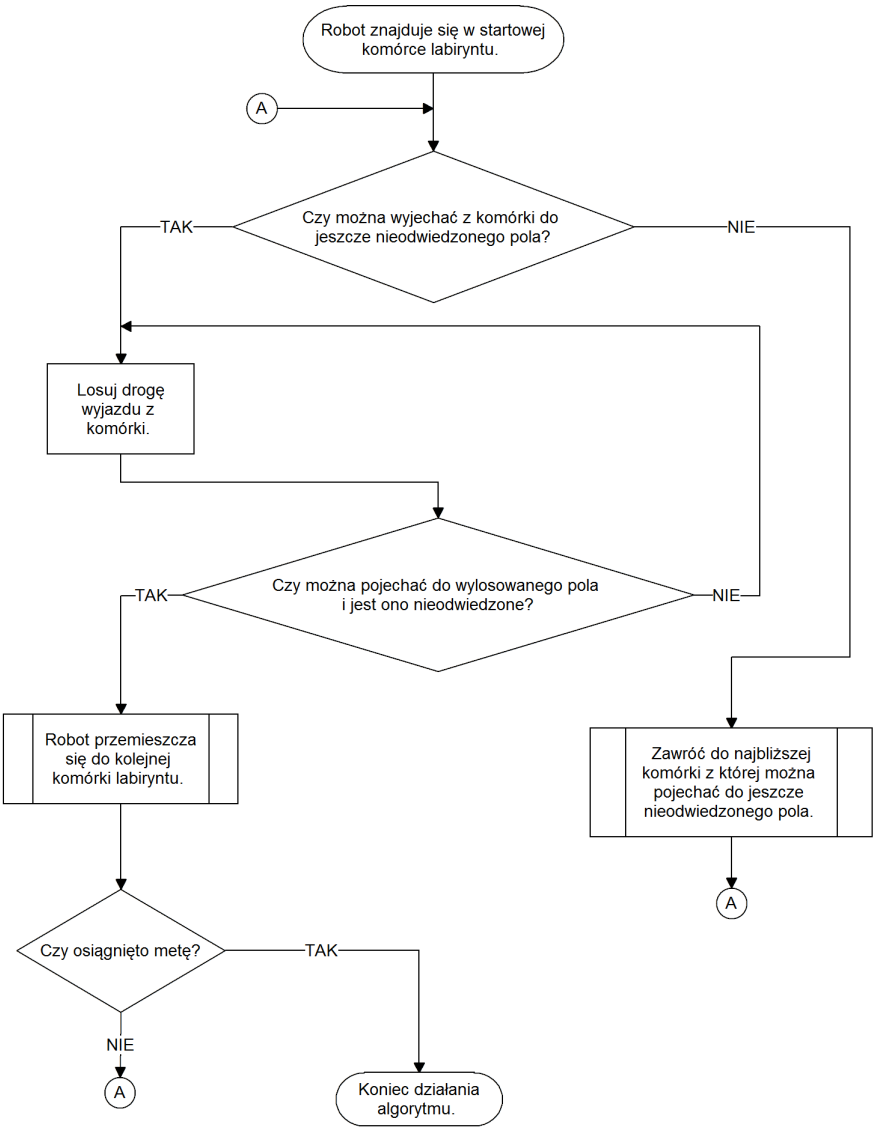
Algorytm losowy stanowi podejście gwarantujące znalezienie jakiegoś rozwiązania, choć prawdopodobnie nieoptymalnego. Czas poszukiwania rozwiązania jest trudny do oszacowania, ze względu na to, że zależy nie tylko od samego schematu labiryntu, ale również od wyników wielu losowań. Nieuporządkowany sposób przemieszczenia się robota sprawia, że wiele komórek może zostać odwiedzonych wielokrotnie. Na rysunku 3.1 widnieje schemat blokowy algorytmu.

Główną zaletą tego algorytmu jest pewność, że rozwiązanie zostanie znalezione. Natomiast jego główną wadą jest brak pewności co do czasu poszukiwania tegoż rozwiązania. Na rysunku 3.2 przedstawiony jest fragment przykładowego przejazdu.

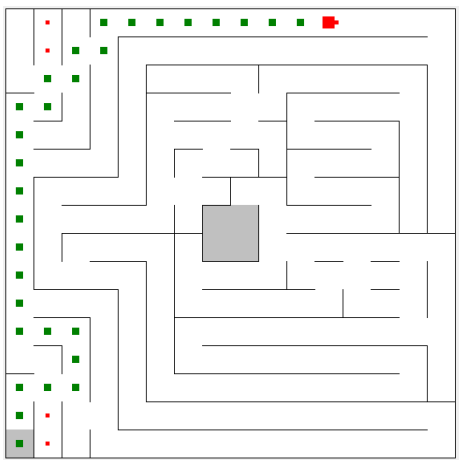
3.2 Algorytm lewej/prawej ręki

Algorytm lewej/prawej ręki [18] to najprostsze podejście do rozwiązywania labiryntu. Opiera się on na banalnej zasadzie — należy ustalić czy śledzona będzie ściana po lewej czy po prawej stronie, a następnie jechać wzdłuż wybranej ściany wykonując wszelkie konieczne skręty. W prezentowanej wersji algorytmu w przypadku wjazdu w ślepą uliczkę robot zawraca nadal śledząc konkretną ścianę, natomiast w przypadku powrotu do wcześniej odwiedzonej komórki wybiera z niej drogę jeszcze nie sprawdzoną. Schemat blokowy algorytmu przedstawiony jest na rysunku 3.3.

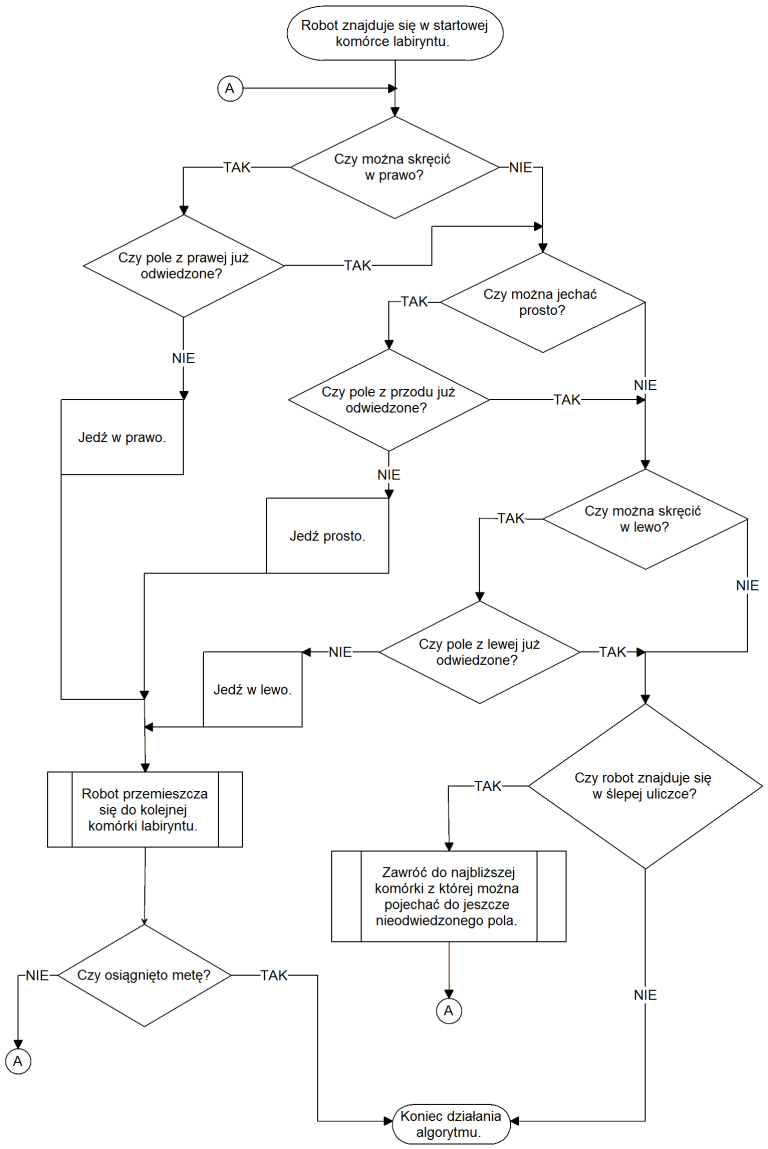
Omawiany algorytm posiada bardzo poważną wadę — nie każdy labirynt da się rozwiązać z niego korzystając. Pomijając czas przeszukiwania labiryntu, który może okazać się długi, istnieje spore prawdopodobieństwo, że dotarcie do mety tą metodą w ogóle nie jest możliwe. Nawet jeśli ścieżka zostanie odnaleziona, nie ma żadnej gwarancji, że jest to ścieżka najkrótsza. Fragment przykładowego przejazdu robota przez labirynt został pokazany na rysunku 3.4.



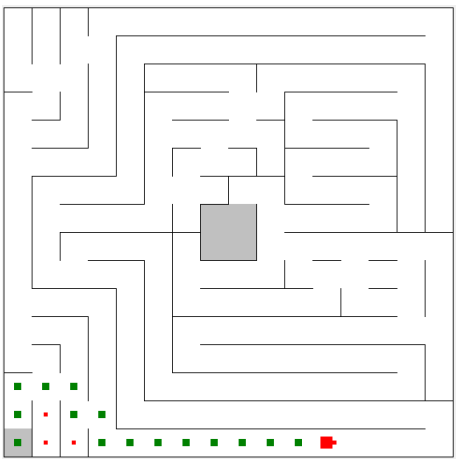
Rysunek 3.1: Schemat blokowy algorytmu losowego



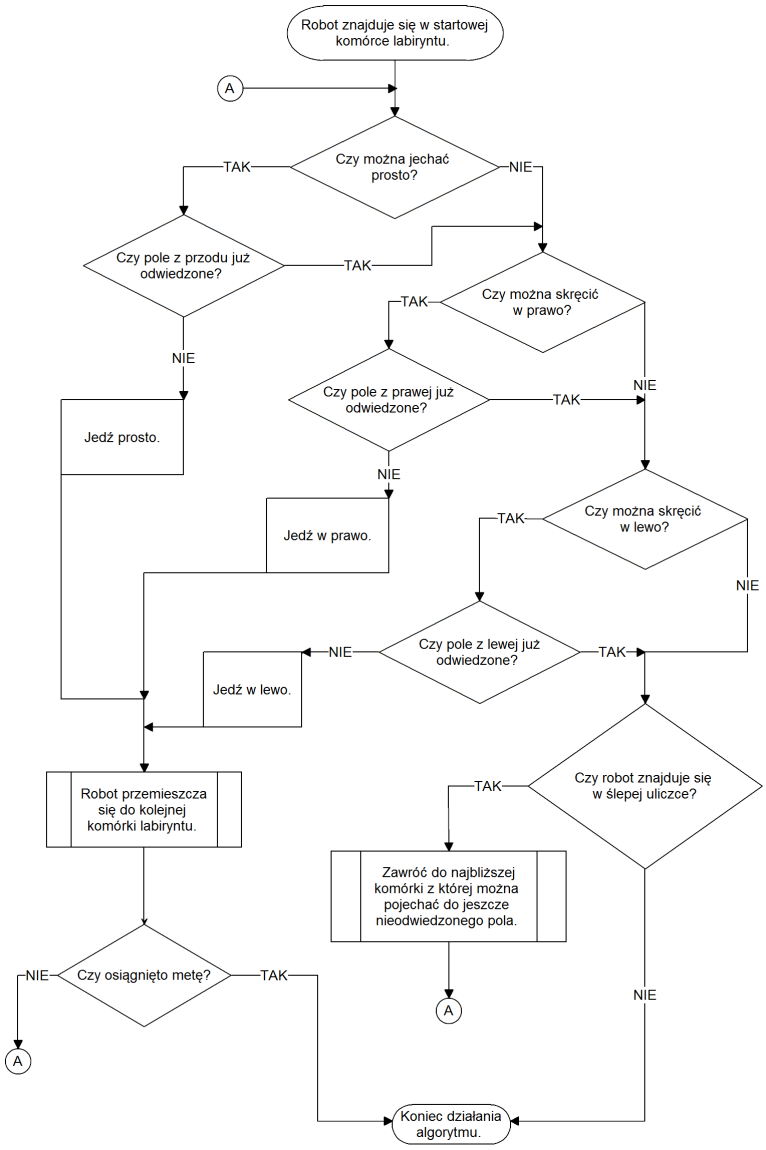
Rysunek 3.2: Fragment przykładowej trasy robota korzystającego z algorytmu losowego



