Politechnika Wrocławska

Wydział Elektroniki, Fotoniki i Mikrosystemów

KIERUNEK: Automatyka i Robotyka (AIR)

PRACA DYPLOMOWA INZYNIERSKA

TYTUŁ PRACY:

Optymalizacja trasy przejazdu robota klasy micromouse

AUTOR:

Jakub Michalski

PROMOTOR:

Dr inż. Robert Muszyński,
Katedra Cybernetyki i Robotyki

WROCŁAW 2024

Spis treści

Rozdział 1

Wstęp

Labirynty od wieków fascynowały ludzkość, zarówno jako symboliczne struktury, jak i praktyczne elementy architektury. Już w starożytności były one obecne w mitach i legendach, jak słynny labirynt na Krecie zbudowany przez Dedala. Współczesne interpretacje labiryntów łączą ich bogate dziedzictwo kulturowe z nowoczesną nauką i technologią, przekształcając je w przestrzeń dla eksperymentów z zakresu algorytmów, nawigacji i sztucznej inteligencji. Labirynty dla robotów mobilnych klasy micromouse są właśnie przestrzenią, w której testowane są ich algorytmy nawigacyjne, zdolności jezdne oraz efektywność podejmowania decyzji w nieznanym środowisku. Za pierwszego robota klasy micromouse uważa się „Tezeusza" autorstwa Claude'a Shannona [8], stworzył on swoją „myszkę" w 1950 roku. Ponad 20 lat później, w 1977 przeprowadzona została pierwsza pełnoprawna konkurencja micromouse zorganizowana przez IEEE Spectrum Magazine [1]. Po tym wydarzeniu zawody, w których mogły mierzyć się konstrukcje robotów micromouse zostały spopularyzowane i organizowane są w wielu zakątkach świata. We Wrocławiu konkurencja odbywa się na wydarzeniu Robotic Arena [15], gdzie można obejrzeć zmagania innych konstrukcji lub przetestować własne.

Celem niniejszej pracy jest rozwinięcie tematu badania labiryntu przez autonomiczne roboty mobilne klasy micromouse oraz sposoby, w jakie wyznaczają one ścieżki, chcąc minimalizować czas przejazdu. Przetestowane zostaną metody, w których robot wyznacza wiele tras przejazdu ze startu do mety, które następnie są testowane w środowisku symulacyjnym wraz ze sprawdzeniem wpływu jazdy po skosie w pozwalających na to fragmentach labiryntu.

Układ pracy jest następujący. W rozdziale drugim przedstawione zostały podstawowe informacje o robotach klasy micromouse, zasady konkurencji, w których uczestniczą oraz zadania z jakimi muszą się zmierzyć. Trzeci rozdział opisuje algorytmy nawigacyjne z których korzystają roboty, algorytmy związane z poszukiwaniem optymalnych ścieżek i metody umożliwiające optymalizację ruchu. W rozdziale czwartym przeprowadzone zostały symulacyjne przejazdy przez przykładowe labirynty ukazujące własności algorytmów opisanych w rozdziale trzecim. W ostatnim rozdziale znajduje się podsumowanie całości pracy.

Rozdział 2

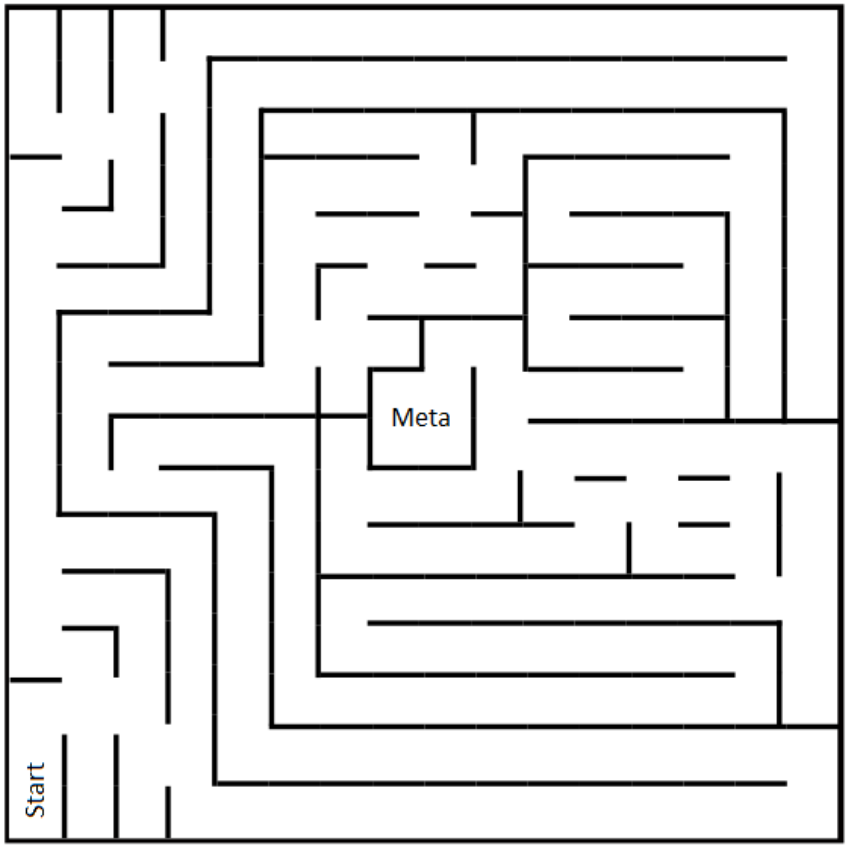
Roboty klasy micromouse

Micromouse to autonomiczny robot mobilny zaprojektowany specjalnie do pokonywania labiryntów. W tym celu jest on wyposażony w kilka kluczowych komponentów, które wspierają jego funkcje eksploracyjne i nawigacyjne. Podstawowe elementy robota [14] obejmują silniki, jednostkę sterującą oraz zestaw czujników, takich jak enkodery, czujniki odległości, żyroskop, czy akcelerometr. Dzięki temu wyposażeniu robot klasy micromouse może precyzyjnie nawigować po labiryncie, co umożliwia mu eksplorację oraz tworzenie mapy. Na podstawie zgromadzonych danych robot wyznacza optymalną trasę, co umożliwia mu szybkie dotarcie do celu. W tym rozdziale zostaną przedstawione podstawowe informacje o robotach klasy micromouse, zasady obowiązujące na zawodach, zadania z jakimi muszą się zmierzyć, zadanie sterowania robotem oraz zadanie rozwiązania labiryntu.

2.1 Konkurencja micromouse

Konkurencja micromouse w Polsce odbywa się na takich wydarzeniach jak Robotic Arena we Wrocławiu [15], czy też EastROBO w Białymstoku [6], na których obowiązują podobne zasady [7, 16]. W klasycznym wydaniu konkurencja rozgrywana jest w labiryncie w kształcie kwadratu składającego się z 256 pól o wymiarach 18 na 18 cm. Pola mogą być oddzielone od siebie ścianką o wysokości 5 cm. Eksploracja labiryntu rozpoczynana jest od wybranego pola znajdującego się w jednym z jego wierzchołków, a celem robota jest dotarcie do 4 nieoddzielonych od siebie ściankami pól w formacie kwadratu, do którego prowadzi tylko jedno wejście. Przykładowy labirynt przedstawiono na rysunku 2.1.

O zwycięstwie decyduje czas w jakim labirynt został rozwiązany. Najbardziej istotny będzie przejazd, w którym robot dotrze ze startu do mety w najkrótszym czasie. Z kolei czas jaki zajmie mu eksploracja labiryntu jest dużo mniej istotny i jest on przemnażany przez odpowiednią wagę np. $\frac{1}{60}$. Do finalnego czasu mogą być doliczane kary wynikające z wymaganej pomocy robotowi w razie wypadków takich jak zgubienie się robota w labiryncie, przewrócenie się, czy też wjechanie w ściankę.



Rysunek 2.1: Przykładowy labirynt

2.2 Konstrukcja robota

Konstrukcja robota micromouse jest dostosowana do wymogów zawodów. Aby osiągnąć optymalną wydajność, roboty tego typu muszą być kompaktowe, zwrotne oraz wyposażone w odpowiednie systemy napędu i czujników, umożliwiające precyzyjną nawigację. W [5] zostało opisane jak w prosty sposób stworzyć swojego micromouse'a. Na rysunku 2.2 pokazano przykładową konstrukcję robota.

Kształt robota zwykle jest kompaktowy, co pozwala mu na łatwe manewrowanie w ograniczonej przestrzeni labiryntu. Najczęściej stosuje się konstrukcje o symetrycznym kształcie (na przykład prostokątne lub okrągłe), co umożliwia równomierne rozłożenie masy i lepszą stabilność podczas szybkich ruchów. Standardowe wymiary robota wynoszą zazwyczaj około 10×10 cm, co pozwala mu na bezproblemowe wykonywanie ruchów w polach labiryntu.

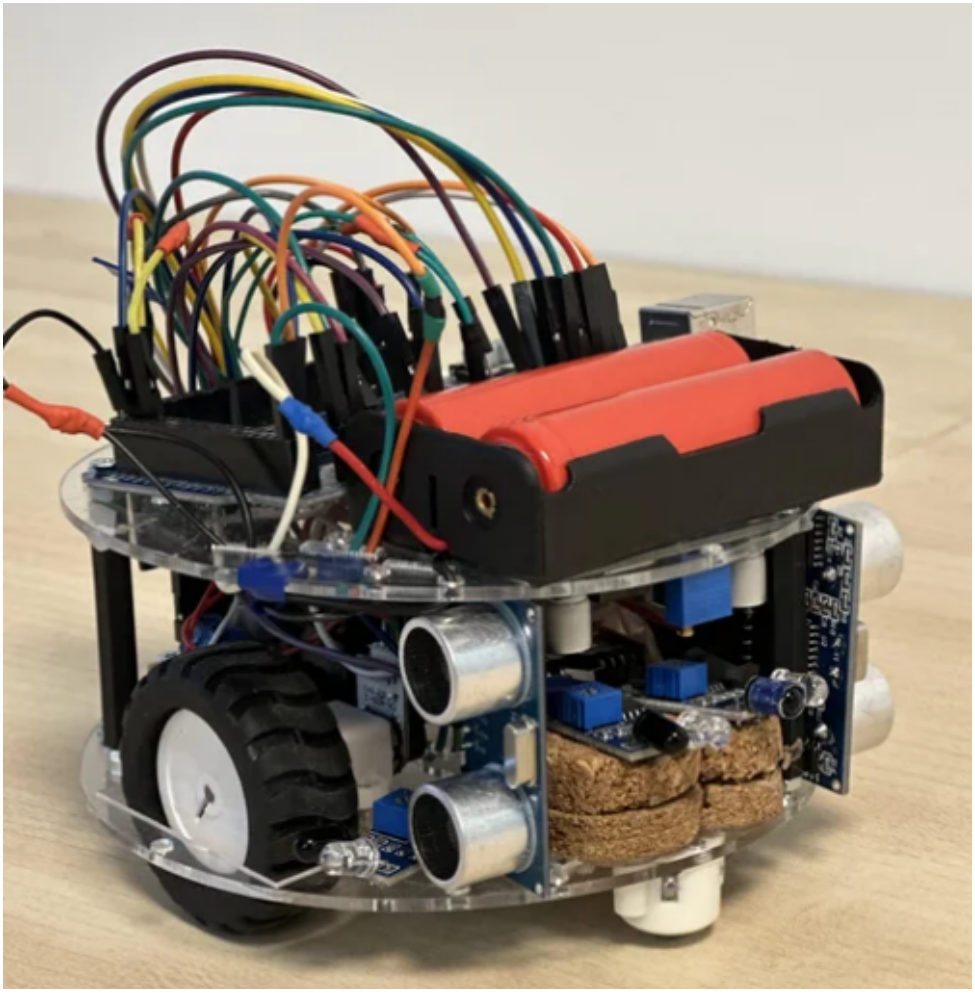
Jako napęd zazwyczaj wykorzystywane są silniki DC z przekładniami [14] ze względu na ich prostą konstrukcję, szeroką dostępność oraz łatwość sterowania. Zastosowanie przekładni zapewnia wysoki moment obrotowy co pozwala na dynamiczne przyspieszanie i hamowanie. Silniki DC mogą być sterowane za pomocą sygnału PWM, co umożliwia płynną regulację prędkości.

Robot musi być wyposażony w jednostkę sterującą oraz system zasilania. Jednost-

ka sterująca, zazwyczaj mikroprocesor lub mikrokontroler, realizuje zadanie przetwarzania danych z czujników, podejmowania decyzji nawigacyjnych oraz kontroli pracy silników. Zasilanie, najczęściej w formie akumulatora litowo-polimerowego, dostarcza energii niezbędnej do pracy wszystkich podzespołów.

Oczujnikowanie robota micromouse odgrywa kluczową rolę w jego nawigacji i eksploracji labiryntu, umożliwiając dokładne zbieranie informacji o otoczeniu oraz precyzyjne podejmowanie decyzji [14]. Wśród stosowanych czujników znajdują się:

- enkodery – monitorują obrót kół, pozwalając na precyzyjne określenie przebytej drogi i kontrolę prędkości,
- czujniki odległości – mierzą odległość od przeszkód, co umożliwia robotowi unikanie kolizji oraz rozpoznawanie gdzie znajdują się ścianki,
- żyroskopy – rejestrują zmiany w orientacji robota, co pozwala na utrzymanie stabilności i kontrolę kierunku ruchu,
- akcelerometry – mierzą przyspieszenie robota, co dostarcza informacji o zmianach prędkości i dynamice ruchu, wspierając precyzyjną nawigację.



Rysunek 2.2: Przykładowy wygląd robota stworzonego według poradnika [5]

2.3 Zadanie sterowania

Sterowanie robota micromouse zazwyczaj odbywa się na dwóch poziomach: niskim oraz wysokim, które współpracują ze sobą, aby zapewnić efektywną nawigację i eksplorację labiryntu. Oprogramowanie niskopoziomowe robota micromouse służy do zarządzania podstawowymi funkcjami robota korzystając z czujników, w które jest wyposażony. Jego celem jest zapewnienie precyzyjnego reagowania na otoczenie oraz bieżące monitorowanie stanu robota. Oprogramowanie to realizuje kluczowe zadania, takie jak kontrola prędkości oraz wykrywanie przeszkód. Dzięki tym funkcjom robot potrafi dostosować swoje zachowanie do zmieniających się warunków.

Oprogramowanie wysokopoziomowe robota micromouse zajmuje się podejmowaniem decyzji na podstawie zebranych danych, co pozwala na optymalne planowanie trasy. Robot analizuje otoczenie, aby wybrać najbardziej efektywną ścieżkę do celu, biorąc pod uwagę różne czynniki, takie jak długość składających się na nią odcinków, czy liczbę zakrętów. Przykładowo gdy robot zbliża się do długiej prostej, ma możliwość zwiększenia prędkości, co skraca czas przejazdu. Gdy robot napotyka fragment trasy złożony z serii naprzemiennych zakrętów, zaawansowane algorytmy umożliwiają mu ich płynne pokonywanie. Dzięki temu może korzystać z jazdy po skosie, co pozwala na efektywne pokonanie złożonych fragmentów labiryntu. Takie podejście do planowania ruchu znacząco zwiększa efektywność i prędkość robota w labiryncie.

2.4 Rozwiązywanie labiryntu

Zadanie rozwiązywania labiryntu można podzielić na dwie główne części: eksplorację oraz planowanie i optymalizację trasy. W fazie eksploracji robot bada strukturę labiryntu, zbierając dane o przeszkodach i możliwych trasach, co umożliwia stworzenie mapy otoczenia. W drugiej części, dzięki zebranym informacjom, robot testuje różne ścieżki, aby znaleźć tę optymalną, pozwalającą na minimalizację czasu przejazdu.

2.4.1 Eksploracja labiryntu

W fazie eksploracji robot zostaje umieszczony w jednym z narożników labiryntu, a jego głównym zadaniem jest przemieszczenie się oraz sukcesywne rozpoznawanie i zapamiętywanie struktury otoczenia. Najprostszą metodą realizacji tego zadania jest podążanie wzdłuż lewej lub prawej ściany labiryntu. Ta technika sprawdza się w prostszych labiryntach, umożliwiając robotowi systematyczne badanie korytarzy i zapamiętywanie układu ścian bez konieczności skomplikowanych obliczeń. Bardziej zaawansowaną metodą jest algorytm zalewowy (z ang. flood-fill algorithm). Oba algorytmy zostały szerzej opisane w 3. Innymi algorytmami związanymi z badaniem labiryntu są: algorytm losowy, algorytm Depth-First Search czy Breadth-First Search.

