



POLITECHNIKA WARSZAWSKA

Wydział Mechatroniki

Praca przejściowa

Ireneusz Szulc

Przegląd metod planowania bezkolizyjnych
tras dla zespołu robotów mobilnych

Opiekun pracy:

Warszawa, 2018

Spis treści

Spis treści	2
1 Wstęp	3
1.1 Cel i zakres pracy	3
1.1.1 Założenia	4
1.2 Koordynacja ruchu robotów	5
1.2.1 Podobieństwo do gier RTS	5
1.3 Podstawowe pojęcia	6
2 Metody planowania tras	8
2.1 Metody planowania tras	8
2.1.1 Metoda pól potencjałowych	9
2.1.2 Rozproszone planowanie tras	9
2.1.3 Priorytetowane planowanie	11
3 Algorytmy oparte o A*	12
3.1 Algorytm A*	13
3.1.1 Heurystyki	14
3.2 Local Repair A*	16
3.3 Algorytm D*	16
3.4 Cooperative A*	17
3.4.1 Trzeci wymiar	17
3.4.2 Tablica rezerwacji	18
3.5 Hierarchical Cooperative A*	19
3.6 Windowed Hierarchical Cooperative A*	19
4 Podsumowanie	22
Bibliografia	24

Wykaz skrótów	24
Spis rysunków	25

Rozdział 1

Wstęp

1.1 Cel i zakres pracy

Przedmiotem niniejszej pracy jest przegląd metod wykorzystywanych do planowania bezkolizyjnych tras dla wielu robotów mobilnych. Stanowi to również wstęp teoretyczny do zaprojektowania algorytmu i implementacji oprogramowania pozwalającego na symulację działania skutecznego planowania tras dla systemu wielorobotowego.

Praca skupia się na przypadkach, w których mamy do czynienia ze środowiskiem z dużą liczbą przeszkód (np. zamknięty budynek z licznymi ciasnymi korytarzami), aby uwypuklić problem blokowania się agentów często prowadzący do zakleszczenia. Często okazuje się, że należy wtedy zastosować nieco inne podejścia niż te, które sprawdzają się w przypadku otwartych środowisk, a które zostały opisane np. w pracach [?], [?]. W otwartych środowiskach z małą liczbą przeszkód wystarczające może się okazać np. proste replanowanie wykorzystujące algorytm D* (por. 3.3) lub LRA* (por. 3.2).

W niniejszej pracy starano się znaleźć metody rozwiązuające zagadnienie, w którym znane są:

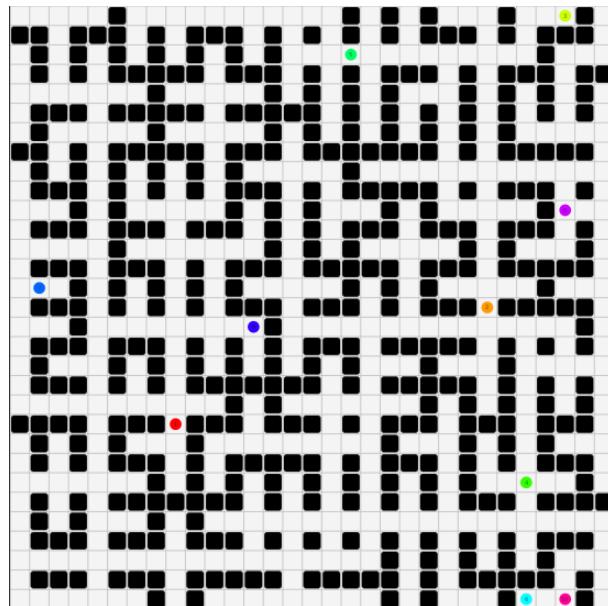
- pełna informacja o mapie otoczenia (położenie statycznych przeszkód),
- aktualne położenie i położenie celu dla każdego z robotów.

Szukany jest natomiast przebieg tras do punktów docelowych dla agentów. Zadaniem algorytmu będzie wyznaczenie możliwie najkrótszej bezkolizyjnej trasy dla wszystkich robotów. Należy jednak zaznaczyć, że priorytetem jest dotarcie każdego z robotów do celu bez kolizji z innymi robotami. Drugorzędne zaś jest, aby wyznaczone drogi były możliwie jak najkrótsze.