Rysunek 3.3: Schemat blokowy algorytmu prawej ręki



Rysunek 3.4: Fragment przykładowej trasy robota korzystającego z algorytmu prawej ręki



Rysunek 3.5: Schemat blokowy zmodyfikowanego algorytmu prawej ręki

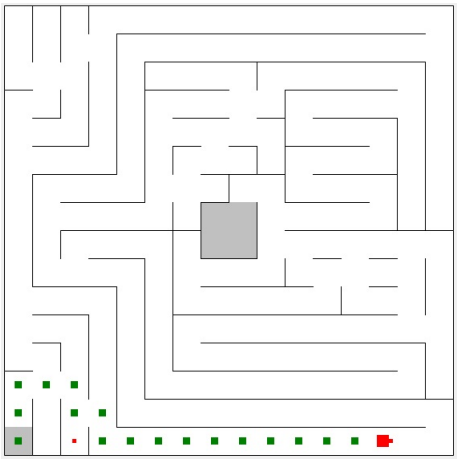
3.2.1 Zmodyfikowany algorytm lewej/prawej ręki

Mając na uwadze niedoskonałości algorytmu lewej/prawej ręki postanowiono wprowadzić do niego pewną modyfikację mającą na celu stworzenie szansy na rozwiązanie labiryntu niemożliwego do rozwiązania przez zwykłą wersję algorytmu oraz zwiększenie szans na osiągnięcie dobrego czasu przez szybkie roboty, zyskujące dużą przewagę na prostych odcinkach. Istotą wprowadzanej zmiany jest nadrzędne traktowanie prostych korytarzy w labiryncie — robot szuka skrętu w określonym kierunku dopiero, gdy nie ma możliwości jazdy w przód. Takie podejście określane jest jako *central-left/central-right rule* [13]. Schemat blokowy algorytmu przedstawiony jest na rysunku 3.5.

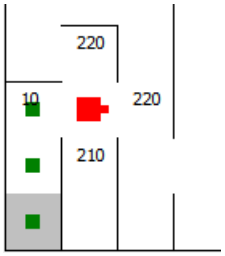
O ile próba wypracowania w ten sposób przewagi może zadziałać zupełnie odwrotnie i dla konkretnych labiryntów znacząco pogarszać czas rozwiązania (lub nawet je uniemożliwiać), o tyle modyfikacja ta faktycznie może pozwolić na znalezienie ścieżki w labiryncie, z którym standardowy algorytm sobie nie poradzi. Nie gwarantuje jednak, że będzie to ścieżka najkrótsza. Fragment przejazdu robota przez labirynt przedstawiony jest na rysunku 3.6.

3.3 Metoda pól potencjałów

Metody sztucznych pól potencjałów są szeroko stosowane przy rozwiązywaniu zagadnienia wyznaczania trasy robota mobilnego w otwartej przestrzeni pełnej nieregularnie rozstawionych przeszkód [21, 16, 5].



Rysunek 3.6: Fragment przykładowej trasy robota korzystającego ze zmodyfikowanej wersji algorytmu prawej ręki



Rysunek 3.7: Przykład wyliczenia potencjałów sąsiednich pól

W labiryncie jedyne możliwe przeszkody to ściany, a ich układ jest regularny, więc ta grupa metod nie jest zbyt popularna wśród konstruktorów. Ich ogólna idea polega na nadaniu punktowi docelowemu potencjału o wybranym znaku, następnie nadaniu przeciwnego potencjału robotowi — w ten sposób robot jest przyciągany do celu. Efekt omijania przeszkód uzyskuje się nadając wykrytym obiektom potencjał o znaku zgodnym ze znakiem potencjału robota, lecz o mniejszej, zależnej od odległości, wartości [21, 16, 5]. Badany algorytm powstał w oparciu o sztuczne pola potencjałów i polega na wybieraniu kierunku jazdy zgodnie z rosnącą wartością wypadkowego potencjału. Potencjał ten jest zawsze liczony dla każdej sąsiedniej komórki według wzoru,

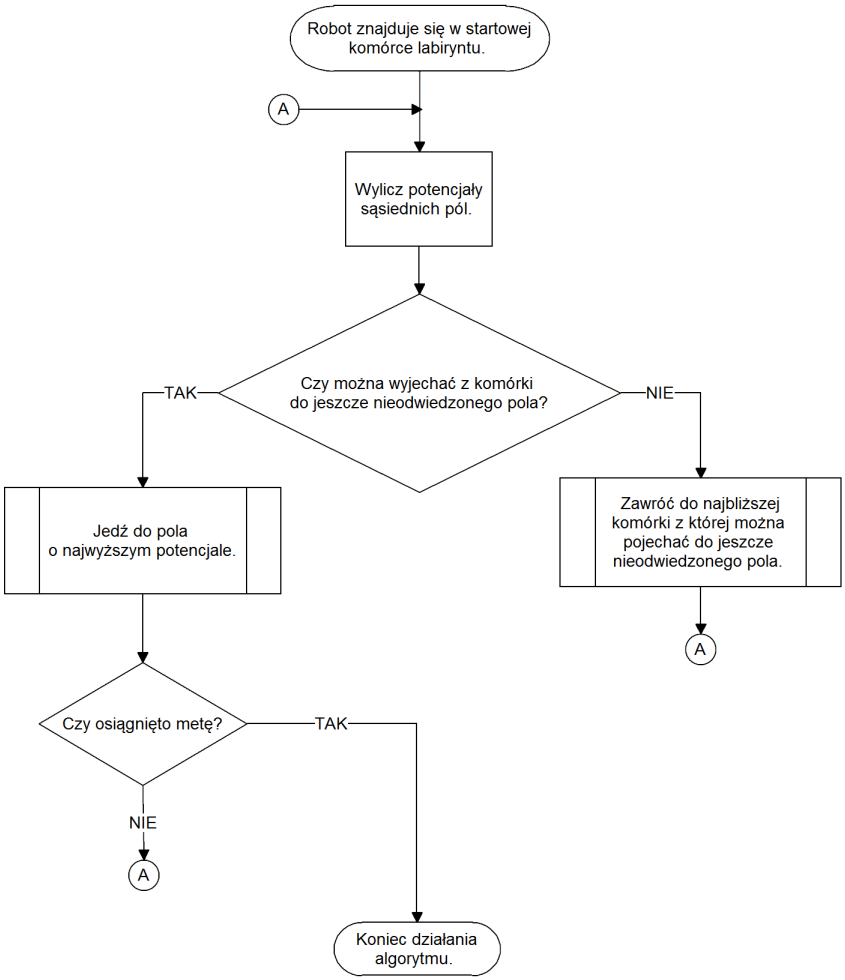
$$PotW = PotM - PotR + PotS - 3 * POT * sw - 2 * POT * odw.$$

Zmienna POT oznacza wartość założonego domyślnego potencjału, podczas gdy $PotW$ oznacza potencjał wypadkowy dla rozważanej komórki, $PotM$ oznacza potencjał końca labiryntu, a $PotR$ potencjał robota. $PotS$ reprezentuje pierwotny potencjał komórki nadawany na podstawie jej odległości od środka labiryntu w metryce taksówkowej (zakładając brak ścian). Występujące w równaniu współczynniki, przyjmują wartość 1 lub 0 i oznaczają obecność ściany (współczynnik sw: 1 gdy jest ściana) oraz fakt ponownego wejścia do komórki (współczynnik odw: 1 gdy komórka była odwiedzona). Na rysunku 3.7 przedstawiono przykład wyliczenia potencjałów sąsiednich pól.

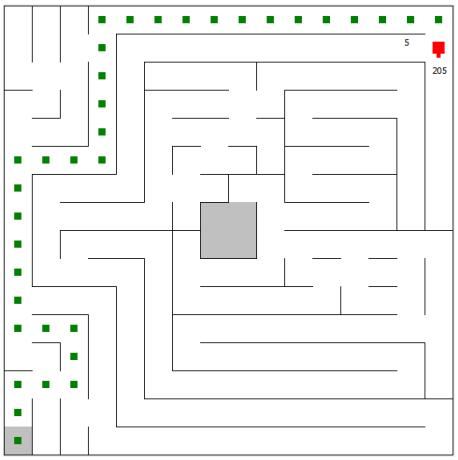
Omawiany algorytm gwarantuje znalezienie rozwiązania, ale nie zapewnia, że będzie ono optymalne. Zależnie od ukształtowania labiryntu możliwe jest zarówno szybkie wyznaczenie ścieżki najkrótszej, jak i długi przejazd w trakcie którego odwiedzone zostaną wszystkie komórki. Na rysunku 3.8 przedstawiony jest schemat blokowy algorytmu, a na rysunku 3.9 fragment przykładowego przejazdu przez labirynt.

3.4 Algorytm zalewający

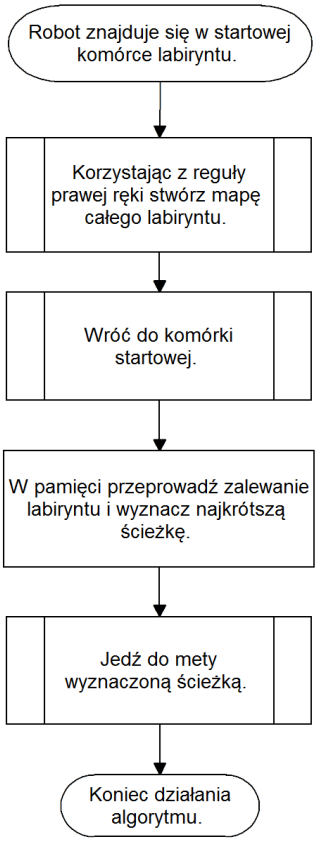
Algorytm zalewający (inaczej: *flood-fill*, *metoda propagacji fali*) [15, 18] wywodzi się ze służącego do wyznaczania najkrótszej ścieżki w grafie algorytmu Bellmana-Forda. Idea algorytmu polega na stopniowym wyznaczaniu najkrótszych ścieżek prowadzących od mety do każdej komórki labiryntu. Po tym procesie wszystkie pola mają przyporządkowaną wartość oznaczającą odległość od mety i wystarczy, by robot



Rysunek 3.8: Schemat blokowy algorytmu opartego na metodach sztucznych pól potencjałów



Rysunek 3.9: Fragment przykładowej trasy robota korzystającego z algorytmu opartego na metodach sztucznych pól potencjałów