2.4.2 Optymalizacja trasy przejazdu

Po zakończeniu części eksploracyjnej robot wraca do punktu startowego. Teraz jego zadaniem jest wyznaczenie trasy, która zapewni najkrótszy czas przejazdu przez labirynt. Najprostszym kryterium optymalizacji trasy jest minimalizacja liczby pól do pokonania między startem a metą, co oznacza znalezienie najkrótszej trasy. Jednak taka trasa, choć wydaje się idealna, nie zawsze prowadzi do najkrótszego czasu przejazdu.

Robot może wybrać trasę, która jest nieco dłuższa pod względem liczby pól, ale pozwala na znaczną redukcję liczby zakrętów. Ograniczając liczbę nagłych zmian kierunku, robot ma możliwość utrzymania wyższej prędkości, co zwiększa efektywność pokonywania trasy. Dzięki takiemu podejściu, mimo większej liczby pól do pokonania, całkowity czas przejazdu może być krótszy, gdyż robot nie traci cennego czasu na manewrowanie.

Ponadto, w sytuacjach gdy trasa składa się z długich prostych, robot może przyspieszyć i wykorzystać pełną moc swojego napędu. Podobnie, gdy na trasie znajduje się sekwencja krótkich naprzemiennych zakrętów, robot może zastosować technikę jazdy po skosie, pokonując tę sekcję w jednym płynnym ruchu. Takie rozwiązania nie tylko zmniejszają liczbę zakrętów na trasie, ale również umożliwiają bardziej efektywne wykorzystanie przyspieszenia, co z kolei prowadzi do skrócenia całkowitego czasu przejazdu.

Rozdział 3

Algorytmy nawigacyjne robotów klasy micromouse

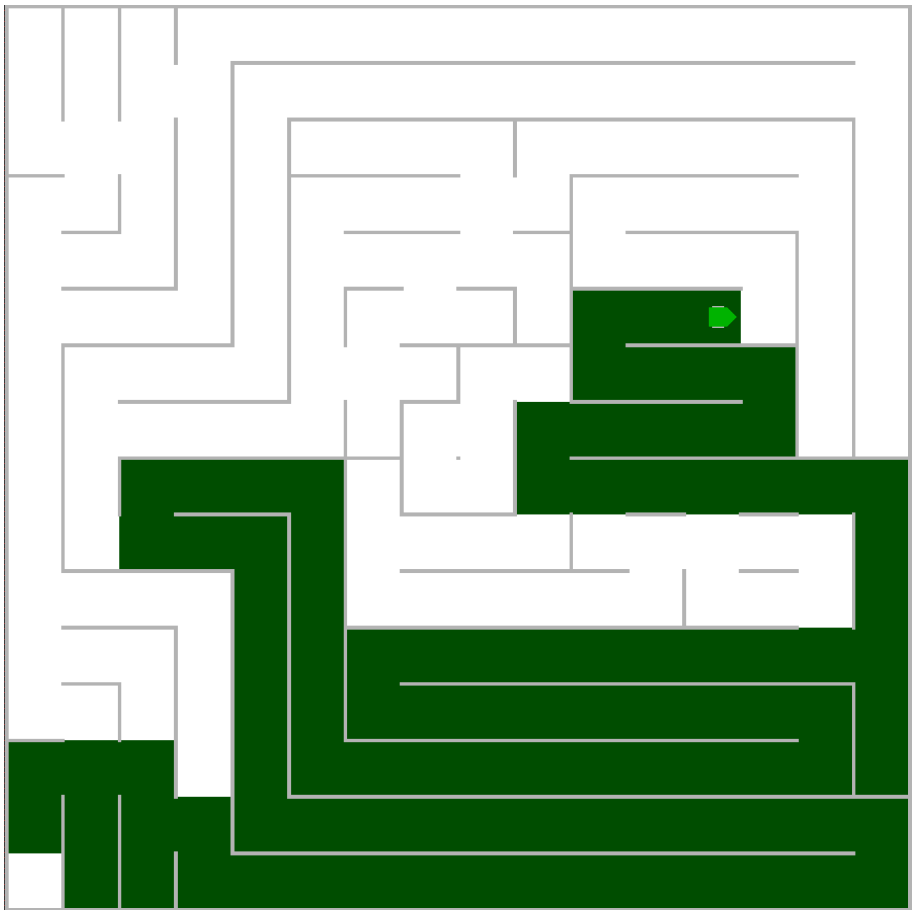
Roboty klasy micromouse korzystają z szeregu zaawansowanych algorytmów, które umożliwiają im zarówno eksplorację labiryntu, jak i znalezienie optymalnej trasy do celu. Te algorytmy pełnią kluczową rolę w procesie nawigacji, ponieważ pozwalają robotowi podejmować szybkie i precyzyjne decyzje, minimalizując czas przejazdu przez labirynt. Algorytmy związane z eksploracją umożliwią zapamiętanie struktury labiryntu, aby potem robot mógl zaplanować trasę i następnie jak najszybciej ją przejechać. Dzięki zastosowaniu algorytmów przeszukiwania, optymalizacji oraz nowoczesnych technik obliczeniowych, roboty te potrafią efektywnie pokonywać nawet najbardziej skomplikowane labirynty.

W tym rozdziale omówione zostaną główne algorytmy wykorzystywane przez roboty klasy micromouse, które pozwalają im skutecznie eksplorować labirynt, unikać pułapek oraz wyznaczać optymalną trasę. Przedstawione algorytmy obejmują zarówno metody przeszukiwania, takie jak algorytm ścienny czy algorytm zalewowy, które są opisane w [12, 10] oraz algorytmy optymalizujące czas przejazdu. Działanie tych algorytmów zostanie zilustrowane na przykładzie labiryntu z rysunku 2.1.

3.1 Algorytmy eksploracji

Podczas eksploracji labiryntu robot klasy micromouse korzysta z algorytmów, które umożliwiają mu sprawne przemieszczanie się oraz identyfikację dostępnych ścieżek [12, 10]. Kluczowym zadaniem tych algorytmów jest zapewnienie, że robot przejedzie przez nieznany labirynt w sposób uporządkowany, jednocześnie gromadząc informacje o jego strukturze.

Najprostszym rozwiązaniem jest zastosowanie algorytmu prawej/lewej ręki, który pozwala robotowi systematycznie „trzymać się" jednej ze ścian labiryntu. Algorytm ten ma swoje ograniczenia, dlatego większość robotów do przeszukania labiryntu korzysta z innych, bardziej zaawansowanych algorytmów, takich jak algorytm zalewowy, czy też algorytm grafowy DFS. Oba te algorytmy mają bardziej zaawansowane podejście do przeszukiwania, co umożliwia skuteczne zbadanie całej struktury labiryntu.

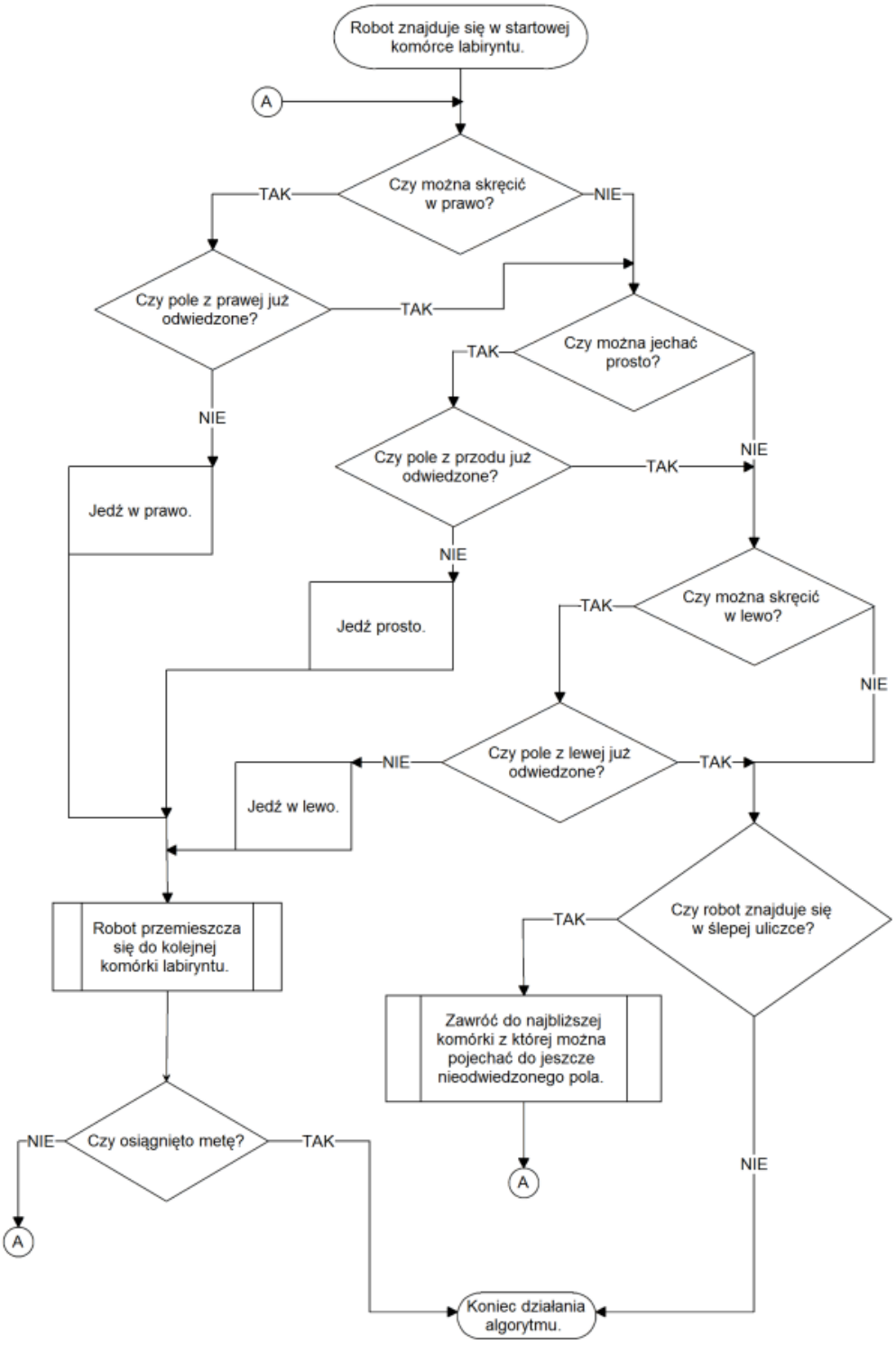


Rysunek 3.1: Przykładowa trasa robota wykorzystującego algorytm prawej ręki

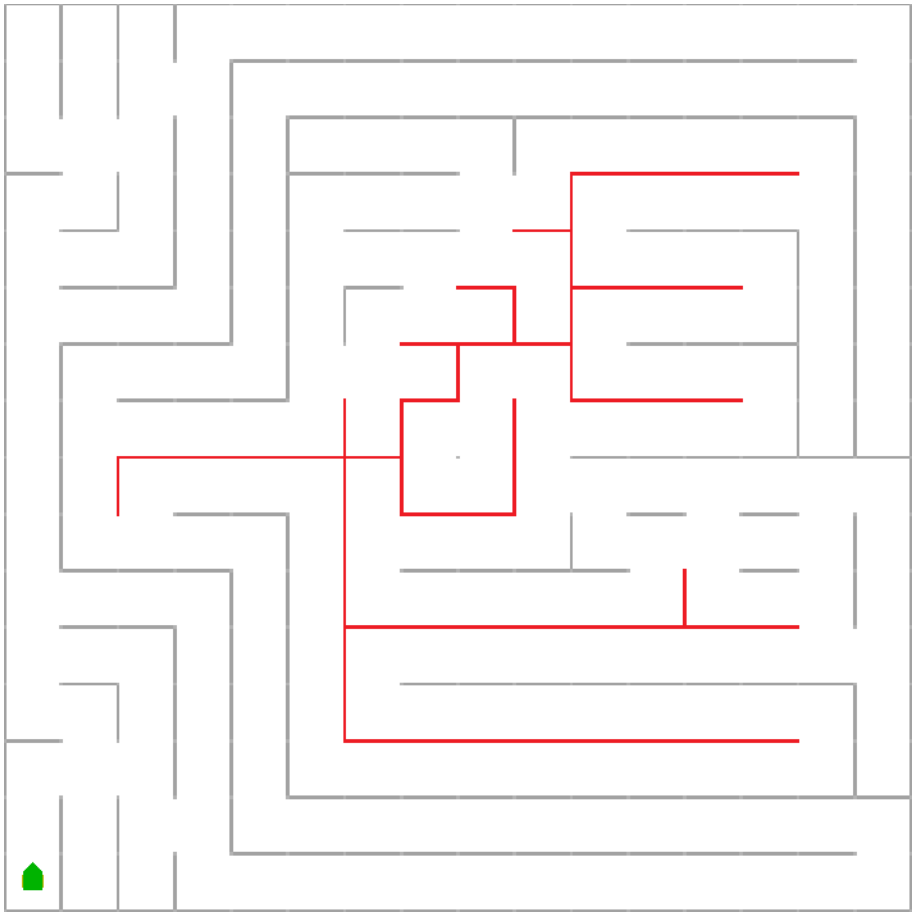
3.1.1 Algorytm prawej/lewej ręki

Algorytm prawej lub lewej ręki [10] to jedna z najprostszych metod rozwiązywania labiryntów. Jego zasada działania opiera się na systematycznym podążaniu wzdłuż jednej wybranej ściany — prawej lub lewej — w zależności od wersji algorytmu. Robot korzystający z tej metody porusza się prosto, aż napotka możliwość skrętu w wybranym kierunku np. w prawo w przypadku algorytmu prawej ręki. Wówczas wykonuje skręt i kontynuuje jazdę, ponownie trzymając się ściany, aż do kolejnej decyzji o skręcie. Na rysunku 3.1 zaprezentowano przykładowy początkowy odcinek trasy, którą podąża robot, stosując opisywaną metodę, natomiast na rysunku 3.2 przedstawiono schemat blokowy obrazujący działanie algorytmu prawej ręki.

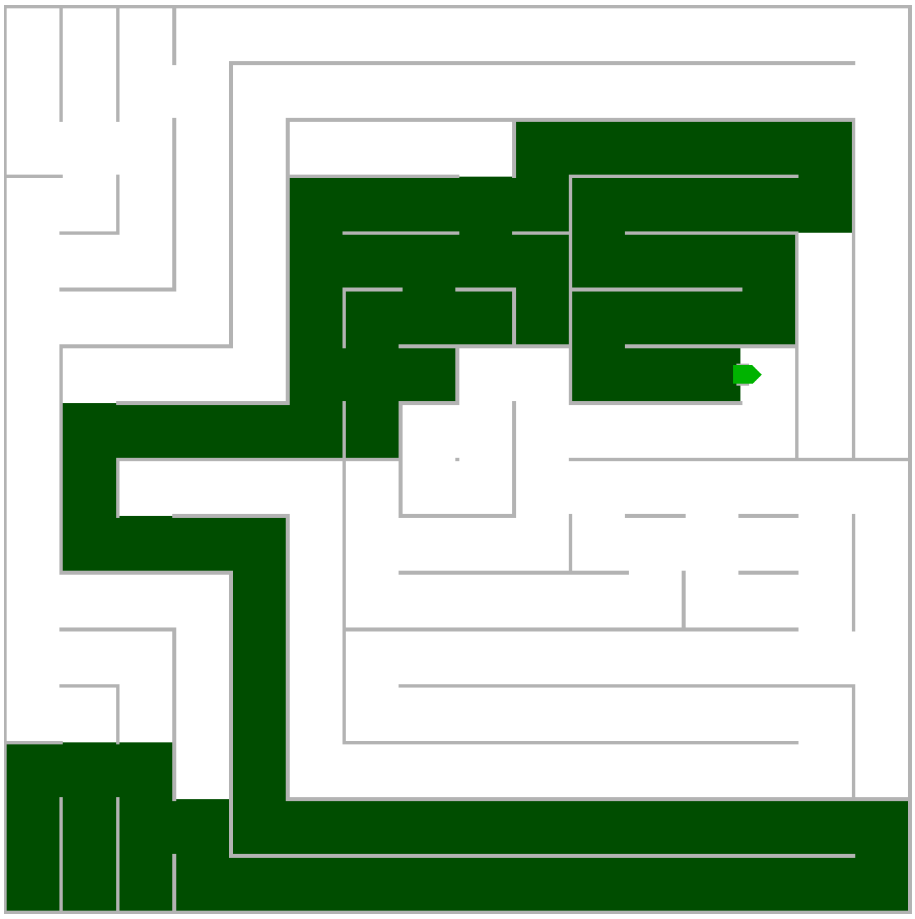
Zaletą opisywanego algorytmu jest jego prostota, łatwość implementacji oraz niska złożoność obliczeniowa. Jednak posiada on również istotne ograniczenia, które sprawiają, że nie zawsze jest skutecznym rozwiązaniem. Bardziej skomplikowane labirynty konstruowane są tak, aby roboty korzystające z tak prostej metody, po prostu nie były w stanie dojechać do mety. Na rysunku 3.3 pokazany jest labirynt, w którym zaznaczona na czerwono jest jego odizolowana część znajdująca się w centrum. Robot podążający tylko wzdłuż ścian, które ma dostępne od początku nie będzie w stanie rozwiązać takiego labiryntu.