1.1.1 Założenia

Założenia i ograniczenia do rozważanego problemu:

1. Każdy z robotów ma wyznaczony inny punkt docelowy, do którego zmierza.
2. Planowanie tras dotyczy robotów holonomicznych.
3. Czas trwania zmiany kierunku robota jest pomijalnie mały.
4. Środowisko, w którym poruszają się roboty, jest dwuwymiarową przestrzenią zawierającą dużą liczbę przeszkód oraz wąskie korytarze (por. rys. 1.1).
5. Roboty "wiedzą" o sobie i mogą komunikować się ze sobą podczas planowania tras.
6. Każdy robot zajmuje w przestrzeni jedno pole. Na jednym polu może znajdować się maksymalnie jeden robot.
7. Planowanie tras powinno odbywać się w czasie rzeczywistym.



Rysunek 1.1: Przykładowe środowisko z dużą liczbą przeszkód i rozmieszczonymi robotami.
Źródło: własna implementacja oprogramowania symulacyjnego

1.2 Koordynacja ruchu robotów

Koordynacja ruchu robotów jest jednym z fundamentalnych problemów w systemach wielorobotowych. [?]

Kooperacyjne znajdowanie tras (ang. *Cooperative Pathfinding*) jest zagadnieniem planowania w układzie wieloagentowym, w którym agenci mają za zadanie znaleźć bezkolizyjne drogi do swoich, osobnych celów. Planowanie to odbywa się w oparciu o pełną informację o środowisku oraz o trasach pozostałych agentów. [?]

Algorytmy do wyznaczania bezkolizyjnych tras dla wielu agentów (robotów) mogą znaleźć zastosowanie w szpitalach (np. roboty TUG i HOMER do dostarczania sprzętu na wyposażeniu szpitala [?]) oraz magazynach (np. roboty transportowe w magazynach firmy Amazon 1.2).



Rysunek 1.2: Roboty Kiva pracujące w magazynie firmy Amazon. Źródło: [?]

1.2.1 Podobieństwo do gier RTS

Problem kooperacyjnego znajdowania tras pojawia się nie tylko w robotyce, ale jest również popularny m.in. w grach komputerowych, gdzie konieczne jest wyznaczanie bezkolizyjnych dróg dla wielu jednostek, unikając wzajemnego blokowania się. Brak wydajnych i skutecznych algorytmów planowania dróg można zauważać w wielu grach typu RTS (*Real-Time Strategy games*), gdzie czasami obserwuje się zjawisko zakleszczenia jednostek w wąskich gardłach (np. Age of Empires II, Warcraft III lub we współczesnych produkcjach) [?] (por. rys. 1.3). Ponadto, zauważalny brak ogólnie dostępnych implementacji lub bibliotek do rozwiązania problemu typu *Cooperative Pathfinding* świadczy o potrzebie rozwoju tych metod.

Często algorytmy wykorzystywane w grach typu RTS zajmują się planowaniem bezkolizyjnych dróg dla układu wielu agentów w czasie rzeczywistym (będącego przedmiotem niniejszej pracy), dlatego nic nie stoi na przeszkodzie, aby stosować je zamiennie również do koordynacji ruchu zespołu robotów mobilnych.

TODO zrobić screen z pożącym zakleszczeniem jednostek



Rysunek 1.3: Popularny problem zakleszczania się jednostek. Źródło: gra komputerowa *Age of Empires II HD*

1.3 Podstawowe pojęcia

Robot holonomiczny

Robot holonomiczny to taki robot mobilny, który może zmienić swoją orientację, stojąc w miejscu.

Przestrzeń konfiguracyjna

Przestrzeń konfiguracyjna to N -wymiarowa przestrzeń będąca zbiorem możliwych stanów danego układu fizycznego. Wymiar przestrzeni zależy od rodzaju i liczby wyróżnionych parametrów stanu. W odróżnieniu od przestrzeni roboczej, gdzie robot ma postać bryły, w przestrzeni konfiguracyjnej robot jest reprezentowany jako punkt.