Rysunek 3.10: Schemat blokowy algorytmu zalewającego

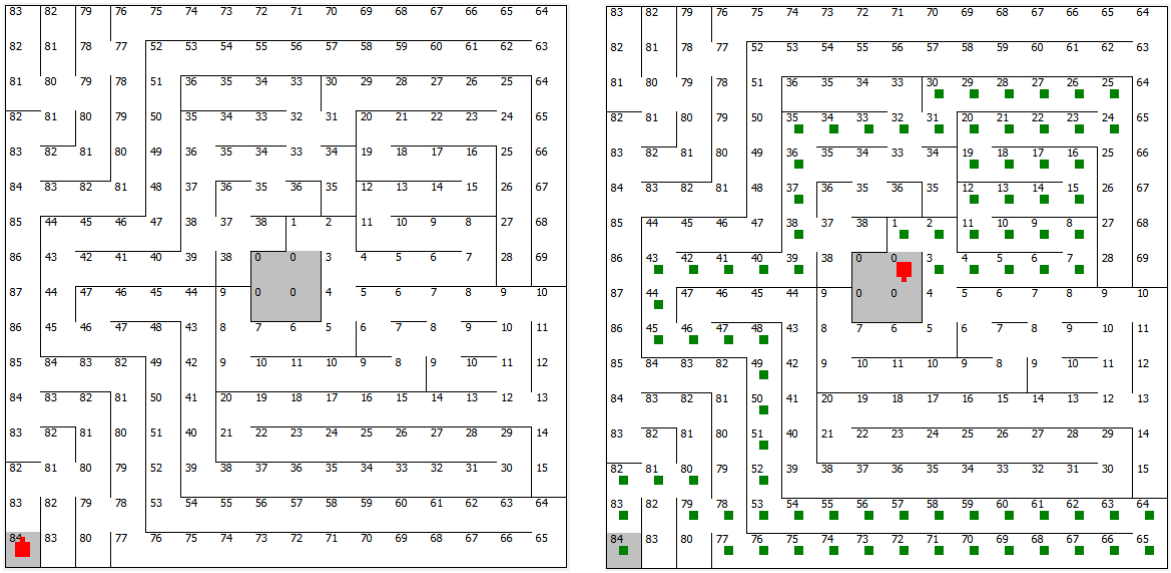
poruszał się od komórki startowej zawsze do tej o mniejszej wartości odległości od mety. Kluczową zaletą tego algorytmu jest to, że wyznaczona ścieżka jest zawsze najkrótsza. Może nie być optymalna pod względem liczby zakrętów czy szybkości przejazdu, ale na pewno nie da się dotrzeć do mety krótszą drogą. Natomiast główną wadą takiego podejścia jest to, że aby przeprowadzić zalewanie i wyznaczanie ścieżki należy przechowywać w pamięci robota kompletną mapę labiryntu, co wymaga wykonania wstępnego przejazdu przez wszystkie komórki. Stąd wniosek, że do poprawnego działania algorytmu konieczne jest, aby w labiryncie nie było niedostępnych komórek (np. mających ściany po każdej stronie), bo taka sytuacja powoduje, że algorytm nigdy nie kończy etapu tworzenia mapy. Na rysunku 3.10 umieszczono schemat blokowy algorytmu, natomiast na rysunku 3.11 znajdują się obraz przykładowego labiryntu po zalewaniu oraz końcowy wynik działania algorytmu — kompletna trasa przejazdu.

3.4.1 Zmodyfikowany algorytm zalewający

Standardowy algorytm zalewający wyznacza ścieżkę najkrótszą, ale niekoniecznie optymalną pod względem czasu przejazdu. Dlatego też popularna wśród konstruktorów jest jego modyfikacja, w której wykonuje się ważone zalewanie [15]. W tym wariancie proces również zaczyna się od mety labiryntu, ale inne są wartości wprowadzane do komórek — jeśli przejście do kolejnego pola nie wymaga skrętu to wprowadzana jest wartość o 1 większa, jeśli jednak skręt jest konieczny wprowadzona zostaje wartość o 3 większa. W ten sposób wyznaczona ścieżka będzie miała mniej zakrętów, dzięki czemu możliwe będzie, że jej pokonanie zajmie mniej czasu niż przejazd ścieżką najkrótszą. Na rysunku 3.12 przedstawiono schemat blokowy algorytmu, natomiast na rysunku 3.13 zamieszczone są obraz przykładowego labiryntu po zalewaniu oraz kompletna trasa przejazdu.

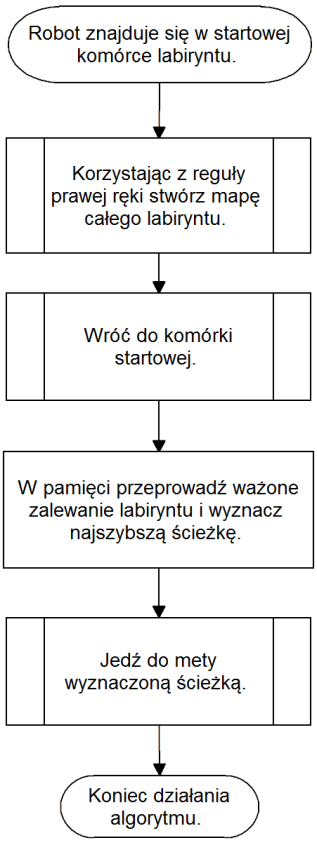
3.4.2 Przyspieszony algorytm zalewający

Przyspieszona wersja algorytmu zalewającego eliminuje czasochłonny etap tworzenia kompletnej mapy labiryntu [15]. Zamiast tego przed rozpoczęciem przejazdu każdej komórce przydzielona jest wartość równa odległości od mety w metryce taksówkowej (przy założeniu braku ścian). W kolejnych krokach robot porusza się w kierunku malejących wartości odległości, a w przypadku, gdy nie jest to możliwe wartości

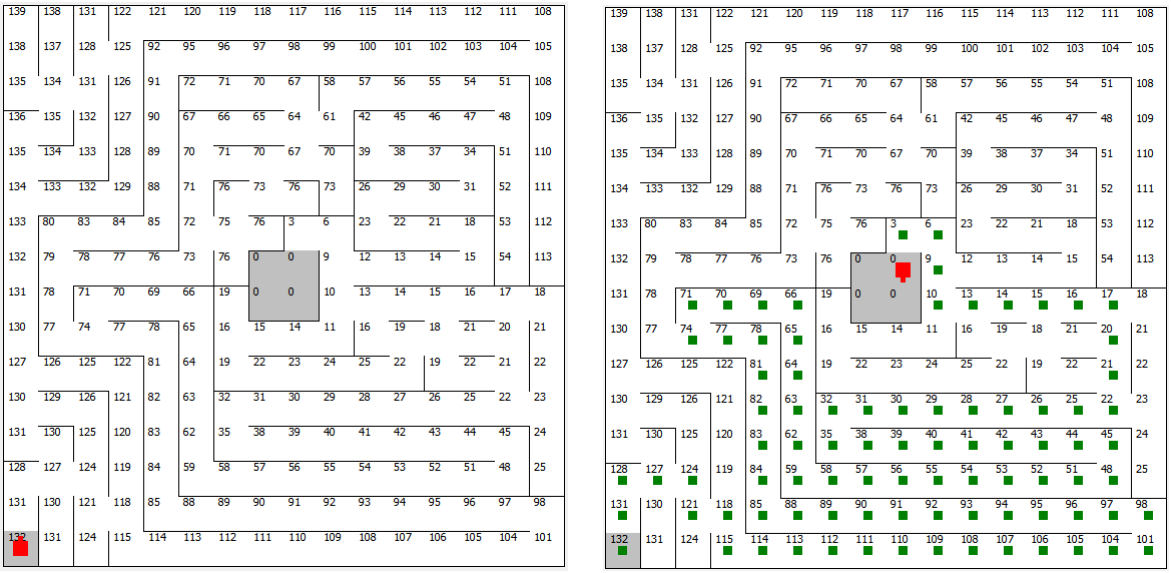


(a) Labirynt po zalewaniu (b) Wyznaczona ścieżka

Rysunek 3.11: Przykładowy labirynt po zalewaniu oraz wyznaczona ścieżka



Rysunek 3.12: Schemat blokowy zmodyfikowanego algorytmu zalewającego



(a) Labirynt po zalewaniu

(b) Wyznaczona ścieżka

Rysunek 3.13: Przykładowy labirynt po ważonym zalewaniu oraz wyznaczona ścieżka

poszczególnych pól są odpowiednio modyfikowane. Dzięki takiej zmianie liczba kroków potrzebna do znalezienia ścieżki jest znacząco mniejsza, ale algorytm traci swoją najważniejszą cechę — nie ma już gwarancji, że wyznaczona droga jest najkrótsza. Na rysunku 3.14 przedstawiono schemat blokowy algorytmu, natomiast na rysunku 3.15 zamieszczone są obraz przykładowego labiryntu po zalewaniu oraz kompletna trasa przejazdu.

3.5 Algorytmy grafowe

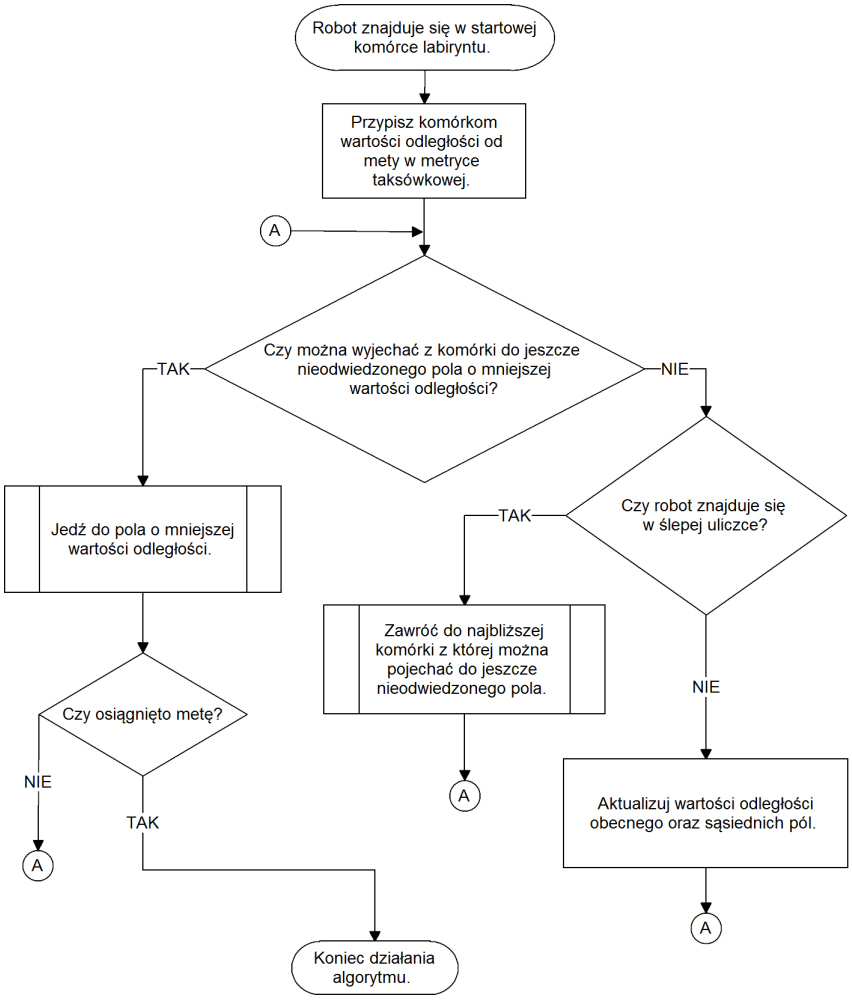
Wszystkie algorytmy z grupy grafowych, podobnie jak algorytm zalewający, potrzebują do prawidłowego działania kompletnej mapy labiryntu, wymagają zatem wykonania wstępnego przejazdu, w trakcie którego taka mapa zostanie utworzona. W parze z tym wymogiem występuje ograniczenie co do schematu labiryntu — każdy z tych algorytmów zawiedzie w sytuacji, gdy w labiryncie znajduje się choć jedna komórka, do której nie można wjechać. Przedstawienie labiryntu jako grafu realizowane jest przez założenie, że każda komórka to wierzchołek, natomiast każda droga, którą można z danej komórki wyjechać, jest krawędzią o wadze równej 1, łączącą dany wierzchołek z kolejnym. Przy takim założeniu oraz implementacji grafu w formie tzw. *listy sąsiedztwa* otrzymano odpowiadającą potrzebom strukturę reprezentującą labirynt [14]. Dalej przedstawiono cztery algorytmy grafowe, które mogą zostać wykorzystane w robotach klasy MicroMouse.