Rysunek 3.2: Schemat blokowy algorytmu prawej ręki [11]



Rysunek 3.3: Labirynt z zaznaczoną odizolowaną ścianą(kolor czerwony)



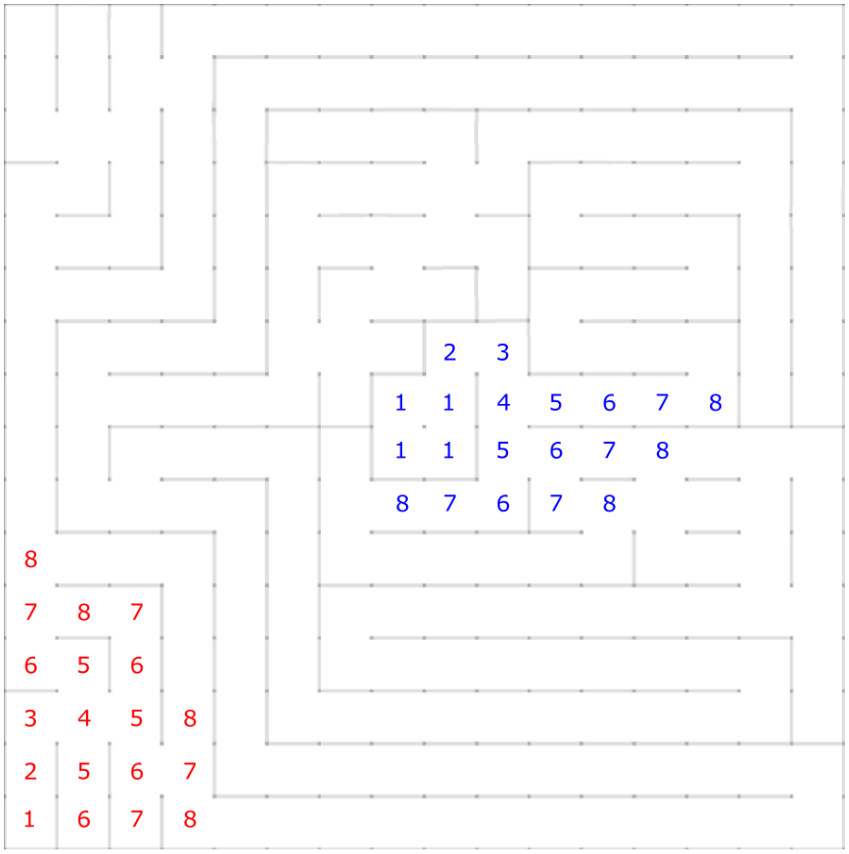
Rysunek 3.4: Przykładowa trasa robota wykorzystującego algorytm DFS

3.1.2 Algorytm grafowy DFS

Algorytm DFS (Depth-First Search) [13] jest algorytmem, którego zadaniem jest eksploracja struktur reprezentowanych w postaci grafów. W kontekście labiryntu wierzchołki grafu odpowiadają poszczególnym komórkom, a krawędzie reprezentują dostępne przejścia między sąsiednimi komórkami. Algorytm działa eksplorując graf w głąb, wybierając każdorazowo pierwszy dostępny sąsiadujący wierzchołek. Gdy napotka komórkę, która nie sąsiaduje z jeszcze nie odwiedzonymi komórkami, cofa się do poprzedniej, by sprawdzić pozostałe możliwości. Proces ten został zobrazowany na rysunku 3.4, trwa dopóki cały labirynt nie zostanie w pełni zbadany.

3.1.3 Algorytm zalewowy

Algorytm zalewowy [12] to zaawansowana metoda nawigacji, która symuluje proces zalewania labiryntu. Jego działanie polega na przypisywaniu każdej komórce w labiryncie wartości liczbowej, która odpowiada minimalnej liczbie kroków potrzebnych do dotarcia do celu. Na rysunku 3.5 kolorem czerwonym pokazano przypisywanie wartości kolejnym komórkom zaczynając z miejsca startu. Dzięki temu robot ma możliwość podejmowania decyzji o kierunku ruchu na podstawie lokalnych wartości sąsiednich komórek.

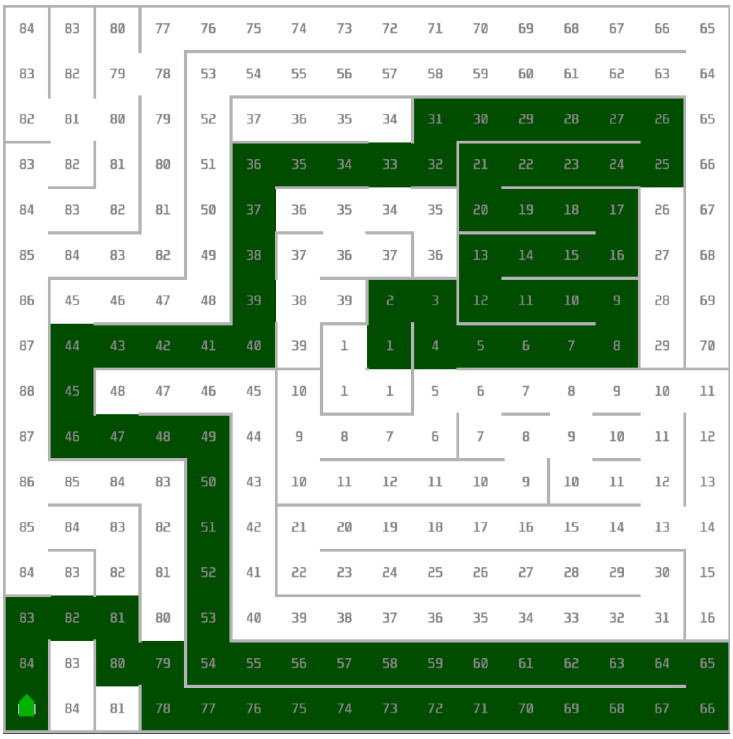


Rysunek 3.5: Wyznaczanie wartości kolejnych pól dla algorytmu zalewowego zaczynając od środka labiryntu (kolor niebieski), zaczynając z miejsca startu (kolor czerwony)

Korzystając z tej metody możemy zbadać labirynt od początku jednak będzie to bardzo długim procesem. Możliwe jest wykorzystanie tego algorytmu już po tym jak labirynt został zbadany i w pełni zmapowany. Możemy wtedy przeprowadzić zalewenie z mety labiryntu tylko symulacyjnie i zapamiętać te wszystkie wartości. Wynik takiego podejścia przedstawiony jest na rysunku 3.5 kolorem niebieskim. Po uzupełnieniu labiryntu wartościami z mety otrzymamy trasę, która jest najkrótszą pod względem ilości komórek od startu do mety. Wystarszy żę będziemy poruszać się ze startu do sąsiedniej komórki, której wartość jest mniejsza od wartości komórki aktualnej.

3.2 Algorytmy wyznaczania i realizacji ścieżki

Po części eksploracyjnej, labirynt został w pełni zmapowany. Kolejnym zadaniem robota jest wyznaczenie ścieżki oraz jej realizacja. Celem tych działań jest minimalizacja czasu potrzebnego na pokonanie drogi ze startu do mety. Poniżej opisane będą sposoby wyznaczania ścieżek oraz to jak będziemy optymalizować ruch robota.



Rysunek 3.6: Wartości komórek labiryntu oraz trasa wyznaczona przez podstawową wersję algorytmu zalewowego

3.2.1 Modyfikacje algorytmu zalewowego

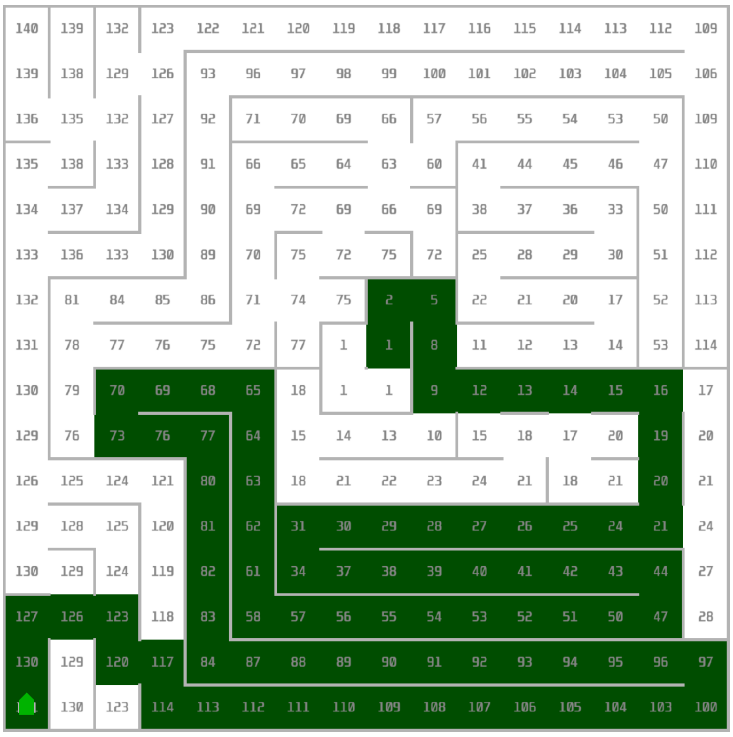
Korzystając z algorytmu zalewowego, otrzymujemy ścieżkę pokazaną na rysunku 3.6. Trasa ta ma 84 komórki długości oraz 28 zakrętów. Jest to najkrótsza trasa prowadząca do mety pod względem liczby komórek, które należy pokonać podczas jej realizacji. Możliwe będzie ulepszenie tego algorytmu tak, aby generował trasy z ograniczoną liczbą zakrętów lub preferujące długie, proste odcinki. Takie modyfikacje umożliwią zmniejszenie czasu potrzebnego na przejazd od startu do mety w labiryncie.

Priorytet tras o mniejszej liczbie zakrętów

Pierwszą modyfikacją jest uwzględnienie kosztu z jakim wiąże się wykonanie przez robota zakrętu. Taki manewr najczęściej wymaga zmniejszenia prędkości, wykonania zakrętu, a następnie ponownego rozpędzenia się, co znacząco wydłuża czas przejazdu. Dokonamy zmiany algorytmu zalewowego, w którym podczas wypełaniania labiryntu odpowiednimi wartościami zgodnie z jego działaniem, komórki do których wymagany jest zakręt będą miały wartość zwiększoną o 3 zamiast 1. Dla przykładowego labiryntu efekt takiej modyfikacji wraz z wyznaczoną ścieżką znajduje się na rysunku 3.7, otrzymana trasa ma długość 86 komórek i 22 zakręty.

Priorytet tras z długimi prostymi odcinkami

Kolejną modyfikacją algorytmu zalewowego jest prioretyzowanie fragmentów o długich prostych odcinkach. Podczas zalewania, gdy zostanie odnaleziony frag-



Rysunek 3.7: Wartości komórek labiryntu oraz trasa wyznaczona przez zmodyfikowany algorytm zalewowy zmniejszający liczbę zakrętów

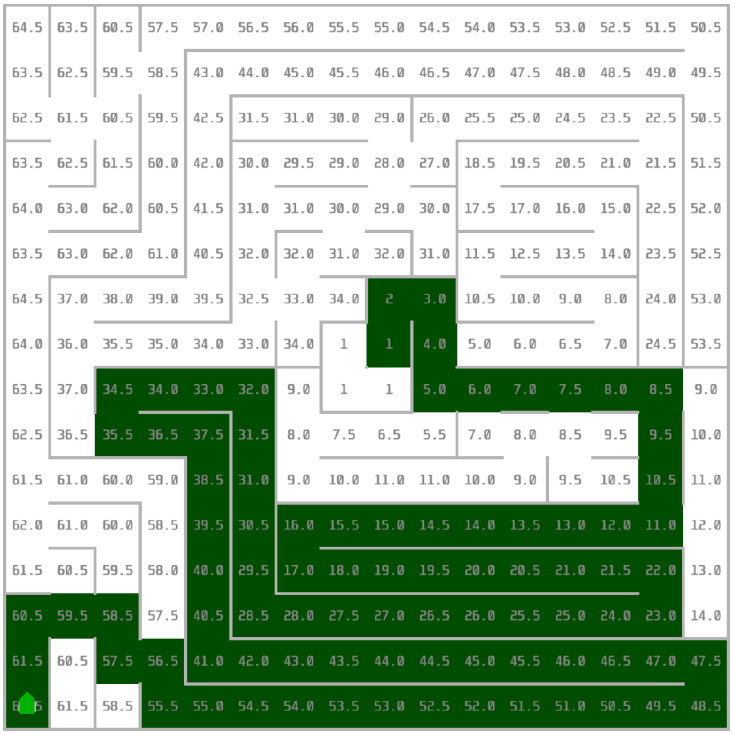
ment, który jest długą prostą, wartości w nim zaczynają być zwiększane o 0,5 zamiast 1. Aby osiągnąć ten efekt, algorytm w trakcie działania śledzi aktualny kierunek ruchu oraz liczbę komórek tworzących bieżącą prostą. Jeśli kierunek pozostaje ten sam przez co najmniej 3 komórki, kolejne komórki w tej samej linii są oceniane korzystniej poprzez obniżenie kosztu ich zalewania. Wynik działania tej wersji algorytmu dla przykładowego labiryntu umieszczono na rysunku 3.8. Trasa wyznaczona jest taka sama jak ta wygenerowana przez poprzednią modyfikację.

Priorytet tras umożliwiających jazdę po skosie

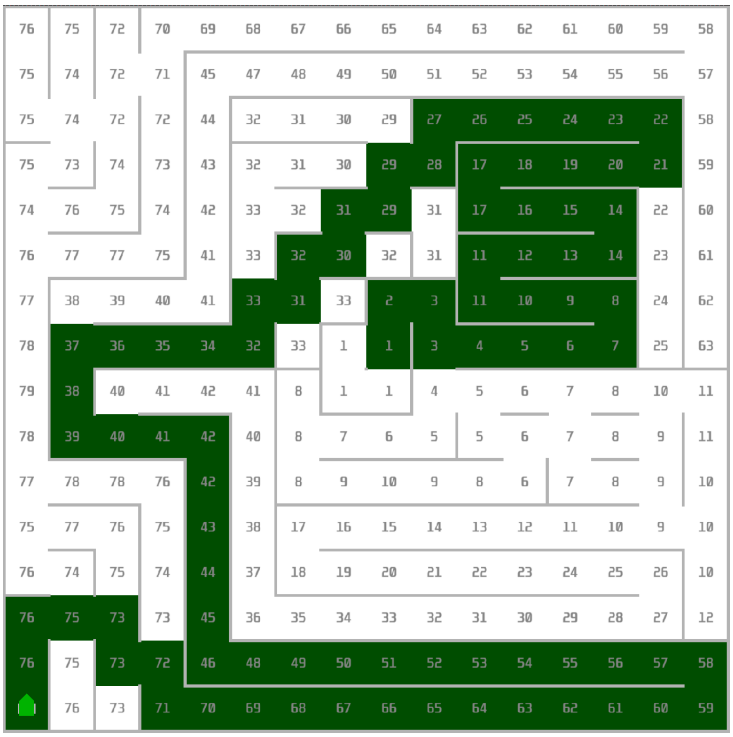
Wcześniejsze modyfikacje nie uwzględniają w żaden sposób możliwości jazdy robota na ukos. Niektóre fragmenty labiryntu, które zawierają wiele naprzemiennych zakrętów, można pokonać w jednym płynnym ruchu jadąc po skosie. W tej wersji algorytmu, priorytet będą miały części, które składają się z wielu naprzemiennych zakrętów. Komórki labiryntu do których jesteśmy w stanie dostać się na ukos nie wjeżdżając w ściankę będą mieć korzystniejsze wartości według algorytmu. Wynik działania takiego algorytmu wraz z zaznaczoną ścieżką pokazano na rysunku 3.9. Zgodnie z oczekiwaniem trasa ma aż 34 zakręty, z czego 10 następuje po sobie.