Zupełność algorytmu (ang. *Completeness*)

W kontekście algorytmu przeszukiwania grafu algorytm zupełny to taki, który gwarantuje znalezienie rozwiązania, jeśli takie istnieje. Warto zaznaczyć, że nie gwarantuje to wcale, że znalezione

rozwiązanie będzie rozwiązaniem optymalnym.

Metoda hill-climbing

Metoda hill-climbing jest rodzajem matematycznej optymalizacji, lokalną metodą przeszukiwania. Jest to iteracyjny algorytm, który zaczyna w wybranym rozwiązaniu problemu, następnie próbuje znaleźć lepsze rozwiązanie poprzez przyrostowe zmiany pojedynczych elementów rozwiązania. Jeśli przyrostowa zmiana przynosi lepsze rozwiązanie, jest ona wprowadzana do nowego rozwiązania. Kroki algorytmu powtarzane są dotąd, aż żadna zmiana nie przynosi już poprawy.

Rozdział 2

Metody planowania tras

2.1 Metody planowania tras

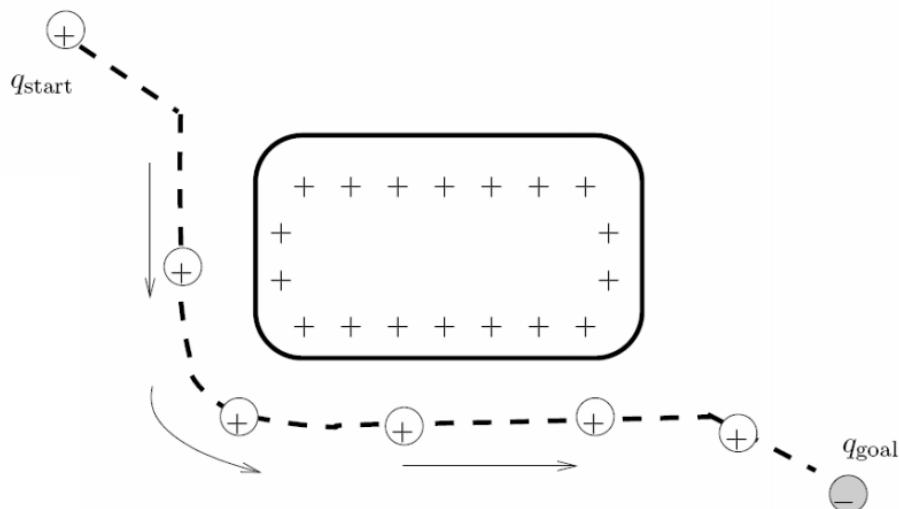
Spośród metod wykorzystywanych do planowania tras dla wielu robotów można wyróżnić dwie zasadnicze grupy [?]:

- **Zcentralizowane** - drogi wyznaczane są dla wszystkich agentów na raz (jednocześnie). Metody tego typu są często trudne do rozwiązania oraz mają bardzo dużą złożoność obliczeniową ze względu na ogólną przestrzeń przeszukiwania. Struktura organizacyjna jest scentralizowana - decyzje podejmowane są na podstawie centralnego systemu.
- **Rozproszone** (ang. *decoupled* lub *distributed*) - Podejście to dekomponuje zadanie na niezależne lub zależne w niewielkim stopniu problemy dla każdego agenta. Dla każdego robota droga wyznaczana jest osobno, w określonej kolejności, następnie rozwiązywane są konflikty (kolizje dróg). Zastosowanie metod rozproszonych wiąże się najczęściej z koniecznością przydzielania priorytetów robotom, co stanowi istotny problem, gdyż od ich wyboru może zależeć zupełność algorytmu. Nie należy mylić tej metody z zagadnieniem typu *Non-Cooperative Pathfinding*, w którym agenci nie mają wiedzy na temat pozostałych planów i muszą przewidywać przyszłe ruchy pozostałych robotów [?]. W podejściu rozproszonym agenci mają pełną informację na temat stanu pozostałych robotów, lecz wyznaczanie dróg odbywa się w określonej kolejności.