3.5.1 Algorytm Dijkstry

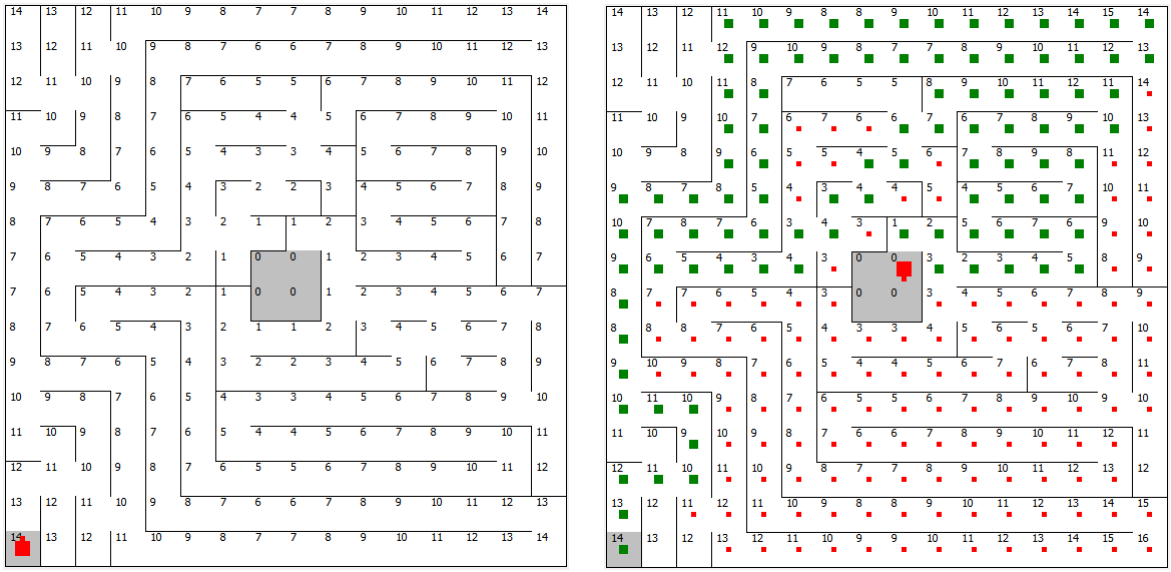
Algorytm Dijkstry jest algorytmem zachłannym przeznaczonym do wyznaczania najkrótszych ścieżek w grafie od zadanego wierzchołka źródłowego do wszystkich pozostałych wierzchołków [14, 17]. Jedynym wymaganiem do uzyskania prawidłowego rozwiązania jest, aby wszystkie krawędzie miały dodatnie wagi. Może jednak dojść do sytuacji, w której mimo niespełnienia wymogu algorytm z powodzeniem zakończy działanie, lecz wtedy nie ma gwarancji co do poprawności wyznaczonej ścieżki. Rezultatem działania algorytmu są dwie tablice: jedna przechowująca odległości od wierzchołka źródłowego do wszystkich pozostałych oraz druga przechowująca indeksy wierzchołków poprzedzających dany wierzchołek na ścieżce. Dzięki drugiej z tych tablic możliwe jest odtworzenie najkrótszej ścieżki z wierzchołka źródłowego do dowolnego innego. Na rysunku 3.16 przedstawiono schemat blokowy oraz wyznaczoną przez algorytm trasę przejazdu.

3.5.2 Algorytm Bellmana-Forda

Algorytm Bellmana-Forda stanowi ogólniejszą wersję algorytmu Dijkstry — w przeciwieństwie do niego potrafi poradzić sobie z ujemnymi wagami krawędzi, poprawne działanie uniemożliwiają mu tylko



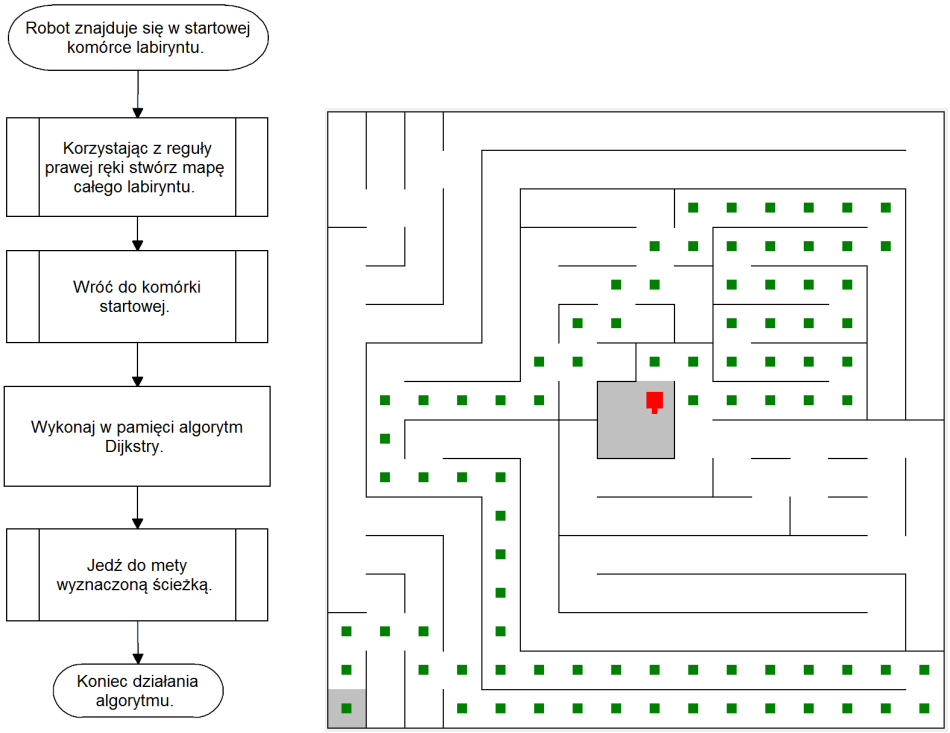
Rysunek 3.14: Schemat blokowy przyspieszonego algorytmu zalewającego



(a) Labirynt po zalewaniu

(b) Wyznaczona ścieżka

Rysunek 3.15: Przykładowy labirynt po wstępnym zalewaniu oraz wyznaczona ścieżka



(a) Schemat blokowy (b) Wyznaczona ścieżka

Rysunek 3.16: Schemat blokowy algorytmu Dijkstry oraz wyznaczona trasa przejazdu

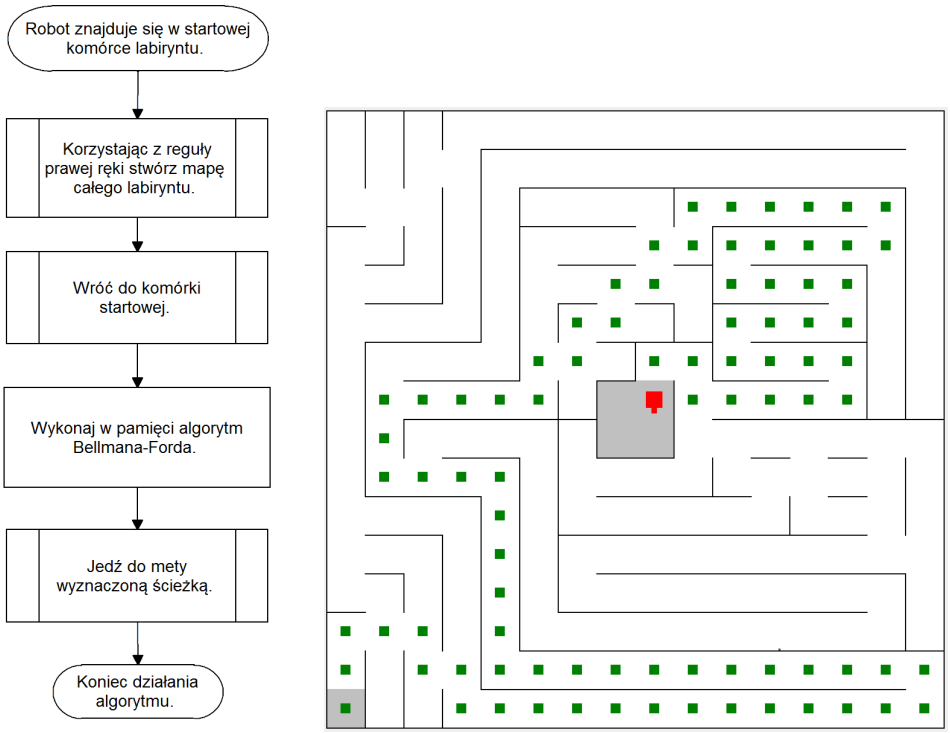
występujące w grafie cykle o ujemnej sumie wag [14, 17]. Przez bardziej ogólne podejście do problemu algorytm ten cechuje się wyższą złożonością czasową niż algorytm Dijkstry, lecz nie jest to zmiana, która uniemożliwiałaby wykorzystanie go w zawodach MicroMouse. Wynikiem działania algorytmu jest tablica odległości każdego wierzchołka od wierzchołka źródłowego oraz tablica wierzchołków poprzedzających dany wierzchołek, pozwalająca odtworzyć najkrótszą ścieżkę. Na rysunku 3.17 przedstawiono schemat blokowy oraz wyznaczoną przez algorytm trasę przejazdu.

3.5.3 Przeszukiwanie wszerz (BFS)

Algorytm BFS jest jednym z najprostszych algorytmów przeszukiwania grafu [14, 17]. Rozpoczyna on przeszukiwanie od zadanego wierzchołka i następnie odwiedza kolejno wszystkie osiągalne z niego wierzchołki. Przeznaczony jest do znajdowania elementów lub wykonywania procedur wymagających odwiedzenia każdego wierzchołka, ale może być wykorzystany również do wyznaczania najkrótszej ścieżki. Wynikiem działania algorytmu są dwie tablice. Pierwsza z nich zawiera odległości wszystkich wierzchołków od wierzchołka źródłowego, natomiast druga przechowuje informacje o wierzchołkach poprzedzających dany, co pozwala na odtworzenie najkrótszej ścieżki do dowolnego z osiągalnych ze źródła wierzchołków. Na rysunku 3.18 przedstawiono schemat blokowy oraz wyznaczoną przez algorytm trasę przejazdu.

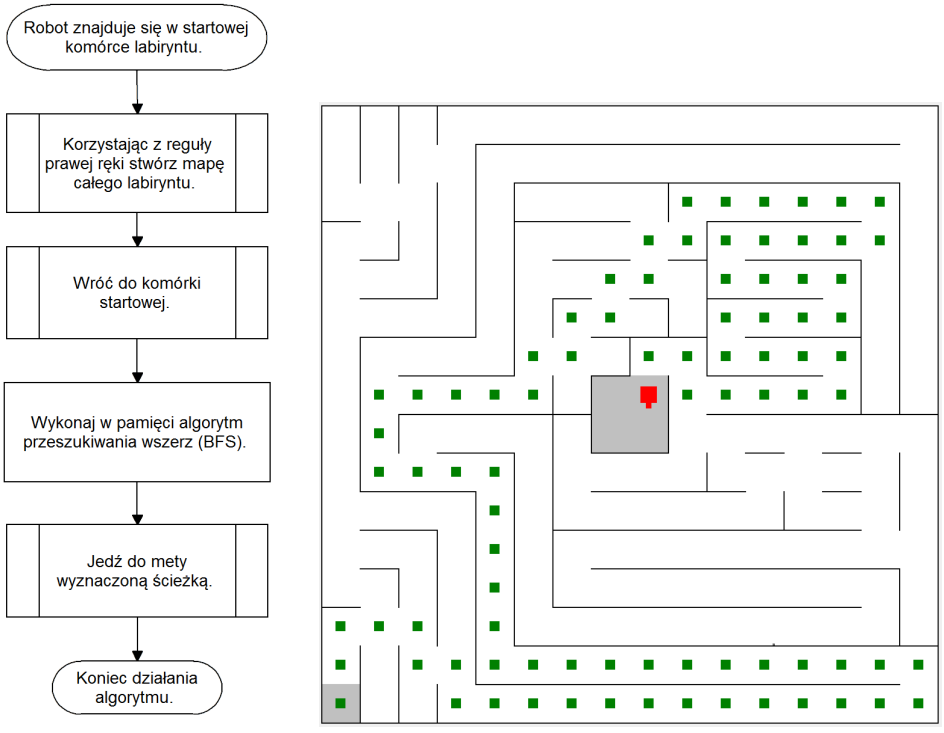
3.5.4 Przeszukiwanie w głąb (DFS)

Algorytm DFS [14, 17] wraz z algorytmem BFS stanowią dwa podstawowe algorytmy przeszukiwania grafu. Algorytm DFS również rozpoczyna przeszukiwanie od zadanego wierzchołka źródłowego i następnie odwiedza kolejne wierzchołki za każdym razem próbując wejść w graf o poziom głębiej. Podejście takie może zostać wykorzystane do wyznaczania ścieżki, ale nie gwarantuje wyznaczenia ścieżki najkrótszej. W wyniku działania algorytmu otrzymuje się tablicę z odległościami wszystkich wierzchołków od wierzchołka źródłowego oraz tablicę wskazującą wierzchołki poprzedzające dany wierzchołek. Druga z tablic pozwala na odtworzenie pierwszej osiągalnej ze źródła ścieżki do dowolnego wierzchołka. Na rysunku 3.19 przedstawiono schemat blokowy oraz wyznaczoną przez algorytm trasę przejazdu.