Priorytet tras z długimi prostymi oraz minimalizujących liczbę zakrętów

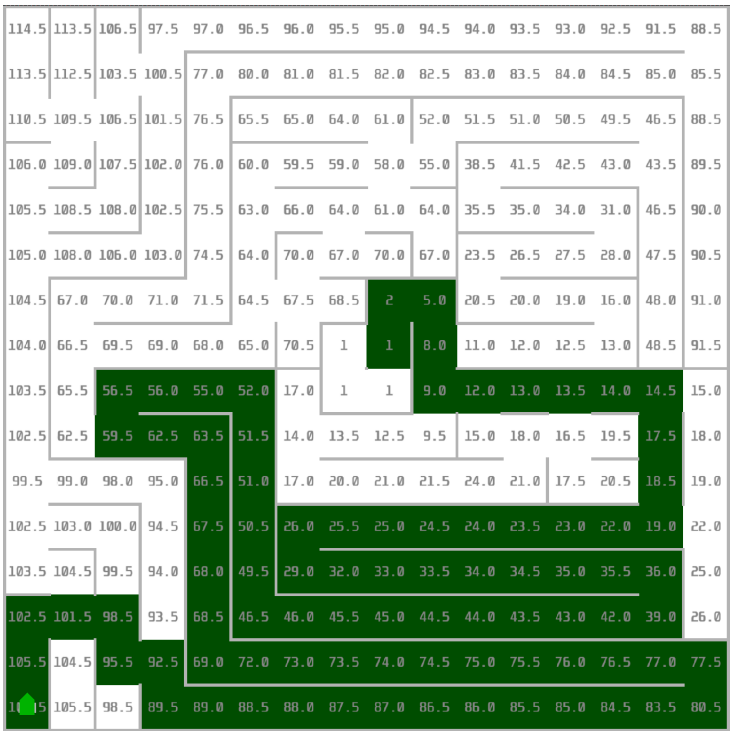
Jest to połączenie dwóch wcześniej omówionych modyfikacji: priorytetu tras z długimi prostymi odcinkami oraz minimalizowania liczby zakrętów. Dla długich prostych o długości ponad 3 pól wartości są zwiększane o 0,5 zamiast 1, a pola dla których wjazd wymaga zakrętu ma wartość zwiększaną o 3 zamiast 1. Wynik



Rysunek 3.8: Wartości komórek labiryntu oraz trasa wyznaczona przez zmodyfikowany algorytm zalewowy, który priorytetyzuje długie proste odcinki



Rysunek 3.9: Wartości komórek labiryntu oraz trasa wyznaczona przez zmodyfikowany algorytm zalewowy, który uwzględnia możliwość jazdy po skosie



Rysunek 3.10: Wartości komórek labiryntu oraz trasa wyznaczona przez zmodyfikowany algorytm zalewowy, który unika zakrętów i szuka długich prostych odcinków

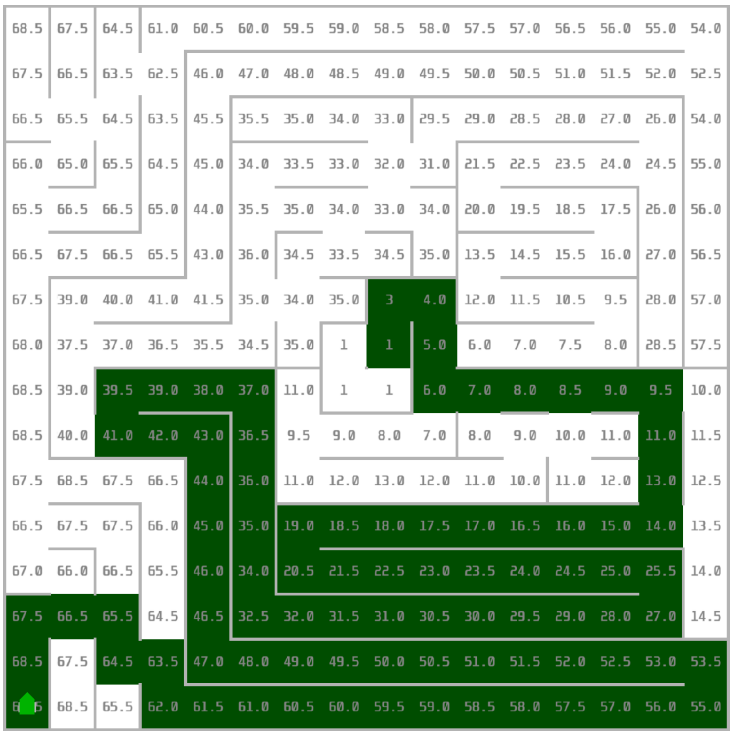
działania takiego algorytmu wraz z zaznaczoną ścieżką pokazano na rysunku 3.10. Uzyskana ścieżka jest taka sama jak dla modyfikacji, która ogranicza liczbę zakrętów oraz priorytetyzująca proste odcinki. Trasa składa się z 86 komórek i wymaga wykonania 22 zakrętów.

Priorytet tras z długimi prostymi oraz umożliwiających jazdę po skosie

Podobnie do poprzedniej modyfikacji jest to kombinacja priorytetu długich prostych oraz fragmentów umożliwiających jazdę po skosie. Dla prostych odcinków o długości przekraczającej 3 pola, ich wartości zwiększa się o 0,5. W przypadku fragmentów umożliwiających ruch po skosie przyznawane wartości są bardziej korzystne. Jeśli jednak długość takiego fragmentu przekracza 4 komórki, zwiększenie wartości wynosi 0,5 zamiast standardowego 1. Na rysunku 3.11 przedstawiono wynik działania algorytmu oraz wyznaczoną trasę, która jest taka sama jak ta wyznaczona przez algorytmy prioretyzujące długie proste, czy minimalizujące liczbę zakrętów.

Porównanie tras otrzymanych z algorytmu zalewowego z modyfikacjami

Dla przykładowego labiryntu na rysunku 3.12 zestawiono trasy wyznaczone powyżej przez algorytm zalewowy oraz jego zmodyfikowane wersje. Jak widać trasy na rysunkach 3.12b, 3.12c, 3.12e oraz 3.12f są identyczne, ponieważ w labiryncie nie ma zbyt wielu fragmentów umożliwiających jazdę po skosie i długich prostych odcinków. Ścieżki na rysunkach 3.12a oraz 3.12d są do siebie bardzo zbliżone, a różnica wynika z tego, że algorytm wykrył fragment umożliwiający jazdę po skosie.



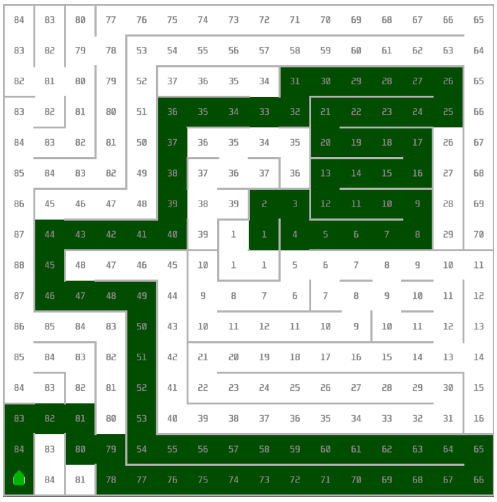
Rysunek 3.11: Wartości komórek labiryntu oraz trasa wyznaczona przez zmodyfikowany algorytm zalewowy, który szuka długich prostych i fragmenów umożliwiających jazdę po skosie

3.2.2 Realizacja ścieżki

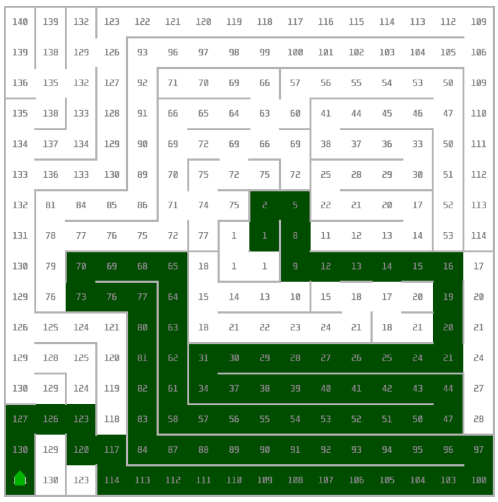
Po tym jak algorytm zalewowy lub jego zmodyfikowane wersje zakończą swoje działanie otrzymamy trasę, którą teraz robot będzie musiał przejechać w jak najkrótszym czasie. Ścieżkę można zapisać jako listę poleceń, takich jak „jedź do przodu" lub „skręć w prawo/lewo" o zadany kąt, które wykonane w odpowiedniej kolejności przeprowadzą robota ze startu do mety. Jednak sama lista poleceń nie uwzględnia optymalizacji czasowej np. maksymalnego wykorzystania prostych odcinków do rozpędzenia robota lub ograniczenia zbędnych manewrów podczas zakrętów.

Przyspieszanie na długich prostych

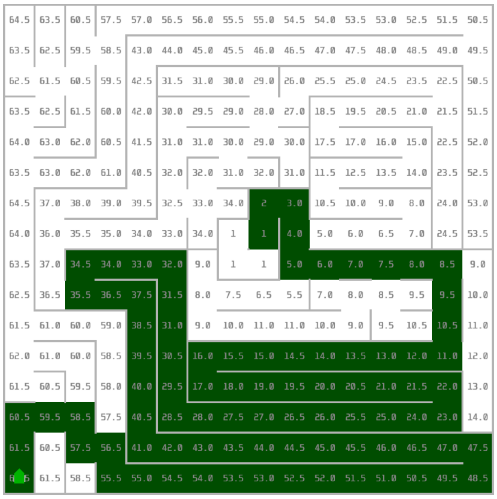
W zależności od długości prostej, którą robot ma za zadanie pokonać wartości do jakich będzie w stanie się rozpędzić będą różne. Do tego trzeba uwzględnić jakie robot ma przyspieszenie oraz jak długa jest jego droga hamowania. Załóżmy, że robot dysponuje stałym przyspieszeniem i potrzebuje n pól labiryntu, aby osiągnąć swoją maksymalną prędkość, takiego samego dystansu do pełnego wyhamowania z maksymalnej prędkości. Na rysunku 3.13 zobrazowany jest wykres prędkości dla $n = 3$ na trasie o długości 8 komórek, a na rysunku 3.14 prędkość, gdy robot musi pokonać 4 pola, zatrzymać się i ponownie rozpocząć ruch.



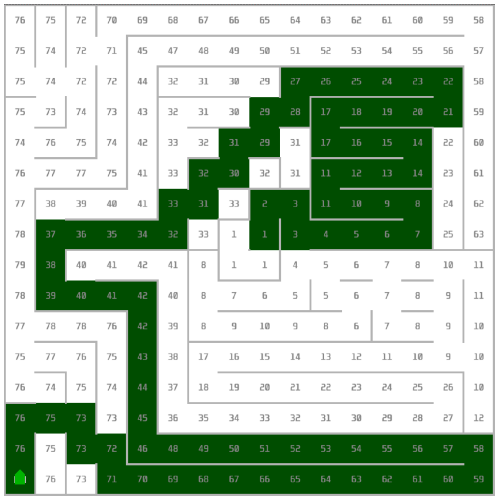
(a) Wersja podstawowa



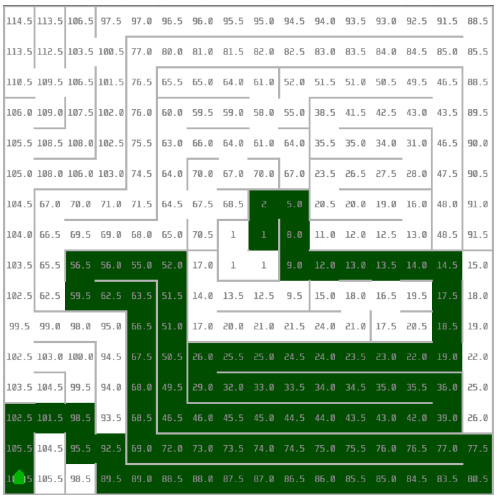
(b) Zmniejszanie liczby zakrętów



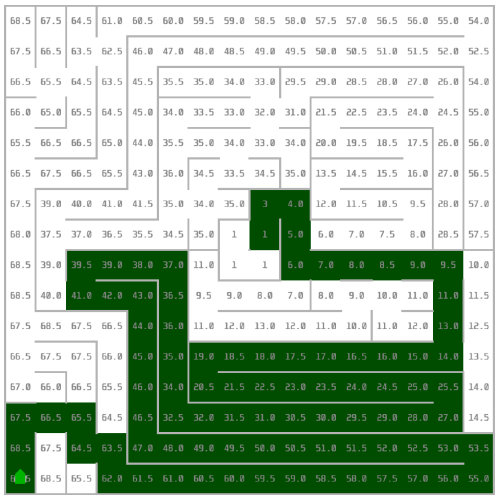
(c) Priorytet tras z długimi prostymi odcinkami



(d) Priorytet tras z naprzemiennymi zakrętami

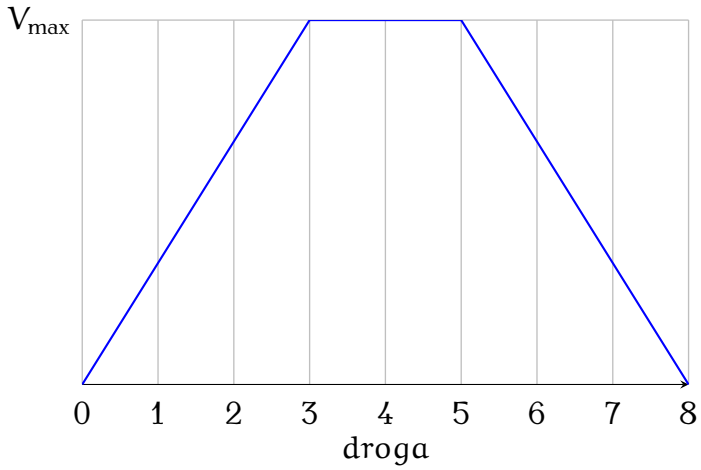


(e) Zmniejszanie liczny zakrętów oraz priorytet długich prostych

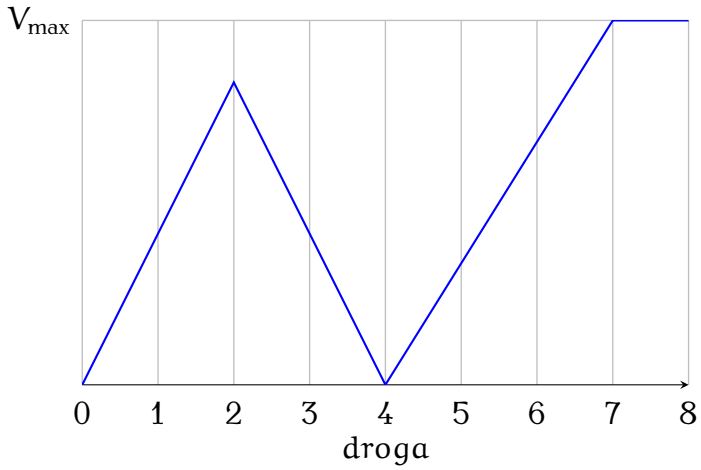


(f) Priorytet tras z naprzemiennymi zakrętami oraz dlugimi prostymi

Rysunek 3.12: Porównanie ścieżek dla przykładowego labiryntu, wyznaczonych przez algorytm zalewowy oraz jego zmodyfikowane wersje



Rysunek 3.13: Wartości prędkości dla $n = 3$, gdy robot ma przejechać 8 komórek



Rysunek 3.14: Wartości prędkości dla $n = 3$, gdy robot ma przejechać 4 komórki, zatrzymać się i ponownie się rozpędzić

Jazda na ukos

W celu jazdy po skosie najpierw musimy wykryć fragment, w którym możemy użyć takiej techniki. Przed rozpoczęciem wykonywania listy poleceń możemy ją przeanalizować i odpowiednio zmodyfikować. Jeśli wykryjemy sekwencję ruchów w postaci: jedź komórkę do przodu , skręć w prawo/lewo, jedź komórkę do przodu, skręć w lewo/prawo itd. możemy wtedy zamienić ją na 3 kolejne polecenia. Pierwsze skręt o 45° lub 135° w odpowiednim kierunku, przejechania odpowiedniej odległości oraz ponownego skrętu o 45° lub 135°, w celu powrotu do pierwotnej orientacji umożliwiającej kontynuację nawigacji.