W systemach czasu rzeczywistego istotne jest, aby rozwiązanie problemu planowania tras uzyskać w określonym czasie, dlatego w tego typu systemach częściej używane są techniki rozproszone.

2.1.1 Metoda pól potencjałowych

Metoda pól potencjałowych (ang. *Artificial Potential Field* lub *Potential Field Techniques*) polega na zastosowaniu zasad oddziaływania między ładunkami znanych z elektrostatyki. Roboty i przeszkody traktowane są jako ładunki jednoimienne, przez co ”odpychają się” siłą odwrotnie proporcjonalną do kwadratu odległości (dzięki temu unikają kolizji między sobą). Natomiast punkt docelowy robota jest odwzorowany jako ładunek o przeciwnym biegunie, przez co robot jest ”przyciągany” do celu. Główną zasadę działania metody przedstawiono na rysunku 2.1. Technika ta jest bardzo prosta i nie wymaga wykonywania złożonych obliczeń (w odróżnieniu do pozostałych metod zcentralizowanych). Niestety bardzo powszechny jest problem osiągania minimum lokalnego, w którym suma wektorów daje zerową siłę wypadkową. Robot zostaje ”uwięziony” w minimum lokalnym, przez co nie jest w stanie dotrzeć do wyznaczonego celu. Do omijania tego problemu muszą być stosowane inne dodatkowe metody [?]. Metoda pól potencjałowych nie daje gwarancji optymalności (ani nawet zupełności).



Rysunek 2.1: Zasada działania metody pól potencjałowych. Źródło: [?]

2.1.2 Rozproszone planowanie tras

W celu przyspieszenia obliczeń w algorytmach przeszukujących grafy (np. A* i jego pochodne), nawet w przypadku ciągłej przestrzeni mapy stosuje się podział na dyskretną siatkę pól (por. rys. 2.2) [?].



Rysunek 2.2: Ciągła przestrzeń mapy zdyskretyzowana do siatki pól, Źródło: edytor map z gry Warcraft III.

Popularne podejścia unikające planowania w wysoko wymiarowej zbiorowej przestrzeni konfiguracyjnej to techniki rozproszone i priorytetowane. Pomimo, że metody te są bardzo efektywne, mają dwie główne wady:

- Nie są zupełne - nie dają gwarancji znalezienia rozwiązania, nawet gdy takie istnieje.
- Wynikowe rozwiązania mogą być nieoptymalne.

W artykule [?] przedstawione zostało podejście do optymalizowania układu priorytetów dla rozproszonych i priorytetowanych technik planowania. Proponowana metoda wykonuje randomizowane przeszukiwanie z techniką hill-climbing do znalezienia rozwiązania i do skrócenia całkowitej długości ścieżek. Technika ta została zaimplementowana i przetestowana na prawdziwych robotach oraz w rozległych testach symulacyjnych.

Najczęściej stosowanymi podejściami są metody oparte o algorytm A*.

Path coordination

Idea metody Path coordination przedstawia się następująco [?]:

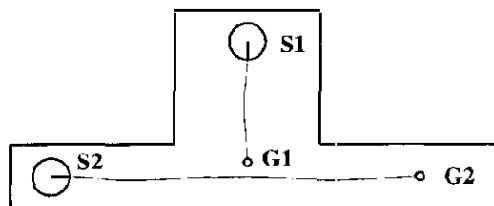
1. Wyznaczenie ścieżki dla każdego robota **niezależnie** (np. za pomocą algorytmu A*)
2. Przydział priorytetów
3. Próba rozwiązania możliwych konfliktów między ścieżkami. Roboty utrzymywane są na ich indywidualnych ścieżkach (wyznaczonych na początku), wprowadzane modyfikacje pozwalają na zatrzymanie się, ruch naprzód, a nawet cofanie się, ale tylko **wzdłuż trajektorii** w celu uniknięcia kolizji z robotem o wyższym priorytecie.