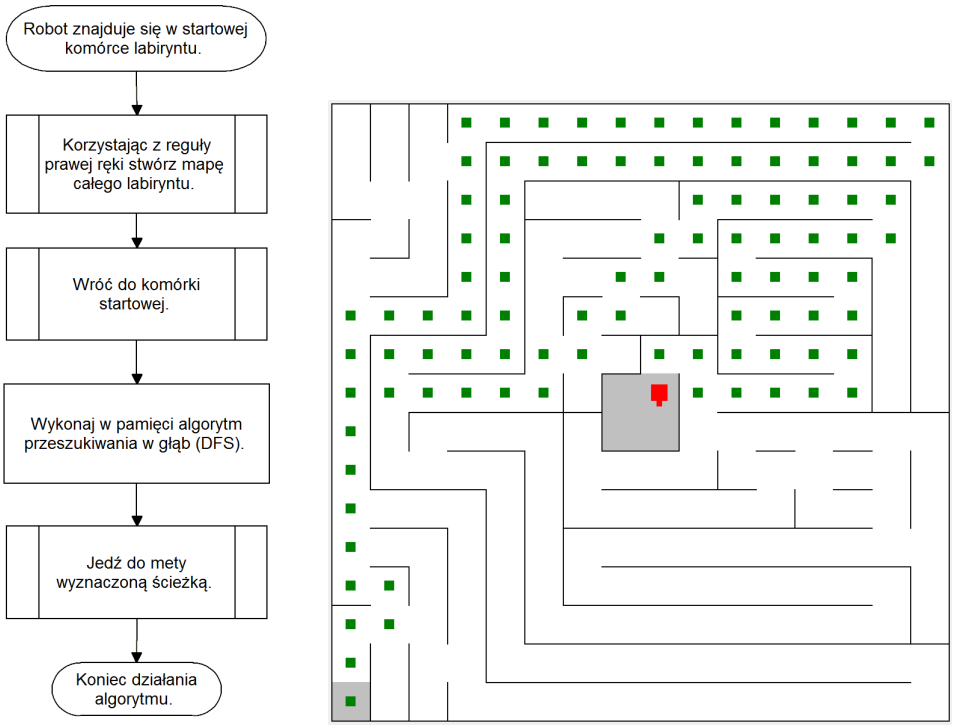
(a) Schemat blokowy (b) Wyznaczona ścieżka

Rysunek 3.17: Schemat blokowy algorytmu Bellmana-Forda oraz wyznaczona trasa przejazdu



(a) Schemat blokowy (b) Wyznaczona ścieżka

Rysunek 3.18: Schemat blokowy algorytmu BFS oraz wyznaczona trasa przejazdu



(a) Schemat blokowy (b) Wyznaczona ścieżka

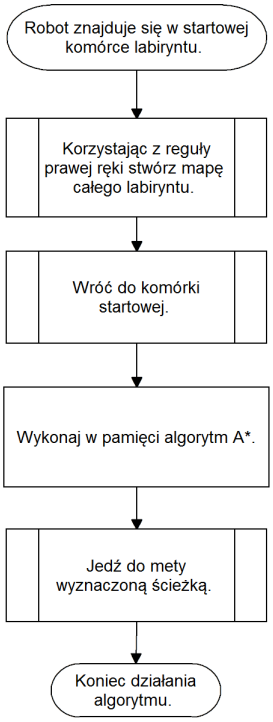
Rysunek 3.19: Schemat blokowy algorytmu DFS oraz wyznaczona trasa przejazdu

3.6 Algorytm A*

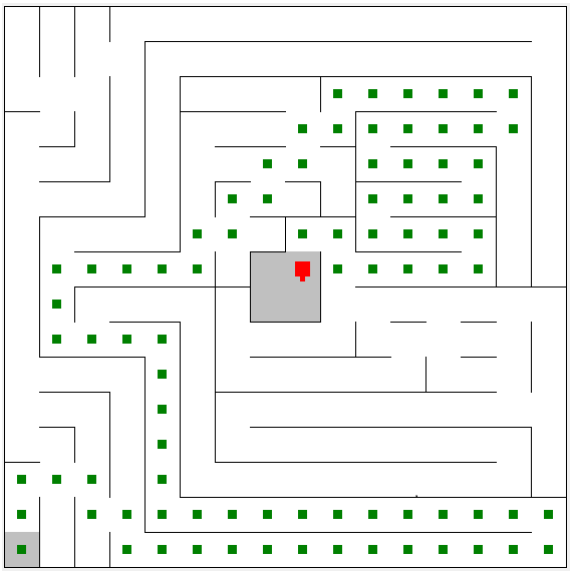
Algorytm A* [17] jest heurystycznym algorytmem poszukiwania najkrótszej ścieżki w grafie ważonym. Umożliwia wyznaczenie ścieżki między wybranym wierzchołkiem źródłowym, a wierzchołkiem spełniającym określony warunek (tzw. *test celu*). Algorytm ten jest zupełny i optymalny, co oznacza, że zawsze wyznacza ścieżkę najkrótszą, o ile taka istnieje. Podobnie jak algorytmy grafowe do poprawnego działania wymaga uprzedniego utworzenia kompletnej mapy labiryntu. Kluczowy jest również dobór odpowiedniej funkcji heurystycznej odpowiedzialnej za podejmowane przez algorytm decyzje. Zaproponowano dwie takie funkcje, z których obie pozwalają na wyznaczenie ścieżki najkrótszej:

1. Odległość od mety liczona w metryce taksówkowej, przy założeniu braku ścian.
2. Odległość od mety liczona w linii prostej.

Schemat blokowy algorytmu przedstawiono na rysunku 3.20, natomiast trasę przejazdu wyznaczoną przez obie funkcje heurystyczne na rysunku 3.21.



Rysunek 3.20: Schemat blokowy dla algorytmu A*



Rysunek 3.21: Trasa przejazdu dla obu funkcji heurystycznych w algorytmie A*

Rozdział 4

Badania symulacyjne

W niniejszym rozdziale znajduje się opis kryteriów oceny algorytmów, analiza ich zachowania na podstawie obserwacji działania w pięciu różnych labiryntach oraz zestawienie osiąganych przez nie wyników.

4.1 Środowisko symulacyjne

Badane algorytmy oceniono na podstawie obserwacji efektów ich wykorzystania w aplikacji symulującej działanie robota umieszczonego w labiryncie. Aplikacja ta zapewnia możliwość wglądu w najważniejsze parametry dotyczące skuteczności algorytmu oraz jego przydatności dla robota mobilnego klasy Micro-Mouse. Szczegółowy opis aplikacji wraz z instrukcją obsługi znajduje się w dodatku A.

4.2 Kryteria oceny

Jako główne kryteria oceny przyjęto:

- liczbę kroków wykonanych do momentu rozwiązania labiryntu,
- liczbę odwiedzonych przez robota komórek labiryntu,
- długość wyznaczonej ścieżki,
- liczba zakrętów w wyznaczonej ścieżce.

Dodatkowo przyjęto bardziej ogólne kryteria pomocnicze pozwalające wspólnie zestawić wyniki osiągnięte w różnych labiryntach:

- procent odwiedzonych komórek,
- stosunek długości wyznaczonej ścieżki do długości ścieżki najkrótszej,
- stosunek liczby zakrętów w wyznaczonej ścieżce do liczby zakrętów w ścieżce z ich najmniejszą liczbą,
- stosunek liczby kroków do długości najkrótszej ścieżki.

4.3 Wyniki

4.3.1 Algorytm losowy

Algorytm losowy potrafił rozwiązać każdy z labiryntów, jednak liczba kroków, długość wyznaczonej ścieżki i pozostałe parametry mocno się wahały, dlatego dla uzyskania wiarygodnych wyników algorytm ten został przetestowany po dziesięć razy w każdym labiryncie i wyniki przedstawione w tabeli 4.1 są wartościami średnimi uzyskanymi w tych przejazdach.

Tabela. 4.1: Średnie wyniki działania algorytmu losowego

Labirynt	Liczba kroków	Odwiedzone komórki	Długość ścieżki	Liczba zakrętów
Labirynt 1	240	167	102	34
Labirynt 2	278	178	85	30
Labirynt 3	202	145	97	49
Labirynt 4	282	164	65	25
Labirynt 5	282	174	85	45

Tabela. 4.2: Wyniki działania algorytmu lewej ręki

Labirynt	Liczba kroków	Odwiedzone komórki	Długość ścieżki	Liczba zakrętów
Labirynt 1	232	201	brak	-
Labirynt 2	90	87	brak	-
Labirynt 3	97	88	83	49
Labirynt 4	59	35	brak	-
Labirynt 5	64	57	brak	-

4.3.2 Algorytm lewej/prawej ręki

Zgodnie z przewidywaniami algorytm lewej/prawej ręki nie potrafił rozwiązać wszystkich labiryntów — poradził sobie tylko z jednym. W pozostałych czterech labiryntach algorytm kończył działanie przed znalezieniem drogi do mety, dla tych labiryntów uwzględniono tylko liczbę wykonanych kroków i odwiedzonych komórek. Wyniki działania algorytmu lewej/prawej ręki przedstawiono odpowiednio w tabelach 4.2 oraz 4.3.

Zmodyfikowany algorytm lewej/prawej ręki
Zmodyfikowany algorytm lewej/prawej ręki miał być udoskonaleniem standardowej wersji algorytmu, a okazał się od niego gorszy i nie potrafił rozwiązać żadnego z pięciu dostępnych labiryntów. Wyniki działania zmodyfikowanego algorytmu lewej/prawej ręki przedstawiono odpowiednio w tabelach 4.4 oraz 4.5.

4.3.3 Metoda pól potencjałów

Metoda pól potencjałów pozwalała wyznaczyć ścieżkę w każdym labiryncie, jednak nie były to ścieżki najkrótsze. Atutem tego algorytmu jest na pewno stosunkowo mała liczba kroków potrzebnych do wyznaczenia rozwiązania bliskiego najkrótszej ścieżce. Wyniki osiągane przez algorytm przedstawiono w tabeli 4.6.

4.3.4 Algorytmy zalewające

Algorytm zalewający wykonywał zawsze zdecydowanie większą liczbę kroków niż wcześniej badane algorytmy, a wynikało to z faktu, że wymaga on na wejściu pełnej mapy labiryntu w związku z czym wykonywał na wstępie dodatkowy przejazd przez cały labirynt mający na celu stworzenie takiej mapy. Korzyścią płynącą z takiego podejścia był fakt, że wyznaczona trasa zawsze była najkrótsza. Szczegółowe wyniki przedstawiono w tabeli 4.7.