Rozdział 4

Implementacja i badania symulacyjne

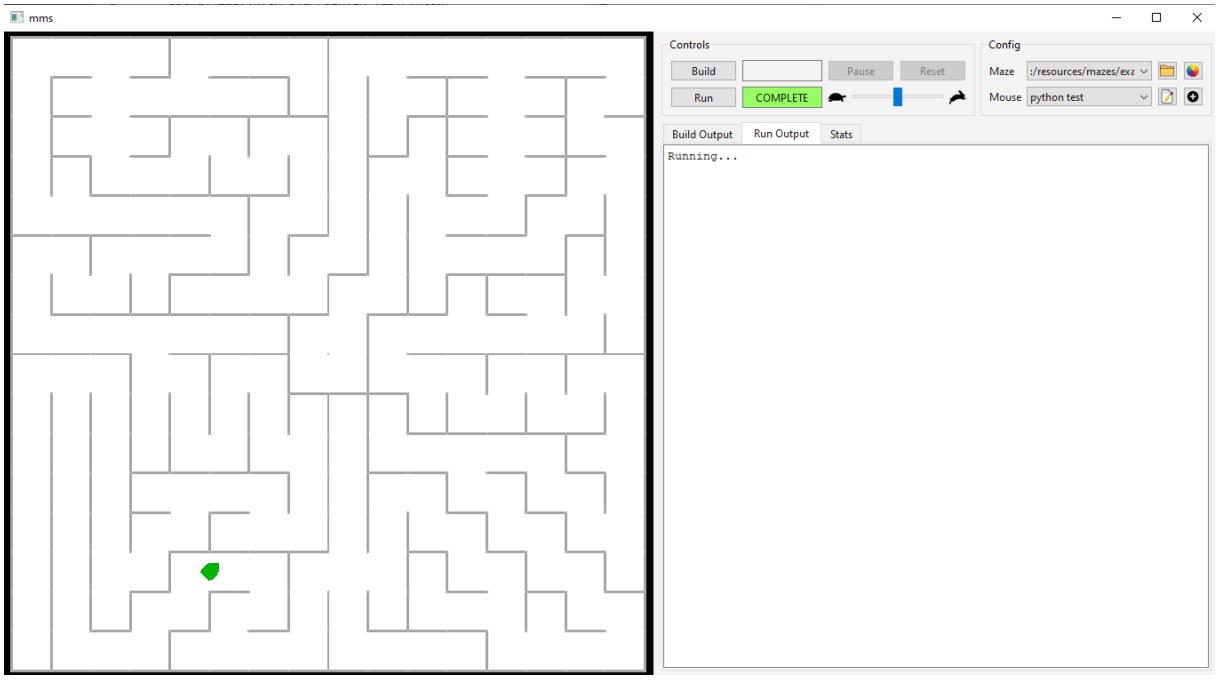
Celem badań symulacyjnych było przetestowanie algorytmów wykorzystywanych przez roboty klasy micromouse, bez konieczności fizycznego wykonania prototypu. Symulacje pozwalają na przeprowadzenie wielu eksperymentów w kontrolowanym środowisku, umożliwiając analizę wydajności i dokładności algorytmów nawigacyjnych oraz optymalizacyjnych dla różnych labiryntów.

W badaniach wykorzystano program *mms* [9], który umożliwia odtworzenie wirtualnego środowiska labiryntu oraz symulację ruchu robota micromouse. Dzięki temu możliwe było przeprowadzenie analiz efektywności poszczególnych algorytmów, m.in. algorytmu prawej/lewej ręki oraz algorytmu zalewowego. Poniżej opisano środowisko symulacyjne oraz przedstawiono analizę wyników otrzymanych podczas testów.

4.1 Opis programu symulacyjnego

Program symulacyjny *mms* [9] to zaawansowane narzędzie do modelowania zachowań robota micromouse w różnych labiryntach. Widoczny na rysunku 4.1 interfejs programu, jest intuicyjny, ma możliwość dodawania własnych labiryntów oraz skryptów, które przekazują robotowi polecenia. Jedną z kluczowych funkcji programu jest możliwość przetestowania algorytmów nawigacyjnych, które można implementować przy pomocy skryptów w róznych językach programowania takich jak: C, C++, Java, JavaScript, Python czy Rust. W opracowywanym przypadku skrypty zostały napisane w języku programowania Python.

Program *mms* pokazuje nam graficznie labirynt z aktualną pozycją robota. Dla ułatwienia analizy można zmienić kolor każdej komórki oraz dodać do nich tekst, co daje możliwość oznaczania pól w jakich robot się już znalazł. Z poziomu skryptów jesteśmy w stanie wydawać polecenia o jeździe robota do przodu o określoną liczbę pól, sprawdzenia jaki jest rozkład ścian w aktualnej komórce labiryntu, w której robot się znajduje oraz wykonanie obrotu w miejscu. Problemem jest zmiana prędkości poruszania się, domyślnie możliwe jest to tylko poprzez suwak znajdujący się na środku górnej części interfejsu. Gdy skrypt zawiera polecenie, by robot wjechał w ściankę, program kończy działanie zwracając informacje co zostało wykonane



Rysunek 4.1: Interfejs symulatora mms

niepoprawnie.

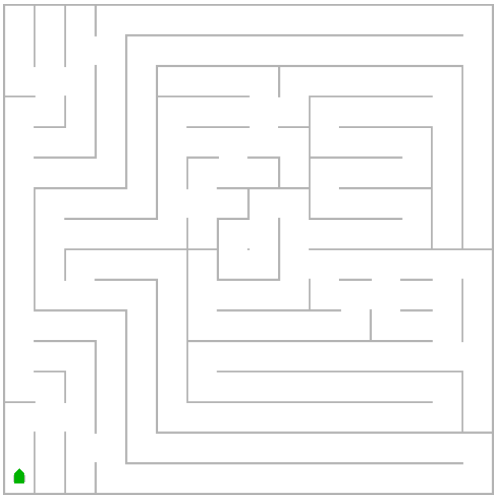
Jedynym wymaganym usprawnieniem programu było dodanie możliwości zmiany prędkości robota bezpośrednio z poziomu skryptu. Dotychczasowa metoda, która zakładała zmianę prędkości za pomocą interfejsu użytkownika, była niewystarczająca w sytuacjach, gdy prędkość robota musiała być dynamicznie dostosowywana, np. podczas przyspieszania. W związku z tym wprowadzono nową funkcjonalność, pozwalającą na modyfikację prędkości bezpośrednio w skrypcie, co umożliło przeprowadzenie testów zgodnie z planem.

Wszystkie testy zostały przeprowadzone na labiryntach zaprezentowanych na rysunku 4.2. Labirynty znajdujące się na rysunkach 4.2a–4.2d są domyślnie zainstalowane w symulatorze. Rysunek 4.2e przedstawia labirynt, w którym rozgrywana była konkurencja micromouse na zawodach APEC 2023 [4] , a na rysunku 4.2f znajduje się labirynt wykorzystany na zawodach w Japonii [2, 3], które odbyły się na terenie kampus Atsugi Politechniki w Tokio w 2024 roku.

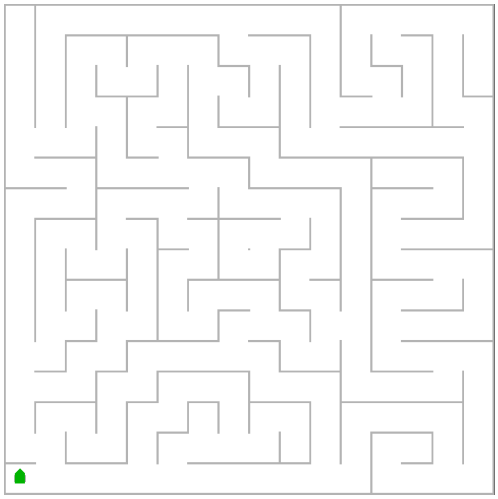
4.2 Symulacje eksploracji labiryntu

Wskaźnikami oceny badanymi podczas symulacji dla każdego z labiryntów są:

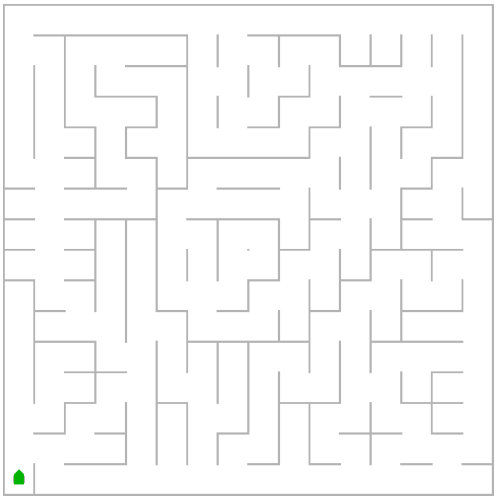
- czas w jakim labirynt został w pełni zbadany,
- liczba przejechanych komórek,
- liczba wykonanych zakrętów,
- fakt czy labirynt został w całości zbadany.



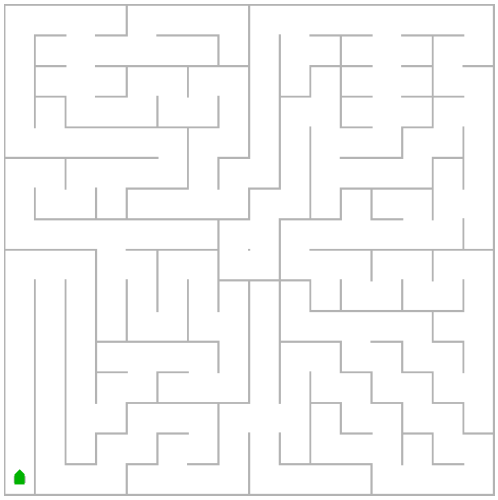
(a) Labirynt 1



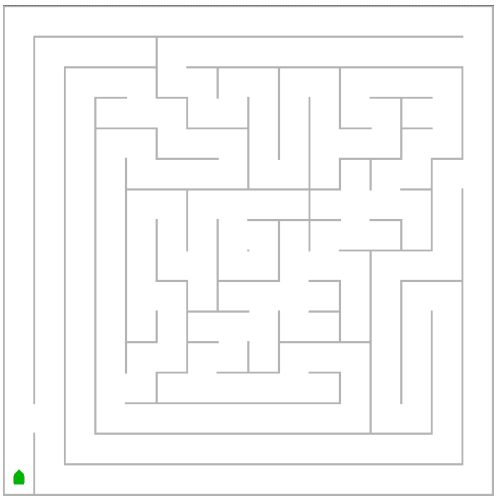
(b) Labirynt 2



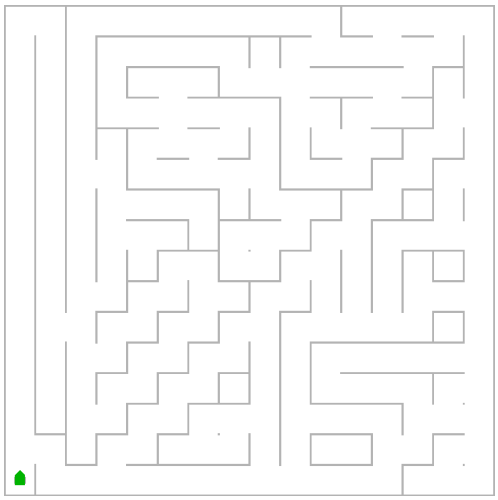
(c) Labirynt 3



(d) Labirynt 4



(e) Labirynt 5



(f) Labirynt 6

Rysunek 4.2: Labirynty wykorzystywane podczas testów

Tabela 4.1: Porównanie wyników algorytmu prawej ręki

	Liczba komórek	Liczba zakrętów	Czas eksploracji
Labirynt 1	258	81	—
Labirynt 2	117	46	—
Labirynt 3	118	41	—
Labirynt 4	230	143	—
Labirynt 5	64	7	—
Labirynt 6	94	26	—

Tabela 4.2: Porównanie wyników algorytmu DFS

	Liczba komórek	Liczba zakrętów	Czas eksploracji
Labirynt 1	510	222	170.30 s
Labirynt 2	510	303	182.99 s
Labirynt 3	510	270	177.06 s
Labirynt 4	510	358	188.47 s
Labirynt 5	510	218	170.24 s
Labirynt 6	498	306	181.03 s

4.2.1 Algorytm prawej/lewej ręki

Algorytm prawej/lewej ręki nie poradził sobie z zadaniem eksploracji w żadnym z labiryntów. W labiryncie 1 i 4 był w stanie zbadać większą ich częśc, z kolei w 5 mógł on tylko praktycznie przejechać się dookoła labiryntu. Wyniki dla tego algorytmu zostały przedstawione w tabeli 4.1. Algorytm kończył swoje działanie gdy ponownie dotarł do pola startu.

4.2.2 Algorytm DFS

Algorytm grafowy DFS znakomicie poradził sobie z eksploracją labiryntu, skutecznie badając każdy z nich w całości w czasie około 3 minut. Wyniki działania tego algorytmu przedstawiono w tabeli 4.2. Jak widać dla każdego z labiryntów z wyjątkiem 6, robot musiał przejechać za każdym razem taką samą liczbę komórek. Różnica dla labiryntu 6 wynika z tego, że znajduje się w nim 6 komórek, które mają ściankę z każdej strony, co powoduje że robot nie jest w stanie do nich wjechać.

4.3 Symulacje realizacji ścieżki

Podobnie do przypadku testów algorytmów eksploracyjnych badanymi wskaźnikami oceny są:

- czas w jakim robot przemieścił się ze startu do mety,
- liczba przejechanych komórek,
- liczba wykonanych zakrętów.

Tabela 4.3: Wyniki dla tras w wersji A

	Scenariusz 1			Scenariusz 2			Scenariusz 3			Scenariusz 4		
Labirynt 1	84	28	26.38	84	28	18.29	84	26	24.46	84	26	17.01
Labirynt 2	26	16	9.71	26	16	7.67	26	16	7.82	26	16	7.12
Labirynt 3	54	34	19.71	54	34	13.36	54	22	13.25	54	22	10.72
Labirynt 4	98	60	35.50	98	60	26.23	98	44	26.65	98	44	22.70
Labirynt 5	113	36	35.27	113	36	20.33	113	33	33.08	113	33	19.23
Labirynt 6	60	21	19.51	60	21	11.66	60	19	16.21	60	19	11.33

Każdy z labiryntów został rozwiązany zgodnie z działaniem algorytmu zalewowego oraz jego modyfikacji opisanych w rozdziale 3.2. Dla każdej trasy testowej przeprowadzane zostały cztery różne scenariusze jazdy:

- scenariusz 1 — jazda ze stałą prędkością, bez możliwości poruszania się na ukos,
- scenariusz 2 — jazda z możliwością przyspieszania, bez możliwości poruszania się na ukos,
- scenariusz 3 — jazda ze stałą prędkością, z możliwością poruszania się na ukos,
- scenariusz 4 — jazda z możliwością przyspieszania oraz poruszania się na ukos.

Dla każdego labiryntu zostało wygenerowanych po 6 tras, każda wyznaczona na bazie algorytmu zalewowego oraz jego modyfikacji:

- wersja A — podstawowy algorytm zalewowy,
- wersja B — priorytet tras o mniejszej liczbie zakrętów,
- wersja C — priorytet tras z długimi prostymi odcinkami,
- wersja D — priorytet tras umożliwiających jazdę po skosie,
- wersja E — priorytet tras z długimi prostymi oraz minimalizujących liczbę zakrętów,
- wersja F — priorytet tras z długimi prostymi oraz umożliwiających jazdę po skosie.

Wyniki dla poszczególnych metod zostały przedstawione w tabelach 4.3–4.8. Dane jakie są umieszczone w poszczególnych komórkach to liczba przejechanych pól, liczba zakrętów oraz czas przejazdu ze startu do mety w sekundach. W dalszej części rozdziału opisy porównywanych wersji odnosić się będą do scenariusza 4.