Zmodyfikowany algorytm zalewający
Modyfikacja polegająca na wykonywaniu ważonego zalewania celem minimalizacji liczby zakrętów miała wyznaczać ścieżkę potencjalnie najszybszą zamiast najkrótszej. W czterech z pięciu labiryntów wyznaczona ścieżka faktycznie miała najmniejszą liczbę zakrętów i nie była dużo dłuższa od najkrótszej, lecz w jednym z labiryntów algorytm ten wyznaczył dokładnie taką ścieżkę jak zwykły algorytm zalewający. Wynikało to z faktu, że proces zalewania oparty jest na relaksacji i może dojść do sytuacji w której konkretna komórka labiryntu wstępnie określona jako wymagająca skrętu (wpisana wyższa wartość) w późniejszym etapie zalewania będzie sprawdzana pod kątem jazdy prosto i zostanie w nią wpisana wartość niższa. Przy odpowiednim ułożeniu całego labiryntu zjawisko to może sprowadzić ważone zalewanie do standardowego zalewania na przestrzeni całego labiryntu lub tylko jego fragmentu. Konsekwencją takiego zdarzenia jest to, że wyznaczona ścieżka nie będzie miała minimalnej liczby zakrętów, wciąż nie ma

Tabela. 4.3: Wyniki działania algorytmu prawej ręki

Labirynt	Liczba kroków	Odwiedzone komórki	Długość ścieżki	Liczba zakrętów
Labirynt 1	234	200	brak	-
Labirynt 2	90	87	brak	-
Labirynt 3	97	88	83	25
Labirynt 4	59	35	brak	-
Labirynt 5	64	57	brak	-

Tabela. 4.4: Wyniki działania zmodyfikowanego algorytmu lewej ręki

Labirynt	Liczba kroków	Odwiedzone komórki	Długość ścieżki	Liczba zakrętów
Labirynt 1	11	11	brak	-
Labirynt 2	73	73	brak	-
Labirynt 3	110	104	brak	-
Labirynt 4	102	77	brak	-
Labirynt 5	24	22	brak	-

Tabela. 4.5: Wyniki działania zmodyfikowanego algorytmu prawej ręki

Labirynt	Liczba kroków	Odwiedzone komórki	Długość ścieżki	Liczba zakrętów
Labirynt 1	119	103	brak	-
Labirynt 2	58	58	brak	-
Labirynt 3	46	44	brak	-
Labirynt 4	54	42	brak	-
Labirynt 5	56	52	brak	-

Tabela. 4.6: Wyniki działania metody pól potencjałów

Labirynt	Liczba kroków	Odwiedzone komórki	Długość ścieżki	Liczba zakrętów
Labirynt 1	374	230	99	36
Labirynt 2	174	126	82	32
Labirynt 3	96	88	83	39
Labirynt 4	258	163	62	25
Labirynt 5	52	47	43	20

Tabela. 4.7: Wyniki działania algorytmu zalewającego

Labirynt	Liczba kroków	Odwiedzone komórki	Długość ścieżki	Liczba zakrętów
Labirynt 1	619	256	85	28
Labirynt 2	567	256	48	20
Labirynt 3	611	256	73	51
Labirynt 4	597	256	50	25
Labirynt 5	584	256	39	24

Tabela. 4.8: Wyniki działania zmodyfikowanego algorytmu zalewającego

Labirynt	Liczba kroków	Odwiedzone komórki	Długość ścieżki	Liczba zakrętów
Labirynt 1	621	256	87	22
Labirynt 2	571	256	52	16
Labirynt 3	613	256	75	47
Labirynt 4	599	256	52	19
Labirynt 5	588	256	43	18

Tabela. 4.9: Wyniki działania przyspieszonego algorytmu zalewającego

Labirynt	Liczba kroków	Odwiedzone komórki	Długość ścieżki	Liczba zakrętów
Labirynt 1	368	227	99	36
Labirynt 2	174	126	82	32
Labirynt 3	96	88	83	39
Labirynt 4	259	163	62	25
Labirynt 5	52	47	43	20

też gwarancji, że będzie to ścieżka najkrótsza. Wyniki działania zmodyfikowanego algorytmu zalewającego przedstawiono w tabeli 4.8.

Przyspieszony algorytm zalewający
Przyspieszony algorytm zalewający jako jedyny spośród zalewających nie potrzebuje na wejściu kompletnej mapy labiryntu, dzięki czemu liczba wykonywanych przez niego kroków jest dużo mniejsza niż w dwóch pozostałych przypadkach. Zmniejszeniu ulega również liczba odwiedzonych komórek. Oczywistym jest, że z powodu gorszej znajomości labiryntu wyznaczona ścieżka jest dłuższa niż w zwykłym algorytmie zalewającym. Ogólna zasada działania algorytmu sprawia, że w rezultatach zauważalne jest podobieństwo do metody pól potencjałów. Szczegółowe wyniki przedstawiono w tabeli 4.9.

4.3.5 Algorytmy grafowe

Podstawową wadą każdego z algorytmów grafowych jest to, że do działania potrzebują kompletnej mapy labiryntu na wejściu, przez co wykonywana przez nie liczba kroków jest zdecydowanie większa niż w przypadku algorytmów podejmujących decyzje na bieżąco w trakcie eksploracji labiryntu. Trzy spośród czterech algorytmów grafowych potrafiły zawsze wyznaczyć ścieżkę najkrótszą. Wyjątkiem był algorytm przeszukiwania w głąb, który wyznaczał ścieżkę dłuższą. Takie działanie wynika z faktu, że podczas przeszukiwania w głąb wybierane jest pierwsze napotkane rozwiązanie, które niekoniecznie jest rozwiązaniem najkrótszym. Wyniki dla algorytmów Dijkstry, Bellmana-Forda oraz przeszukiwania wszerz przedstawiono w tabeli 4.10, natomiast algorytmu przeszukiwania w głąb w tabeli 4.11.

4.3.6 Algorytmy A*

Działanie algorytmu heurystycznego jest uzależnione od doboru funkcji heurystycznej na podstawie której podejmowane będą decyzje. Wyniki były identyczne dla obu proponowanych funkcji oceniających — wyznaczana była trasa najkrótsza. Wadą algorytmu, podobnie jak w przypadku algorytmów grafowych, jest konieczność wykonania wstępnego przejazdu celem stworzenia mapy labiryntu, który powoduje znaczny wzrost liczby wykonanych kroków. Szczegółowe wyniki przedstawiono w tabeli 4.12.

4.4 Zestawienie wyników

W trakcie analizy własności algorytmów wyraźnie rysuje się podział na dwie grupy

- algorytmy podejmujące decyzję na podstawie bieżącej znajomości labiryntu,
- algorytmy wymagające na wejściu pełnej mapy labiryntu.

Zauważalne są też podstawowe własności obu tych grup. Algorytmy z pierwszej wykonują mniejszą liczbę kroków, ale nie potrafią wyznaczyć rozwiązania optymalnego, w zasadzie niektóre z nich w ogóle nie potrafią wyznaczyć ścieżki do mety. Algorytmy z drugiej grupy zawsze wyznaczają ścieżkę i w większości

Tabela. 4.10: Wyniki działania algorytmów grafowych z wyjątkiem przeszukiwania w głąb

Labirynt	Liczba kroków	Odwiedzone komórki	Długość ścieżki	Liczba zakrętów
Labirynt 1	619	256	85	28
Labirynt 2	567	256	48	20
Labirynt 3	611	256	73	51
Labirynt 4	597	256	50	25
Labirynt 5	584	256	39	24

Tabela. 4.11: Wyniki działania algorytmu przeszukiwania w głąb

Labirynt	Liczba kroków	Odwiedzone komórki	Długość ścieżki	Liczba zakrętów
Labirynt 1	631	256	97	36
Labirynt 2	569	256	50	20
Labirynt 3	635	256	97	55
Labirynt 4	615	256	68	25
Labirynt 5	610	256	65	27

Tabela. 4.12: Wyniki działania algorytmu A*

Labirynt	Liczba kroków	Odwiedzone komórki	Długość ścieżki	Liczba zakrętów
Labirynt 1	619	256	85	34
Labirynt 2	567	256	48	20
Labirynt 3	611	256	73	51
Labirynt 4	597	256	50	25
Labirynt 5	584	256	39	24

przypadków jest to ścieżka najkrótsza. Ich poważną wadą jest jednak fakt, że jeśli którakolwiek z komórek będzie niedostępna to algorytm bardzo szybko zakończy swoje działanie nie wyznaczywszy ścieżki.

Aby możliwe było sensowne porównanie rezultatów z różnych labiryntów wykorzystano pomocnicze kryteria oceny. Zestawienie wyników algorytmów pierwszej grupy przedstawiono w tabeli 4.13, natomiast algorytmów drugiej grupy w tabeli 4.14. Przy wglądzie w tabelę 4.13 należy pamiętać, że algorytm lewej/prawej ręki rozwiązał tylko jeden labirynt, natomiast jego modyfikacja nie rozwiązała żadnego. W tabelach 4.13 oraz 4.14 wykorzystano następujące oznaczenia algorytmów:

- **Los.** — algorytm losowy,
- **Lewa ręka** — algorytm lewej ręki,
- **Prawa ręka** — algorytm prawej ręki,
- **Lewa mod.** — zmodyfikowany algorytm lewej ręki,
- **Prawa mod.** — zmodyfikowany algorytm prawej ręki,
- **MSPP** — metoda pól potencjałów,
- **Zal. przys.** — przyspieszony algorytm zalewający,
- **Zal.** — algorytm zalewający,
- **Zal. mod.** — zmodyfikowany algorytm zalewający,
- **Dijkstra** — algorytm Dijkstry,
- **B-F** — algorytm Bellmana-Forda,
- **BFS** — przeszukiwanie wszerz,
- **DFS** — przeszukiwanie w głąb,
- **A*** — algorytm A*.

Wykorzystane w omawianych tabelach kryteria oceny algorytmów zostały opisane w sekcji 4.2.

Tabela. 4.13: Wyniki działania algorytmów decydujących na podstawie bieżącej wiedzy o labiryncie

Algorytm	Odw. komórki [%]	Dł. ścieżki / Dł. najkrótszej ścieżki	L. zakretów / Min. l. zakrętów	L. kroków / Dł. najkrótszej ścieżki
Los.	64,7	1,56	1,84	4,85
Lewa ręka	36,6	1,14	1,00	1,33
Prawa ręka	39,2	1,14	1,88	1,25
Lewa mod.	22,4	-	-	-
Prawa mod.	23,4	-	-	-
MSPP	51,1	1,27	1,53	3,17
Zal. przys.	50,9	1,27	1,53	3,16

Tabela. 4.14: Wyniki działania algorytmów wymagających na wejściu kompletnej mapy labiryntu

Algorytm	Odw. komórki [%]	Dł. ścieżki / Dł. najkrótszej ścieżki	L. zakretów / Min. l. zakrętów	L. kroków / Dł. najkrótszej ścieżki
Zal.	100	1,00	1,44	10,87
Zal. mod.	100	1,05	1,18	10,93
Dijkstra	100	1,00	1,44	10,87
B-F	100	1,00	1,44	10,87
BFS	100	1,00	1,44	10,87
DFS	100	1,31	1,58	11,18
A*	100	1,00	1,44	10,87

W przypadku wyboru strategii polegającej na podejmowaniu decyzji na bieżąco podczas eksploracji labiryntu najlepszym okazuje się przyspieszony algorytm zalewający. Z praktycznie identycznymi wynikami, zaraz za nim plasuje się metoda pól potencjałów. Złym wyborem są algorytm lewej/prawej ręki oraz jego modyfikacja — ich atutem jest prostota, jednak w przeprowadzonych testach wypadły bardzo słabo. Jeśli konstruktor zdecyduje się na algorytm prosty w implementacji to zdecydowanie lepszym wyborem będzie algorytm losowy, który przede wszystkim gwarantuje rozwiązania, a dodatkowo średnie wyniki jego działania są naprawdę zadowalające.

W przypadku wyboru strategii opartej na tworzeniu kompletnej mapy labiryntu na początku działania algorytmu jest kilka algorytmów o identycznych rezultatach osiąganych zbliżonym kosztem — algorytmy zalewający, Dijkstry, Bellmana-Forda oraz przeszukiwanie wszerz wyznaczają ścieżkę najkrótszą. Najgorszym spośród tych algorytmów jest przeszukiwania w głąb, które prowadziło robota drogą nawet o 30% dłuższą. Ciekawą opcją jest zmodyfikowany algorytm zalewający, który wyznaczał ścieżki minimalnie dłuższe od najkrótszych, a w czterech na pięć przypadków wyraźnie zmniejszył liczbę zakrętów, co potencjalnie może znacząco poprawić czas przejazdu rzeczywistego robota.

Rozdział 5

Wnioski

Celem pracy było ustalenie, który z popularnych algorytmów proponowanych do poszukiwania ścieżki w labiryncie jest najbardziej odpowiedni dla robota mobilnego klasy MicroMouse. Niemożliwym jednak okazało się jednoznaczne określenie najbardziej odpowiedniego algorytmu, gdyż w trakcie badań symulacyjnych wyklarował się wyraźny podział algorytmów na dwie klasy

- algorytmy podejmujące decyzję na podstawie bieżącej znajomości labiryntu,
- algorytmy wymagające na wejściu pełnej mapy labiryntu.

Udało się natomiast uszeregować algorytmy w ramach tych klas.

Na podstawie wyników widać, że jeśli konstruktor chce mieć pewność wyznaczenia najkrótszej ścieżki powinien zdecydować się na algorytmy z drugiej klasy, konkretnie algorytm zalewający, Dijkstry, Bellmana-Forda, przeszukiwanie wszerz lub algorytm A*. Liczba wykonywanych kroków, a co za tym idzie czas działania całego algorytmu, jest wtedy bardzo długi, ale gwarantowana jest poprawność rozwiązania. W sytuacji, gdy robot porusza się wolno (np. potrafi wykonywać obrót tylko po zatrzymaniu) lepsze okażą się algorytmy z pierwszej klasy, np. przyspieszony algorytm zalewający, metoda pól potencjałów czy nawet banalny w implementacji algorytm losowy. Wykorzystanie jednego z tych algorytmów również gwarantuje znalezienie rozwiązania i eliminuje czasochłonny proces tworzenia kompletnej mapy labiryntu. Natomiast bez względu na założone cele należy unikać algorytmu lewej/prawej ręki i jego modyfikacji, gdyż żaden z nich nie zapewnia wyznaczenia ścieżki do mety.

W trakcie badań symulacyjnych na jaw wyszły słabości poszczególnych algorytmów, wiele z nich jest jednak możliwych do poprawienia.

Problem 1: Algorytm lewej/prawej ręki i jego modyfikacja nie potrafią znaleźć ścieżki w niektórych labiryntach.

Rozwiązaniem jest odpowiednia modyfikacja obu algorytmów, tak aby wjazd do komórki, która nie jest ślepą uliczką, ale nie ma nieodwiedzonego sąsiada nie kończył działania algorytmu, tylko traktowany był jak ślepa uliczka i powodował powrót do najbliższej komórki z której można pojechać do jeszcze nieodwiedzonego pola. Sprawi to, że każdy labirynt będzie możliwy do rozwiązania.

Problem 2: Przyspieszony algorytm zalewający nie wyznacza najkrótszej ścieżki.

W zaimplementowanej wersji algorytmu po wyborze złej trasy następuje aktualizacja wartości danej komórki i jej sąsiadów. Rozszerzenie fragmentu labiryntu objętego aktualizacją lub wprowadzenie w takiej sytuacji ponownego zalewania może poprawić wyniki osiągane przez algorytm. Pozytywny wpływ może też mieć wykonanie po pierwszym dotarciu do mety przejazdu w kierunku startu. W ciągu jednego lub kilku takich przejazdów możliwe będzie zoptymalizowanie trasy i skrócenie długości wyznaczonej ścieżki.

Problem 3: Algorytmy wymagające na wejściu kompletnej mapy labiryntu wykonują łącznie bardzo dużą liczbę kroków.

Zaimplementowany algorytm wykonywania kompletnego przejazdu polega na eksploracji labiryntu zgodnie z zasadą prawej ręki, aż do momentu, gdy wszystkie komórki zostaną odwiedzone. Wtedy następuje powrót na pole startowe, wyznaczenie ścieżki i przejazd prosto do mety. Możliwe, że zmiana sposobu przejazdu zmniejszy liczbę wykonywanych kroków.

Problem 4: Algorytmy wymagające na wejściu kompletnej mapy labiryntu nie mogą działać poprawnie, gdy choć jedna komórka labiryntu jest niedostępna.

Wśród rozważanych labiryntów nie było takiego przypadku, więc problem ten został pominięty, ale trzeba liczyć się z możliwością jego wystąpienia. Aby uodpornić algorytm na taką sytuację należy zaimplementować określanie niedostępnych obszarów labiryntu.

Problem 5: Zmodyfikowany algorytm zalewający w jednym z pięciu labiryntów nie potrafił wyznaczyć ścieżki o najmniejszej liczbie zakrętów.

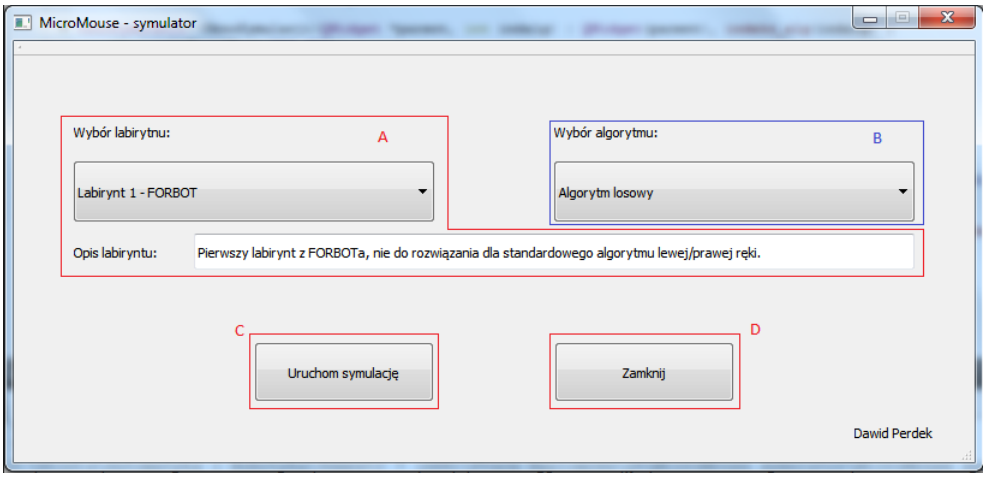
Wyeliminowanie tego problemu wymagałoby przechowywania w trakcie zalewania potencjalnych ścieżek, tak aby w trakcie działania algorytm miał dostęp do wszystkich wariantów przejazdu przez daną komórkę. Takie rozwiązanie pozwoliłoby odtwarzać ścieżki i wartości komórek przez które przebiega, a dzięki temu finalnie wybrać ścieżkę spełniającą oczekiwania.

Planowane jest zaimplementowanie wymienionych poprawek oraz uzupełnienie przeprowadzonych w ramach pracy badań o testy z wykorzystaniem rzeczywistego robota, dzięki czemu kryteria oceny algorytmów zostaną poszerzone o faktyczne czasy rozwiązania labiryntu. Następnie planuje się przygotowanie robota do jazdy po przekątnych i zbadanie wpływu wykorzystania tej możliwości na osiągane rezultaty.

Dodatek A

Instrukcja użytkowania aplikacji

Badanie własności algorytmów przeprowadzono z wykorzystaniem autorskiej aplikacji symulującej robota w labiryncie napisanej w języku *C++* w oparciu o bibliotekę *Qt*. Aplikacja umożliwia wybór labiryntu oraz jednego z badanych algorytmów i przeprowadzenie symulacji. Dostępnych jest pięć różnych labiryntów i czternaście wariantów algorytmu poszukującego rozwiązania. Możliwe jest równoległe działanie wielu okien symulacji o różnych parametrach. Na rysunku A.1 przedstawiono okno główne aplikacji.



Rysunek A.1: Główne okno aplikacji

Kolejnymi literami oznaczono:

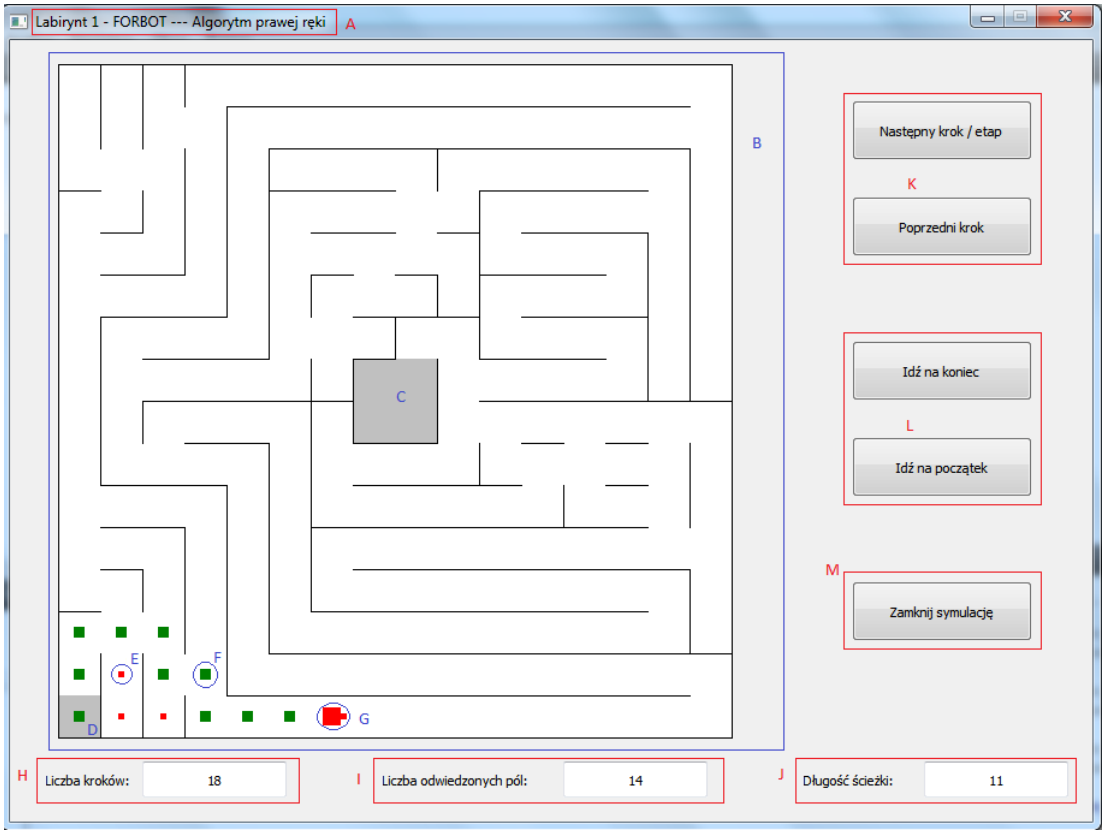
A — rozwijane menu pozwalające na wybór labiryntu oraz pole zawierające krótki opis obecnie wybranego labiryntu,

B — rozwijane menu pozwalające na wybór badanego algorytmu,

C — przycisk powodujący uruchomienie okna symulacji wybranego algorytmu w wybranym labiryncie,

D — przycisk kończący działanie całej aplikacji i zamykający jej wszystkie okna.

Po wybraniu żądanych parametrów i uruchomieniu symulacji ukazuje się nowe okno z jednym z trzech wariantów interfejsu użytkownika, zależnie od badanego algorytmu. Pierwszy z nich to tryb pracy krokowej, w którym możliwe jest obserwowanie poczynań robota krok po kroku. Jako krok traktowany jest przejazd prosto, skręt w lewo, skręt w prawo oraz manewr zawracania wewnątrz jednej komórki. W dowolnym momencie możliwe są również skok do końca działania algorytmu, jak i powrót do początkowego stanu symulacji. Tryb ten jest wykorzystywany przy symulowaniu algorytmów na bieżąco tworzących mapę labiryntu, analizujących położenie robota i na tej podstawie podejmujących dalsze działania. Widok omawianego okienka przedstawiony jest na rysunku A.2.