Tabela 4.4: Wyniki dla tras w wersji B

	Scenariusz 1			Scenariusz 2			Scenariusz 3			Scenariusz 4		
Labirynt 1	86	22	25.62	86	22	13.84	86	20	24.95	86	20	13.85
Labirynt 2	26	16	9.71	26	16	7.67	26	16	7.82	26	16	7.12
Labirynt 3	58	24	19.22	58	24	12.09	58	22	15.68	58	22	11.31
Labirynt 4	100	52	34.37	100	52	26.72	100	48	28.32	100	48	23.99
Labirynt 5	113	36	35.27	113	36	20.33	113	33	33.08	113	33	19.23
Labirynt 6	60	21	19.33	60	21	12.68	60	19	16.69	60	19	11.61

Tabela 4.5: Wyniki dla tras w wersji C

	Scenariusz 1			Scenariusz 2			Scenariusz 3			Scenariusz 4		
Labirynt 1	86	22	25.62	86	22	13.84	86	20	24.95	86	20	13.85
Labirynt 2	26	16	9.71	26	16	7.67	26	16	7.82	26	16	7.12
Labirynt 3	58	24	19.45	58	24	12.71	58	22	15.89	58	22	11.53
Labirynt 4	100	52	34.37	100	52	26.72	100	48	28.32	100	48	23.99
Labirynt 5	113	36	35.27	113	36	20.33	113	33	33.08	113	33	19.23
Labirynt 6	60	21	20.21	60	21	12.61	60	17	17.33	60	17	12.04

Tabela 4.6: Wyniki dla tras w wersji D

	Scenariusz 1			Scenariusz 2			Scenariusz 3			Scenariusz 4		
Labirynt 1	88	34	27.73	88	34	18.41	88	24	23.84	88	24	15.41
Labirynt 2	26	18	10.05	26	18	8.05	26	15	7.33	26	15	6.99
Labirynt 3	54	34	19,71	54	34	13.36	54	22	13.25	54	22	10.72
Labirynt 4	98	60	35.50	98	60	26.23	98	44	26.65	98	44	22.70
Labirynt 5	117	54	39.21	117	54	24.14	117	37	32.42	117	37	19.07
Labirynt 6	60	45	22.99	60	45	17.88	60	19	14.04	60	19	9.92

Tabela 4.7: Wyniki dla tras w wersji E

	Scenariusz 1			Scenariusz 2			Scenariusz 3			Scenariusz 4		
Labirynt 1	86	22	25.62	86	22	13.84	86	20	24.95	86	20	13.85
Labirynt 2	26	16	9.71	26	16	7.67	26	16	7.82	26	16	7.12
Labirynt 3	56	24	18.63	56	24	11.70	56	21	15.43	55	21	11,14
Labirynt 4	100	52	34.37	100	52	26.72	100	48	28.32	100	48	23.99
Labirynt 5	113	36	35.27	113	36	20.33	113	33	33.08	113	33	19.23
Labirynt 6	68	17	20.41	68	17	10.43	68	17	18.99	68	16	9.90

Tabela 4.8: Wyniki dla tras w wersji F

	Scenariusz 1			Scenariusz 2			Scenariusz 3			Scenariusz 4		
Labirynt 1	86	22	25.62	86	22	13.84	86	20	24.95	86	20	13.85
Labirynt 2	26	16	9.71	26	16	7.67	26	16	7.82	26	16	7.12
Labirynt 3	56	28	19.08	56	28	11.61	56	16	14.08	56	16	9.05
Labirynt 4	98	60	35.50	98	60	26.23	98	44	26.65	98	44	22.70
Labirynt 5	113	36	35.27	113	36	20.33	113	33	33.08	113	33	19.23
Labirynt 6	60	41	22.23	60	41	16.43	60	16	14.31	60	16	9.51

Tabela 4.9: Wyniki dla labiryntu 1

Wersja	Scenariusz 1			Scenariusz 2			Scenariusz 3			Scenariusz 4		
A	84	28	26.38	84	28	18.29	84	26	24.46	84	26	17.01
D	88	34	27.73	88	34	18.41	88	24	23.84	88	24	15.41
B, C, E, F	86	22	25.62	86	22	13.84	86	20	24.95	86	20	13.85

Tabela 4.10: Wyniki dla labiryntu 2

Wersja	Scenariusz 1			Scenariusz 2			Scenariusz 3			Scenariusz 4		
A, B, C, E, F	26	16	9.71	26	16	7.67	26	16	7.82	26	16	7.12
D	26	18	10.05	26	18	8.05	26	15	7.33	26	15	6.99

4.3.1 Labirynt 1

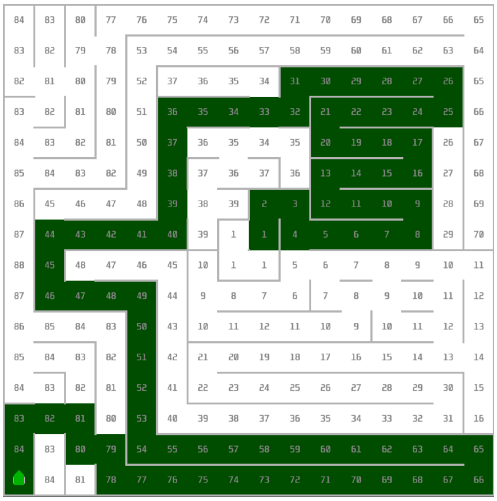
Zestawienie wszystkich tras dla labiryntu 1 widoczne jest na rysunku 4.3. Otrzymaliśmy w sumie 3 różne ścieżki znajdujące się na rysunkach 4.3a, 4.3b i 4.3d. Dla każdej z tych ścieżek przeprowadzone zostały testy, których wyniki przedstawione zostały w tabeli 4.9. Patrząc na tą tabelę możemy zauważyć, że zmodyfikowane wersje algorytmu zalewowego umożliwiły robotowi odnalezienie ścieżek, które był w stanie pokonać w krótszym czasie w porównaniu do wersji podstawowej. Dla tego konkretnego labiryntu wersje B, C, E oraz F poradziły sobie najlepiej.

4.3.2 Labirynt 2

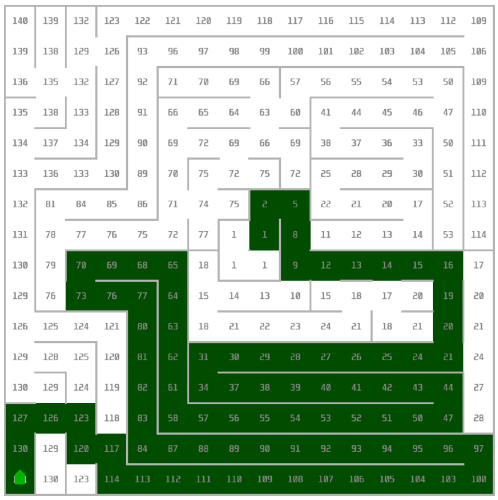
Na rysunku 4.4 pokazano wszystkie ścieżki dla labiryntu 2, dla którego wyznaczone zostały tylko 2 różne trasy. Jedna widoczna na rysunku 4.4a, a druga na 4.4d. Wyniki testów przeprowadzonych dla tych ścieżek widoczne są w tabeli 4.10. Wersja D algorytmu zalewowego dała lepszą trasę pod względem czasu przejazdu.

4.3.3 Labirynt 3

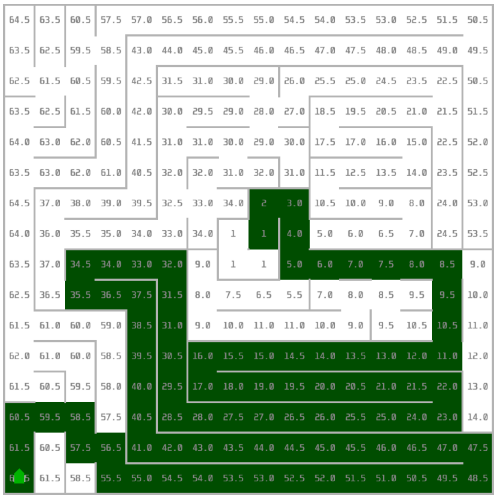
Na rysunku 4.5 pokazane są wszystkie ścieżki dla labiryntu 3. Wyznaczono aż 5 różnych tras kolejno zaprezentowanych na rysunkach 4.5a, 4.5b, 4.5c, 4.5e i 4.5f. Wyniki przejazdów testowych znajdują się w tabeli 4.11. Wersja A oraz D wyznaczyły ścieżkę, która dawała lepsze czasy przejazdów w porównaniu do wersji C, B i E. Trasa w wersji F została pokonana w znacząco krótszym czasie, co wynika to z tego, że został znaleziony idealny fragment wykorzystujący możliwość jazdy po skosie.



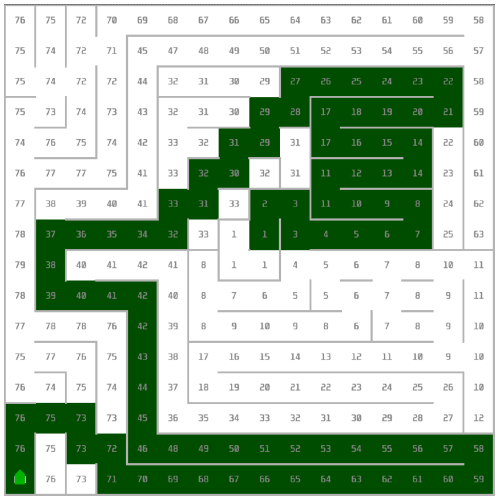
(a) Wersja A



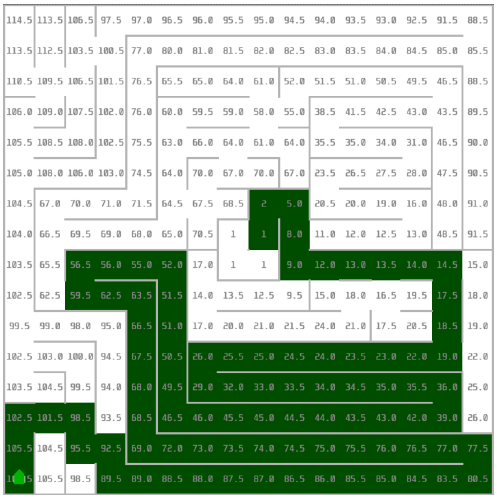
(b) Wersja B



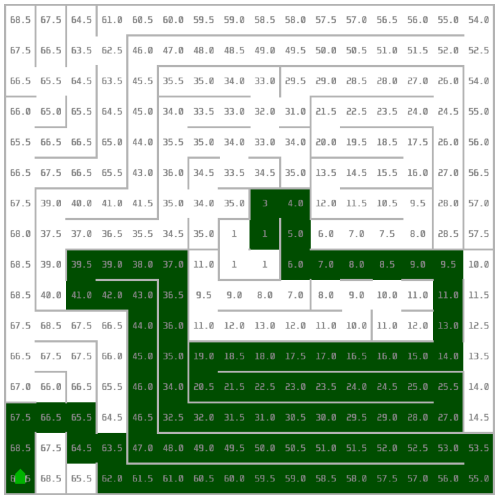
(c) Wersja C



(d) Wersja D

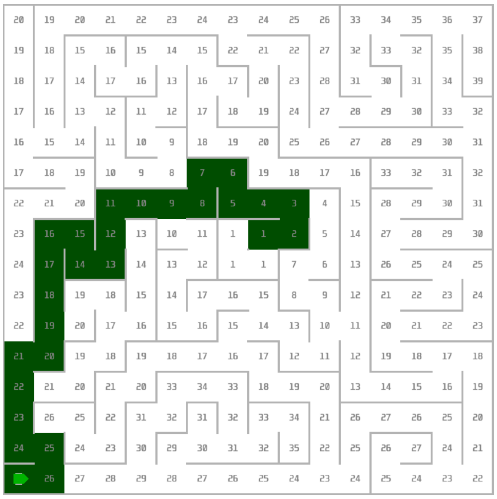


(e) Wersja E

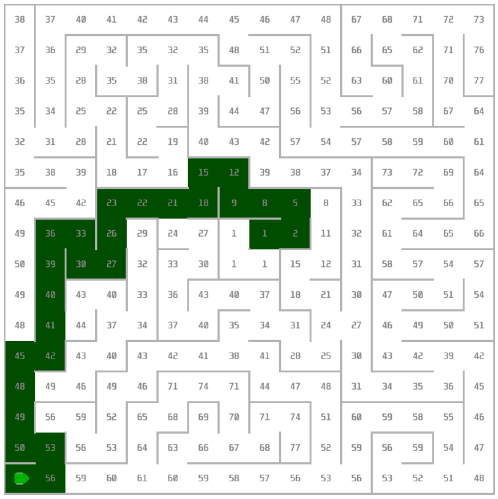


(f) Wersja F

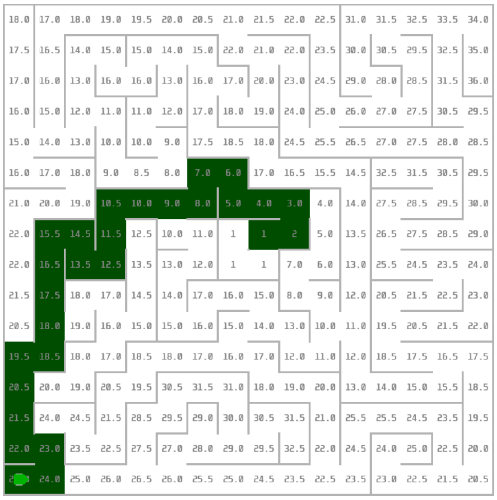
Rysunek 4.3: Porównanie ścieżek dla labiryntu 1



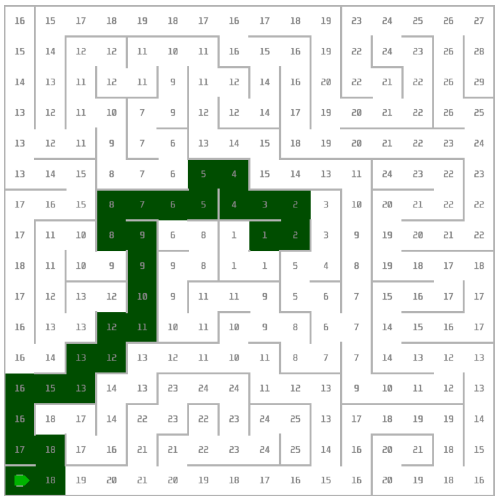
(a) Wersja A



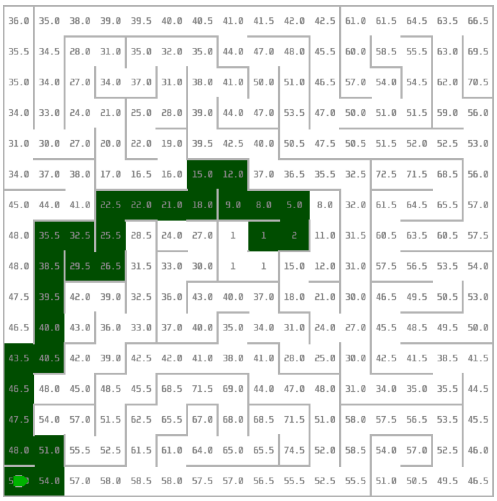
(b) Wersja B



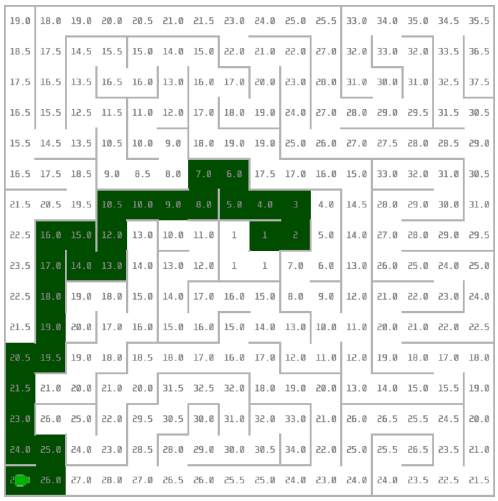
(c) Wersja C



(d) Wersja D

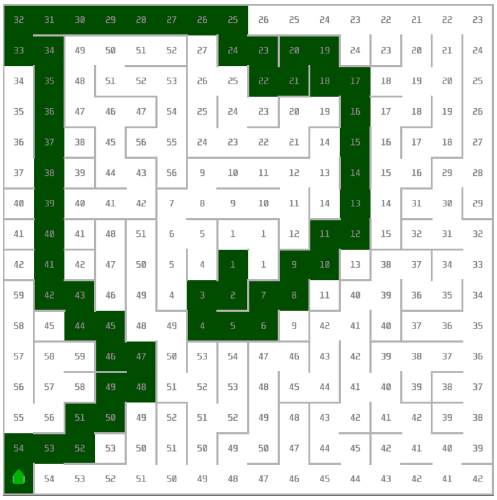


(e) Wersja E

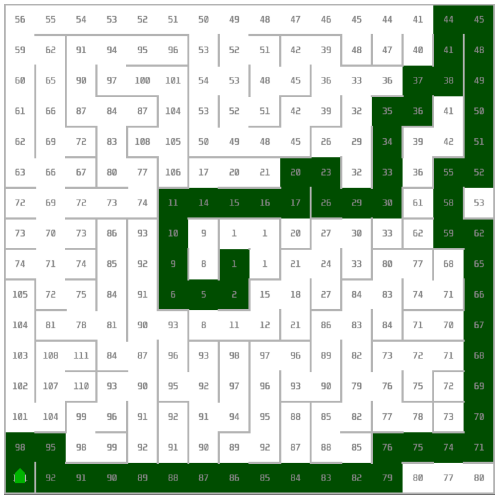


(f) Wersja F

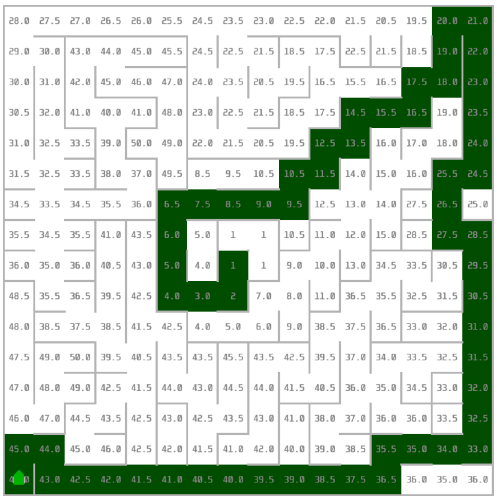
Rysunek 4.4: Porównanie ścieżek dla labiryntu 2