Rysunek A.2: Okno symulacji — widok pracy krokowej

Kolejnymi literami oznaczono:

A — tytuł okna symulacji zawierający nazwę aktywnego labiryntu oraz badanego algorytmu,

B — obszar, w którym wyświetlany jest labirynt, robot oraz przebyta przez niego droga,

C — pole docelowe labiryntu,

D — pole startowe labiryntu,

E — sposób zaznaczania pól odrzuconych przez algorytm,

F — sposób zaznaczania pól traktowanych przez algorytm jako docelowa ścieżka,

G — sposób zaznaczania robota, kwadratowa wypustka reprezentuje jego przód,

H — wskaźnik liczby wykonanych w obecnej symulacji kroków,

I — wskaźnik liczby odwiedzonych przez robota komórek labiryntu,

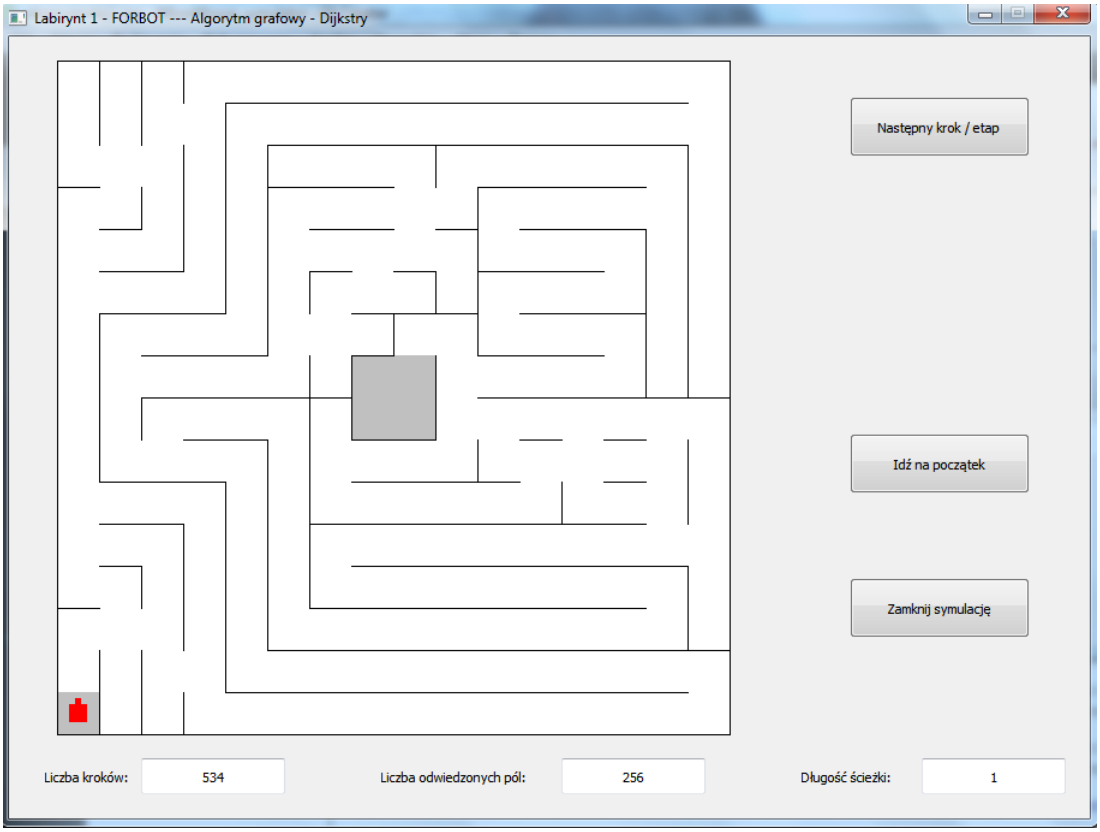
J — wskaźnik długości wyznaczonej przez algorytm ścieżki,

K — przyciski pozwalające na posunięcie symulacji o krok w przód oraz w tył,

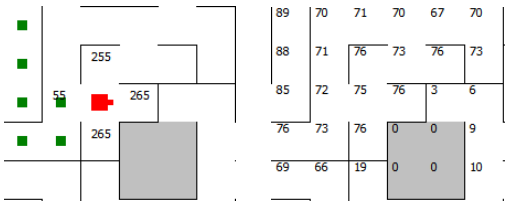
L — przyciski pozwalające na skok do końcowego rezultatu działania algorytmu oraz powrót do początkowego stanu symulacji,

M — przycisk zamykający dane okno symulacji.

Drugim wariantem działania okna symulacji jest tryb pracy etapowej. Jako etap rozumiany jest ciąg kroków prowadzący do osiągnięcia pewnego pośredniego celu, np. przejazdu całej wyznaczonej ścieżki. Tryb ten wykorzystywany jest przy badaniu algorytmów wymagających do poprawnego działania kompletnej mapy labiryntu — dzięki takiemu podejściu użytkownik nie musi samodzielnie przechodzić



Rysunek A.3: Okno symulacji — widok pracy etapowej



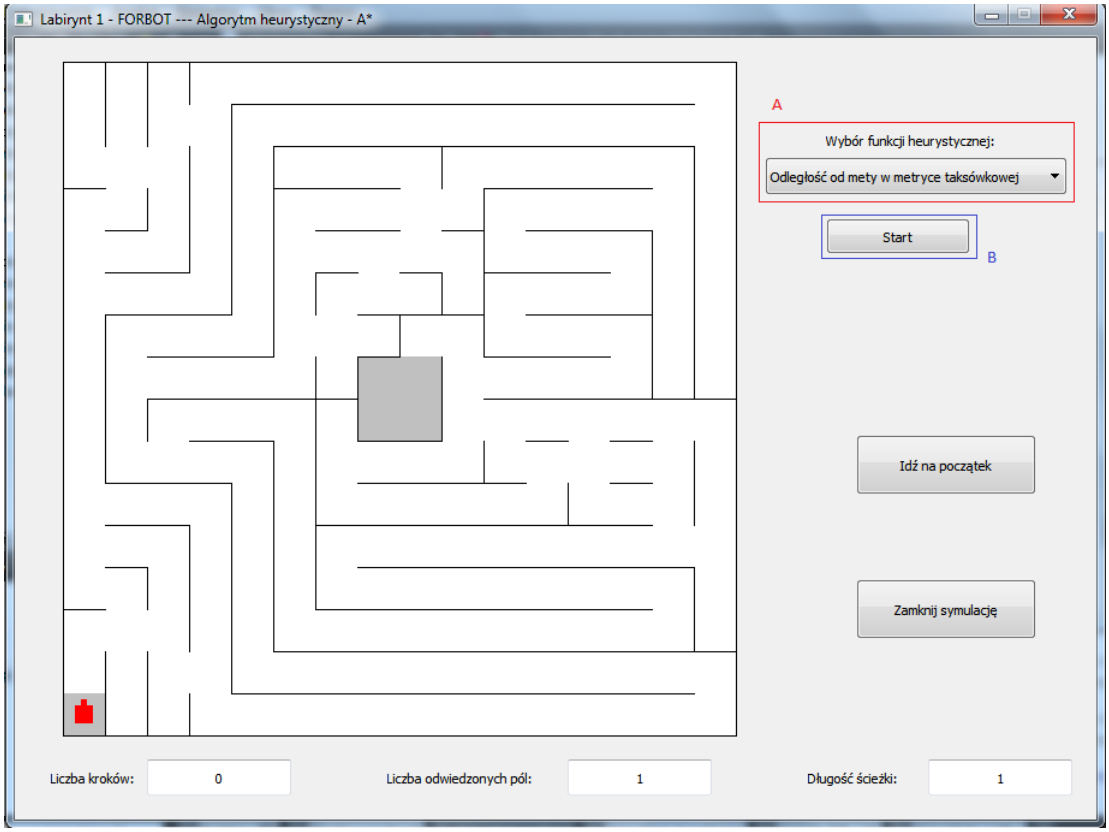
(a) Pierwszy sposób (b) Drugi sposób

Rysunek A.4: Dwa sposoby wyświetlania wartości komórek

przez żmudny proces odwiedzania każdej komórki. Jedyna różnica w interfejsie polega na niedostępności przycisków pozwalających na cofnięcie przebiegu symulacji o krok oraz na przeskok do końca działania algorytmu. Przycisk powodujący posunięcie symulacji o krok w trybie krokowym, w trybie etapowym powoduje przeskok do kolejnego etapu algorytmu, natomiast przycisk odpowiadający za przywrócenie początkowego stanu symulacji zachowuje swoje działanie. Widok okna symulacji dla trybu etapowego przedstawiony jest na rysunku A.3.

Zarówno w trybie krokowym, jak i etapowym możliwe jest pojawianie się różnych wartości liczbowych w poszczególnych komórkach labiryntu. Dzieje się tak w przypadku badania algorytmów, które decyzje podejmują na podstawie wyliczenia wartości konkretnej funkcji przyporządkowującej współczynniki komórkom. Przykłady wyświetlania takich wartości przedstawione są na rysunku A.4.

Trzeci wariant wyglądu okna symulacji jest dedykowany dla algorytmów heurystycznych, przy których istotny jest dobór funkcji heurystycznej z której będą korzystać — okno jest przystosowane w taki sposób, aby to umożliwić. Widok ten jest dostępny tylko do momentu wyboru tejże funkcji, następnie symulacja przechodzi w tryb etapowy. W omawianym widoku niedostępny jest również przycisk odpowiadający za przejście do następnego etapu, a w jego miejsca pojawia się obszar wyboru i zatwierdzenia funkcji heurystycznej. Na rysunku A.5 przedstawiono omawiany widok.



Rysunek A.5: Okno symulacji — początkowy widok dla algorytmów heurystycznych

Kolejnymi literami oznaczono:

A — obszar pozwalający na wybór funkcji heurystycznej,

B — przycisk zatwierdzający wybór i przenoszący symulację do trybu etapowego.

Bibliografia

[1] Devil — opis robota klasy MicroMouse konstrukcji użytkownika *grabo* z forum *FORBOT*. http://www.forbot.pl/forum/topics7/micromouse-devil-vt6542.htm.

[2] *Festiwal Robotów ROBOCOMP*. http://www.robocomp.info/.

[3] *Festiwal Robotyki Cyberbot*. http://www.sumo.put.poznan.pl/.

[4] Lista opisów robotów klasy MicroMouse tworzonych przez użytkowników forum *FORBOT*. http://www.forbot.pl/forum/robots_list.php?tag=Micromouse.

[5] Opis metod sztucznych pól potencjałów na stronie dr Krzysztofa Arenta. http://rab.ict.pwr.wroc.pl/~arent/rr/mspp/algorytm.html.

[6] Przykład labiryntu wykorzystywanego w zawodach MicroMouse. http://www.forbot.pl/forum/topics20/micromouse-metody-przeszukiwania-labiryntu-vt2246.htm.

[7] Regulamin konkurencji MicroMouse z *Festiwalu Robotyki Cyberbot*. http://cyberbot.put.poznan.pl/wp-content/themes/cyberbot/downloads/MicroMouse.pdf.

[8] Regulamin konkurencji MicroMouse z *Festiwalu Robotyki ROBOCOMP*. http://www.robocomp.info/konkurencje/regulamin/micromouse/.

[9] Regulamin konkurencji MicroMouse z *Mistrzostw Polski Robotów*. http://mistrzostwapolskirobotow.pl/files/MicroMouse_ver_1.0.pdf.

[10] Regulamin konkurencji MicroMouse z zawodów *Robotic Arena*. http://konar.pwr.edu.pl/index.php/robotic-arena.

[11] Zawody *Robomaticon*. http://www.robomaticon.pl/.

[12] Zawody *Robotic Arena*. http://konar.pwr.edu.pl/index.php/robotic-arena.

[13] Jianping Cai, Xuting Wan, Meimei Huo, and Jianzhong Wu. An algorithm of micromouse maze solving. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 1995–2000, 2010.

[14] Adam Drozdek. *C++ Algorytmy i struktury danych*. HELION, 2004.

[15] Użytkownik forum *FORBOT marcin13021988*. Micromouse — metody przeszukiwania labiryntu. Dział *"Artykuły"* forum *FORBOT*. http://www.forbot.pl/forum/topics20/micromouse-metody-przeszukiwania-labiryntu-vt2246.htm, 2009.

[16] M. Garbacz and M. Zaczyk. Metoda pól potencjałowych w nawigacji kołowego robota mobilnego. *Automatyka / Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie*, T. 15, z. 3:339–347, 2011.

[17] G. Heineman, G. Pollice, and S. Selkow. *Algorytmy. Almanach*. HELION, 2010.

[18] S. Mishra and P. Bande. Maze solving algorithms for micro mouse. In *Signal Image Technology and Internet Based Systems, 2008. SITIS '08. IEEE International Conference on*, pages 86–93, 2008.

[19] Wikipedia. Krótki opis zawodów MicroMouse. `https://pl.wikipedia.org/wiki/Micromouse`.

[20] Wikipedia. Labirynt jako budowla. `http://pl.wikipedia.org/wiki/Labirynt`.

[21] L. Wyard-Scott and Q.-H.M. Meng. A potential maze solving algorithm for a micromouse robot. In *Communications, Computers, and Signal Processing, 1995. Proceedings., IEEE Pacific Rim Conference on*, pages 614–618, 1995.

Spis rysunków

Spis tabel