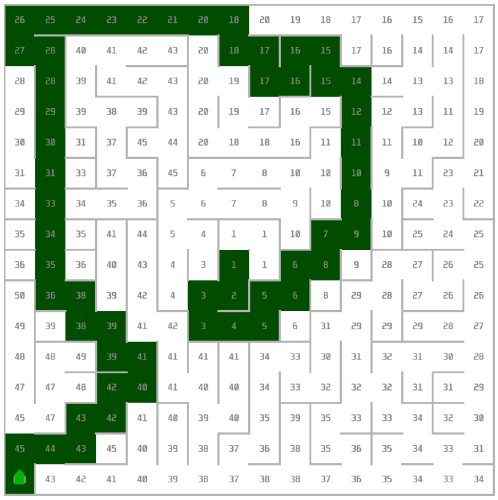
(a) Wersja A



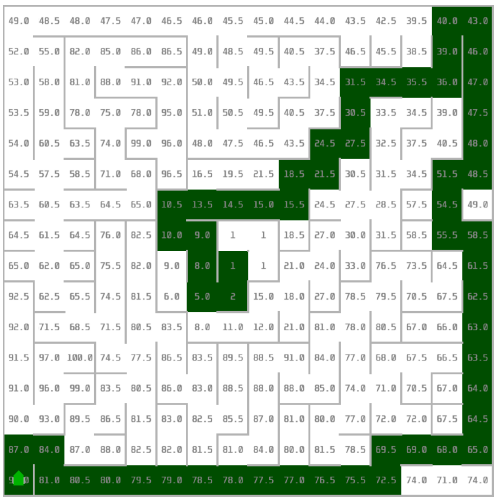
(b) Wersja B



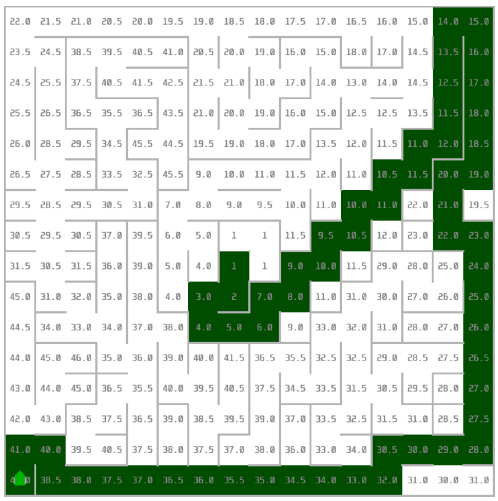
(c) Wersja C



(d) Wersja D



(e) Wersja E



(f) Wersja F

Rysunek 4.5: Porównanie ścieżek dla labiryntu 3

Tabela 4.11: Wyniki dla labiryntu 3

Wersja	Scenariusz 1			Scenariusz 2			Scenariusz 3			Scenariusz 4		
A, D	54	34	19.71	54	34	13.36	54	22	13.25	54	22	10.72
B	58	24	19.22	58	24	12.09	58	22	15.68	58	22	11.31
C	58	24	19.45	58	24	12.71	58	22	15.89	58	22	11.53
E	56	24	18.63	56	24	11.70	56	21	15.43	55	21	11.14
F	56	28	19.08	56	28	11.61	56	16	14.08	56	16	9.05

Tabela 4.12: Wyniki dla labiryntu 4

Wersja	Scenariusz 1			Scenariusz 2			Scenariusz 3			Scenariusz 4		
A, D, F	98	60	35.50	98	60	26.23	98	44	26.65	98	44	22.70
B, C, E	100	52	34.37	100	52	26.72	100	48	28.32	100	48	23.99

Tabela 4.13: Wyniki dla labiryntu 5

Wersja	Scenariusz 1			Scenariusz 2			Scenariusz 3			Scenariusz 4		
A, B, C, E, F	113	36	35.27	113	36	20.33	113	33	33.08	113	33	19.23
D	117	54	39.21	117	54	24.14	117	37	32.42	117	37	19.07

4.3.4 Testy dla labiryntu 4

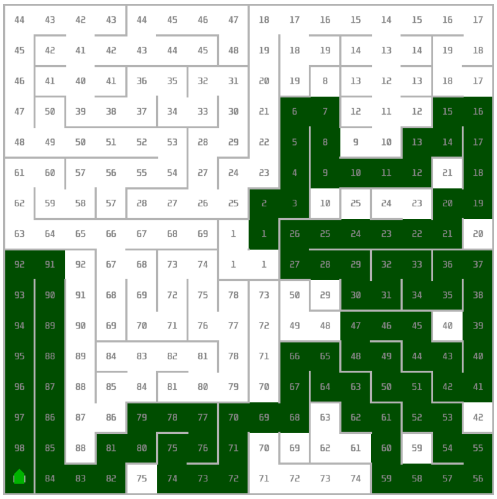
Dla labiryntu 4 wygenerowane zostały 2 ścieżki, pierwsza wyznaczona przez wersje A, D i F znajduje się na rysunku 4.6a oraz druga wyznaczona przez wersje B, C i E na rysunku 4.6b. Wszystkie trasy dla tego labiryntu zostały zestawione na rysunku 4.6, a wyniki w tabeli 4.12. Porównywane ścieżki znacząco się od siebie różnią, prowadzą one robota w zupełnie inne fragmenty labiryntu. Algorytm podstawowy A oraz wersje D i F, które mają za zadanie odnajdywać fragmenty umożliwiające jazdę po skosie wyznaczają trasę, którą robot pokonuje w krótszym czasie.

4.3.5 Testy dla labiryntu 5

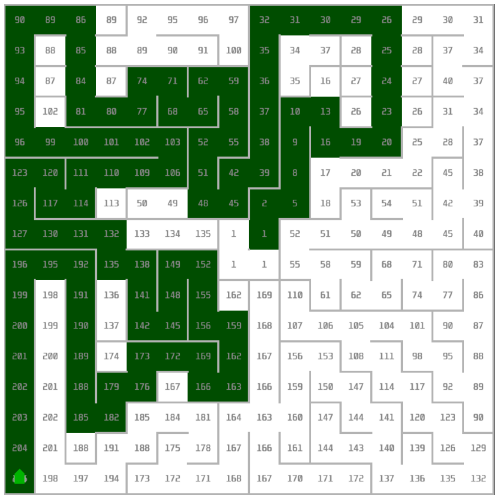
Podobnie do labiryntu 4, dla labiryntu 5 również zostały wygenerowane tylko 2 trasy. Pierwsza wyznaczona przez wersje D znajduje się na rysunku 4.7d oraz druga wyznaczona przez resztę wersji rysunek 4.7a. Wyniki badań symulacyjnych dla tych 2 ścieżek zaprezentowane zostały w tabeli 4.13. Wersja D wyznaczyła trasę, którą robot pokonał w czasie krótszym o 0.16 sekundy.

4.3.6 Testy dla labiryntu 6

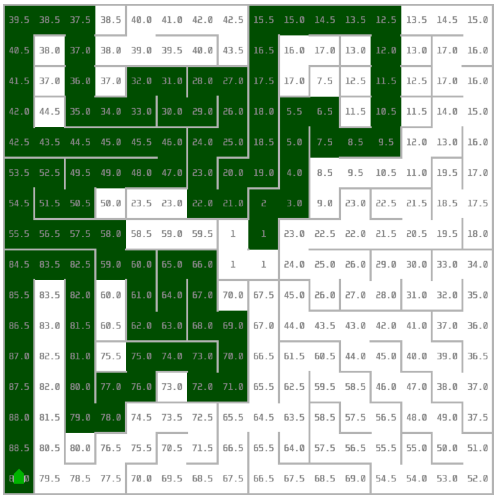
Na rysunku 4.8 przedstawione są ścieżki dla labiryntu 6. Wszystkie omawiane wersje algorytmu zalewowego wyznaczyły różne trasy, wyniki dla każdej z nich znajdują się w tabeli 4.14. Trasę którą robot pokonał w najkrótszym czasie jest ta wyznaczona przez wersje F, wykorzystuje ona praktycznie do maksimum możliwości przyspieszenia robota na odcinku umożliwiającym jazdę po skosie. Wersja E jako jedyna wybrała trasę, która prowadzi przez odcinki nieodwiedzane podczas przejazdów innych wersji. Pomimo tego, że jest ona dłuższa aż o 8 komórek, wciąż jest



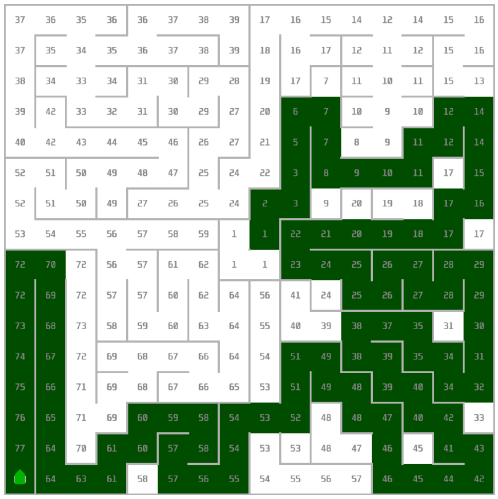
(a) Wersja A



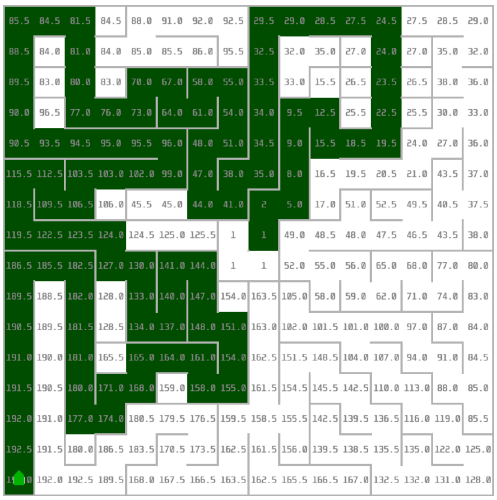
(b) Wersja B



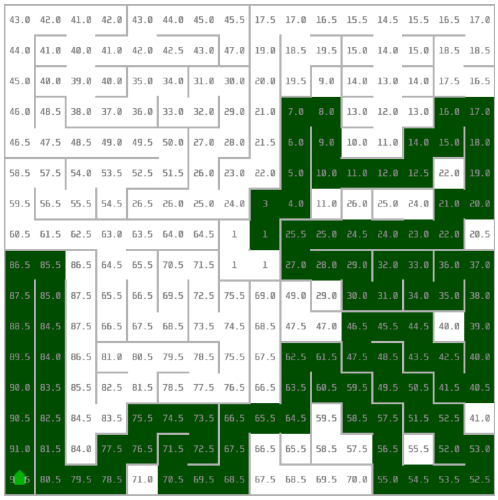
(c) Wersja C



(d) Wersja D

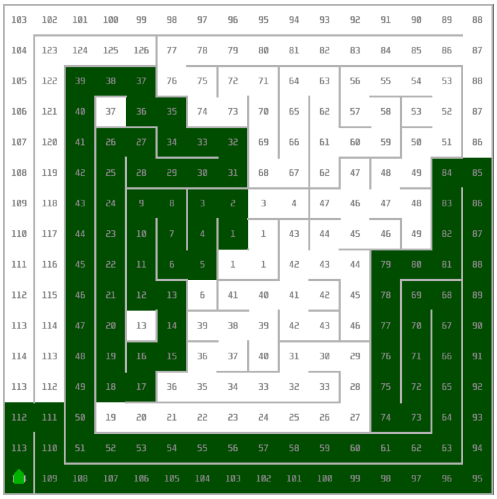


(e) Wersja E

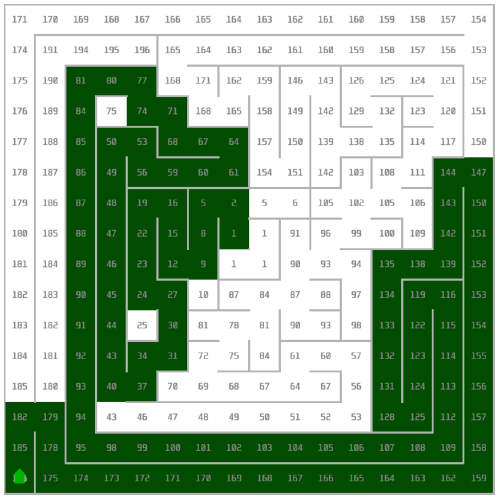


(f) Wersja F

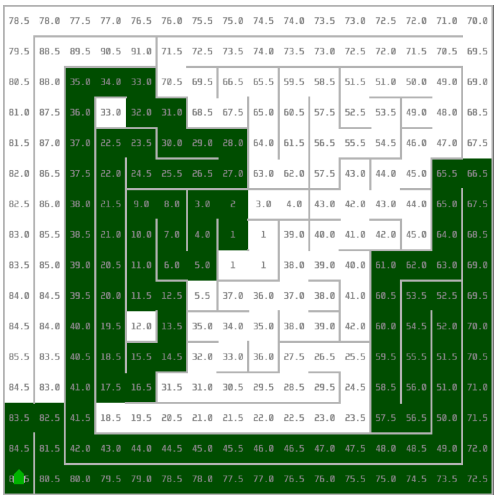
Rysunek 4.6: Porównanie ścieżek dla labiryntu 4



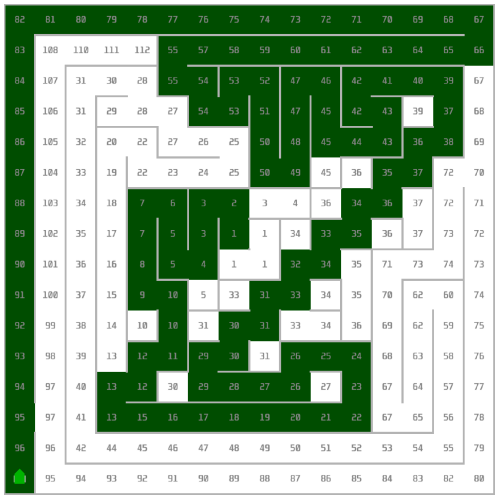
(a) Wersja A



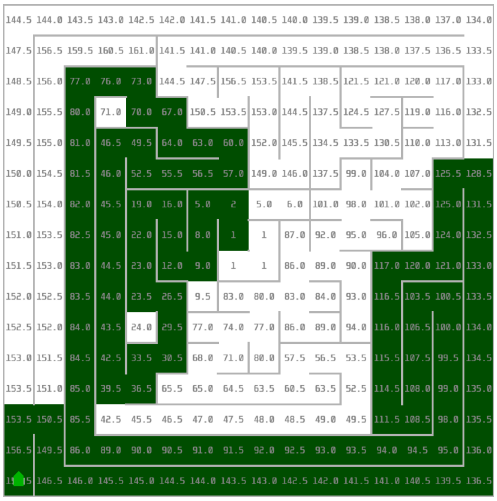
(b) Wersja B



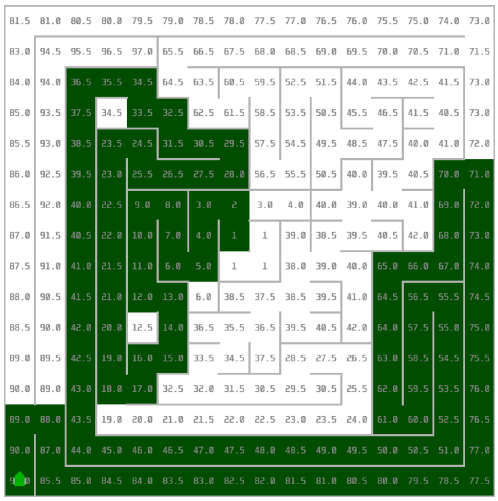
(c) Wersja C



(d) Wersja D

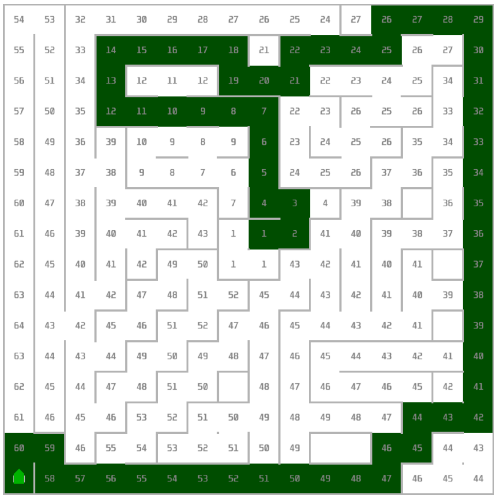


(e) Wersja E

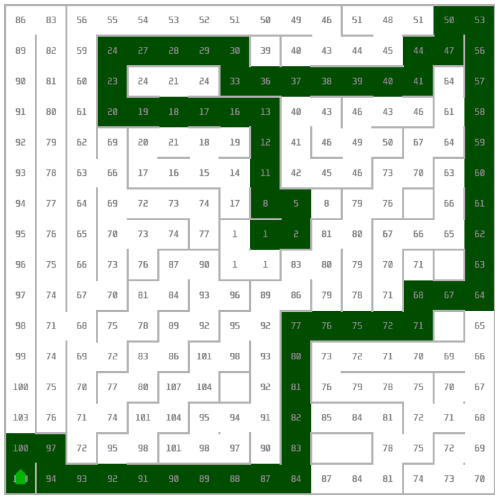


(f) Wersja F

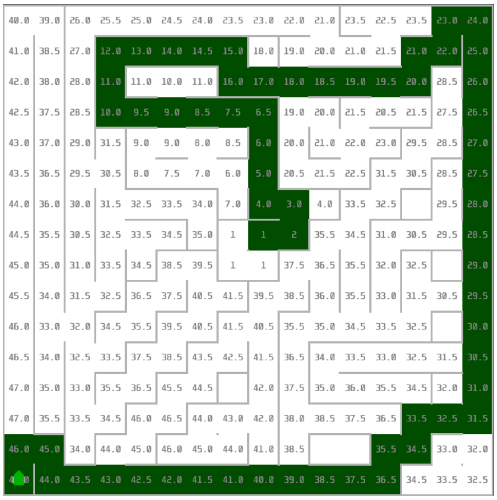
Rysunek 4.7: Porównanie ścieżek dla labiryntu 5



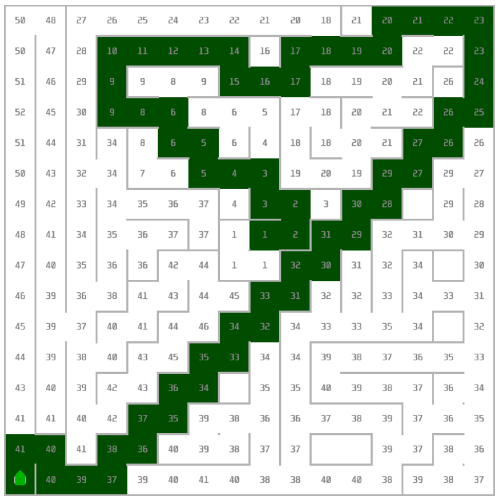
(a) Wersja A



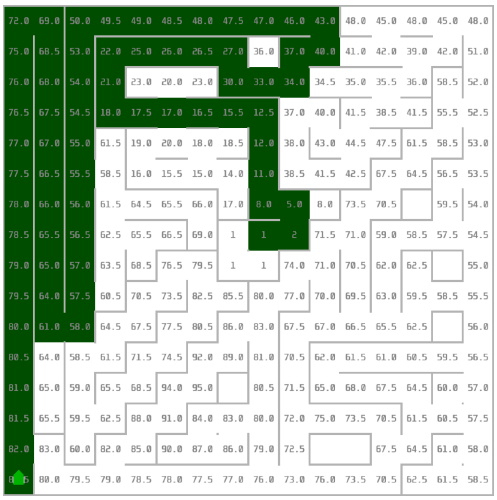
(b) Wersja B



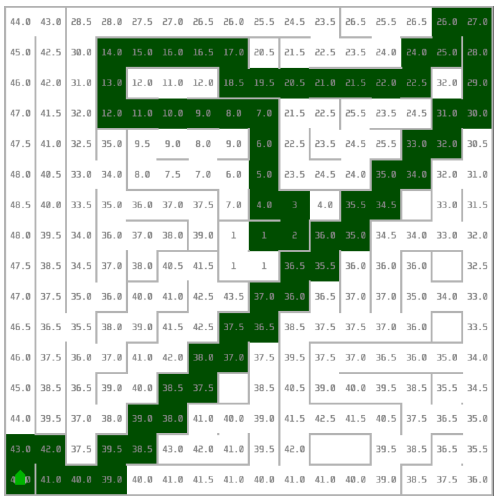
(c) Wersja C



(d) Wersja D



(e) Wersja E



(f) Wersja F

Rysunek 4.8: Porównanie ścieżek dla labiryntu 6

Tabela 4.14: Wyniki dla labiryntu 6

Wersja	Scenariusz 1			Scenariusz 2			Scenariusz 3			Scenariusz 4		
A	60	21	19.51	60	21	11.66	60	19	16.21	60	19	11.33
B	60	21	19.33	60	21	12.68	60	19	16.69	60	19	11.61
C	60	21	20.21	60	21	12.61	60	17	17.33	60	17	12.04
D	60	45	22.99	60	45	17.88	60	19	14.04	60	19	9.92
E	68	17	20.41	68	17	10.43	68	17	18.99	68	16	9.90
F	60	41	22.23	60	41	16.43	60	16	14.31	60	16	9.51

w stanie pokonać labirynt w czasie zaledwie 0.4 sekundy dłuższym w porównaniu do trasy wyznaczonej przez wersję F.

4.3.7 Porównanie wyników

W tabeli 4.15 znajduje się porównanie dla wszystkich labiryntów wpływu możliwości jazdy na ukos tam gdzie było to możliwe. Liczby w tabeli oznaczają różnicę w liczbie zakrętów oraz czasie przejazdów dla scenariusza bez usprawnień i scenariusza wykorzystującego jazdę po skosie. Dla każdego labiryntu niezależnie od wyznaczonej trasy, możliwość jazdy po skosie zmniejszyła czas przejazdu robota. Największe różnice widoczne są dla wersji D we wszystkich labiryntach.

W kolejnej tabeli 4.16 porównane są długości tras i liczby zakrętów, które wyznaczyły poszczególne wersje algorytmu zalewowego oraz ile zajęło robotowi pokonanie ich, kiedy poruszał się zgodnie ze scenariuszem 1. Jak widać w większości przypadków trasy były nieco dłuższe, ale robot pokonywał je szybciej. Wyjątkiem jest tutaj wersja D, która wyznacza nam trasy, które posiadają dużo więcej zakrętów, co znacząco wydłuża czas jazdy.

W tabeli 4.17 przedstawione zostały te same dane tylko dla scenariusza 4. Wyniki dla algorytmów B i C są lepsze tylko dla pierwszego labiryntu, gdzie dla pozostałych są takie same lub gorsze pod względem czasu przejazdu. Wersja D po uwzględnieniu możliwości jazdy po skosie oraz przyspieszania prezentuje się znacznie lepiej w porównaniu do scenariusza 1, wszystkie trasy różniące się od tras wersji A pokonywane są w krótszym czasie.

Zestawienie czasu przejazdów dla scenariusza 4 znajduje się w tabeli 4.18, a w tabeli 4.19 oznaczone zostały wersje, dla których wyznaczone zostały trasy z najkrótszym czasem przejazdu dla danego labiryntu. Najlepsze wyniki otrzymaliśmy dla wersji F, która aż 4 razy wyznaczyła trasę pokonywaną najszybciej. Dla labiryntu 2 i labiryntu 5 algorytm D jako jedyny dał nam inną trasę, gdy pozostałe wersje dawały taki sam rezultat. Obie te trasy okazały się tymi z najkrótszymi czasami przejazdów.

Tabela 4.15: Porównanie liczby zakrętów i czasu przejazdu pomiędzy scenariuszem 1, a scenariuszem 3 dla wszsytkich labiryntów

	Wersja A		Wersja B		Wersja C		Wersja D		Wersja E		Wersja F	
Labirynt 1	2	1.92	2	0.67	2	0.67	10	3.89	2	0.67	2	0.67
Labirynt 2	0	1.89	0	1.89	0	1.89	3	2.72	0	1.89	0	1.89
Labirynt 3	12	6.46	2	3.54	5	3.56	12	6.46	3	3.20	12	5.00
Labirynt 4	16	8.85	4	6.05	4	6.05	16	8.85	4	6.05	16	8.85
Labirynt 5	3	2.19	3	2.19	3	2.19	17	6.97	3	2.19	3	2.19
Labirynt 6	2	3.30	2	2.64	4	2.88	26	8.95	1	1.42	25	7.91

Tabela 4.16: Porównanie wersji B, C, D, E, F względem wersji A, gdy robot porusza się zgodnie ze scenariuszem 1

	Wersja B			Wersja C			Wersja D			Wersja E			Wersja F		
Labirynt 1	0	-6	0.76	2	-6	0.76	2	6	-1.35	2	-6	0.76	2	-6	0.76
Labirynt 2	0	0	0	0	0	0	0	2	-0.34	0	0	0	0	0	0
Labirynt 3	4	-10	0.49	2	-10	0.26	0	0	0	2	-10	1.08	2	-6	0.63
Labirynt 4	2	-8	1.13	2	-8	1.13	0	0	0	2	-8	1.13	0	0	0
Labirynt 5	0	0	0	0	0	0	4	24	-3.94	0	0	0	0	0	0
Labirynt 6	0	0	0.18	0	0	0.10	0	24	-3.48	8	-4	-0.9	0	20	-2.72

Tabela 4.17: Porównanie wersji B, C, D, E, F względem wersji A, gdy robot porusza się zgodnie ze scenariuszem 4

	Wersja B			Wersja C			Wersja D			Wersja E			Wersja F		
Labirynt 1	2	-6	3.16	2	-6	3.16	2	-2	1.6	2	-6	3.16	2	-6	3.16
Labirynt 2	0	0	0	0	0	0	0	-1	0.13	0	0	0	0	0	0
Labirynt 3	4	0	-0.59	2	0	-0.81	0	0	0	2	-1	-0.42	2	-6	1.67
Labirynt 4	2	4	-1.29	2	4	-1.29	0	0	0	2	4	-1.29	0	0	0
Labirynt 5	0	0	0	0	0	0	4	4	0.16	0	0	0	0	0	0
Labirynt 6	0	0	-0.28	0	-2	-0.73	0	0	1.41	8	-3	1.43	0	-3	1.92

Tabela 4.18: Porównanie czasów przejazdów według scenariusza 4

	Wersja A	Wersja B	Wersja C	Wersja D	Wersja E	Wersja F
Labirynt 1	17.01	13.85	13.85	15.41	13.85	13.85
Labirynt 2	7.12	7.12	7.12	6.99	7.12	7.12
Labirynt 3	10.72	11.31	11.53	10.72	11.14	9.05
Labirynt 4	22.70	23.99	23.99	22.70	23.99	22.70
Labirynt 5	19.23	19.23	19.23	19.07	19.23	19.23
Labirynt 6	11.33	11.61	12.04	9.92	9.90	9.51

Tabela 4.19: Wersje dające najkrótszy czas przejazdu dla każdego z labiryntów

	Wersja A	Wersja B	Wersja C	Wersja D	Wersja E	Wersja F
Labirynt 1		×	×		×	×
Labirynt 2				×		
Labirynt 3						×
Labirynt 4	×			×		×
Labirynt 5				×		
Labirynt 6						×
Suma	1	1	1	3	1	4

Rozdział 5

Podsumowanie

Celem pracy było wyznaczenie tras przejazdu robotów klasy micromouse ze startu do mety przy użyciu różnych metod, a także zbadanie wpływu jazdy po skosie na czas przejazdu oraz przeprowadzenie testów tych tras w środowisku symulacyjnym. Opracowane zostały różne metody wyznaczania ścieżek, które następnie poddano testom symulacyjnym w celu oceny, która z nich zapewni robotowi najkrótszy czas przejazdu w zależności od przyjętego scenariusza jazdy. Wyniki testów wykazały, że dla każdego z badanych labiryntów jazda po skosie miała pozytywny wpływ na czas przejazdu. Dla tras wyznaczonych przez wersje priorytetyzujące fragmenty umożliwiające jazdę po skosie, konieczna jest zdolność pokonywania tych fragmentów na ukos. Jeśli robot nie będzie w stanie jechać w ten sposób, nieopłacalny będzie wybór tych tras. Gdy robot jest w stanie poruszać się po skosie oraz przyspieszać na odpowiednich fragmentach najkrótsze czasy przejazdu dla testowanych labiryntów dawała wersja, która priorytetowo traktuje fragmenty składające się z naprzemiennych zakrętów oraz długie proste odcinki. W związku z tym, że po eksploracji labiryntu możliwe jest wykonanie kilku prób przejazdów, najlepszą kolejnością realizacji tras jest: wersja priorytetyzująca fragmenty umożliwiające jazdę po skosie oraz długie proste odcinki, wersja priorytetyzująca fragmenty umożliwiające jazdę po skosie, wersja priorytetyzująca długie proste oraz unikająca zakrętów, podstawowa wersja algorytmu zalewowego, wersja minimalizująca liczbę zakrętów oraz wersja priorytetyzująca długie proste odcinki.

Dalszy rozwój pracy może obejmować kwestie sposobu pokonywania zakrętów przez roboty klasy micromouse. W niniejszej pracy robot musiał zmniejszyć prędkość, wykonać obrót w miejscu, a następnie rozpocząć ruch z zerowej prędkości. Możliwym rozwinięciem tego podejścia jest wykonanie tego samego manewru, ale bez zatrzymywania się — poprzez kontynuowanie ruchu. Innym kierunkiem kontynuacji pracy mogłaby być fizyczna konstrukcja robota, który posłużyłby do testów metod przedstawionych w pracy na rzeczywistych labiryntach, umożliwiając ocenę ich skuteczności w warunkach praktycznych. Realizacja takiego projektu jest jednak trudna, ponieważ oprócz zaimplementowania algorytmów opisanych w pracy, konieczne byłoby stworzenie, oprogramowanie i przetestowanie robota, tak aby był w stanie wykonywać podstawowe czynności, takie jak obrót o zadany kąt, przejazd zadanego dystansu, rozpędzanie się do konkretnych prędkości, czy odczyty z różnych czujników.

Literatura

[1] *Micromouse Book*. Forum Micromouse Online, https://micromouseonline.com/micromouse-book/history/, 2014.

[2] *38. Ogólnojapoński Konkurs Studenckich Robotów Micromouse*. Ogólnojapoński Komitet Wykonawczy Konwencji Micromouse, https://www.ntf.or.jp/student2023/, 2023.

[3] *38. Ogólnojapoński Konkurs Studenckich Robotów Micromouse (Konkurencja Klasyczna)*. Ogólnojapoński Komitet Wykonawczy Konwencji Micromouse, https://www.youtube.com/watch?v=FUqvbnK0wiA&t=0s, 2023.

[4] *MicroMouse Contest*. APEC, https://apec-conf.org/attendees/special-events-social-activities/special-events/, 2023.

[5] AhmadAbbas. *Micro Mouse for Beginners*. Forum internetowe Autodesk Instructables, https://www.instructables.com/Micro-Mouse-for-Beginnersth/.

[6] EastRobo. *Oficjalna strona zawodów*. https://eastrobo.pl/, 2023.

[7] EastRobo. *Regulamin Micromouse*. https://eastrobo.pl/assets/rules/Micromouse%20PL.pdf, 2023.

[8] Peter Harrison. *Claude Shannon made a micromouse first*. Forum Micromouse Online, https://micromouseonline.com/2014/05/17/claude-shannon-made-micromouse-first/, 2014.

[9] Mackrone. *Micromouse Simulator mms*. https://github.com/mackorone/mms, 2022.

[10] marcin13021988. *Roboty MicroMouse — pięć metod przeszukania labiryntu*. Forum internetowe Forbot, https://forbot.pl/blog/roboty-micromouse-5-metod-przeszukiwania-labiryntu-id17354, 2009.

[11] Dawid Perdek. *Badanie własności algorytmów poszukiwania ścieżki w labiryncie dla robota klasy micromouse*. Katedra Cybernetyki i Robotyki, Wydział Elektroniki, Politechnika Wrocławska, https://kcir.pwr.edu.pl/~mucha/Pracki/Dawid_Perdek_praca_inzynierska.pdf, 2015.

[12] Pankaj Bande Swati Mishra. *Maze Solving Algorithms for Micro Mouse*. International Institute of Information Technology Pune, https://swati-mishra.com/wp-content/uploads/2020/02/04725791.pdf, 2008.

[13] Thomashek. *Przeszukiwanie w głąb (DFS)*. Forum internetowe 4programmers, https://4programmers.net/Algorytmy/Przeszukiwanie_w_g%C5%82%C4%85b_(DFS), 2006.

[14] Drew Hall Tondra De. *The Inception of Chedda: A detailed design and analysis of Micromouse*. University of Nevada, Las Vegas, https://www.physics.unlv.edu/~bill/ecg497/Drew_Tondra_report.pdf, 2004.

[15] Koło Naukowe Robotyków „KoNaR". *Strona wydarzenia Robotic Arena*. https://roboticarena.pl/, 2024.

[16] Koło Naukowe Robotyków „KoNaR". *Zasady konkurencji Miromouse*. https://roboticarena.pl/media/competitions/en_micromouse16x16.pdf, 2024.

Spis rysunków

Spis tabel