2.1.3 Priorytetowane planowanie

Często używaną w praktyce metodą jest planowanie priorytetowane. W tej technice agenci otrzymują unikalne priorytety. Algorytm wykonuje indywidualne planowanie sekwencyjnie dla każdego agenta w kolejności od najwyższego priorytetu. Trajektorie agentów o wyższych priorytetach są ograniczeniami (ruchomymi przeszkodami) dla pozostałych agentów [?].

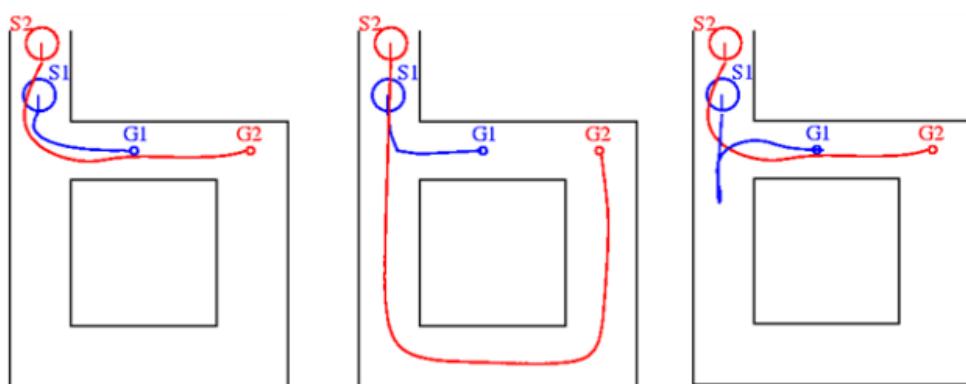
Złożoność ogólnego algorytmu rośnie liniowo wraz z liczbą agentów, dzięki temu to podejście ma zastosowanie w problemach z dużą liczbą agentów. Wyraźnie widać, że algorytm ten jest zachłanny i niezupełny w takim znaczeniu, że agentów zadowala pierwsza znaleziona trajektoria niekolidująca z agentami wyższych priorytetów.

Istotną rolę doboru priorytetów robotów w procesie planowania tras ukazuje prosty przykład przedstawiony na rysunku 2.3. Jeśli robot 1 (zmierzający z punktu S1 do G1) otrzyma wyższy priorytet niż robot 2 (zmierzający z S2 do G2), spowoduje to zablokowanie przejazdu dla robota 2 i w efekcie prawidłowe rozwiązanie nie zostanie znalezione.



Rysunek 2.3: Sytuacja, w której żadne rozwiązanie nie zostanie znalezione, jeśli robot 1 ma wyższy priorytet niż robot 2. Źródło: [?]

Układ priorytetów może mieć także wpływ na długość uzyskanych tras. Potwierdzający przykład został przedstawiony na rysunku 2.4. W zależności od wyboru priorytetów, wpływających na kolejność planowania tras, otrzymujemy różne rozwiązania.



Rysunek 2.4: a) Niezależne planowanie optymalnych tras dla 2 robotów; b) suboptimalne rozwiązanie, gdy robot 1 ma wyższy priorytet; c) rozwiązanie, gdy robot 2 ma wyższy priorytet. Źródło: [?]

Rozdział 3

Algorytmy oparte o A*

Kiedy pojedynczy agent dokonuje znalezienia drogi do wyznaczonego celu, prosty algorytm A* sprawdza się bardzo dobrze. Jednak w przypadku, gdy wiele agentów porusza się w tym samym czasie, to podejście może się nie sprawdzić, powodując wzajemne blokowanie się i zakleszczenie jednostek. Rozwiązaniem tego problemu może być kooperacyjne znajdowanie tras. Roboty będą mogły skutecznie przemieszczać się przez mapę, omijając trasy wyznaczone przez inne jednostki oraz schodząc innym jednostkom z drogi, jeśli to konieczne. [?]

Zagadnienie znajdowania drogi jest ważnym elementem sztucznej inteligencji zaimplementowanej w wielu grach komputerowych. Chociaż klasyczny algorytm A* potrafi doprowadzić pojedynczego agenta do celu, to jednak dla wielu agentów wymagane jest zastosowanie innego podejścia w celu unikania kolizji. Algorytm A* może zostać zaadaptowany do replanowania trasy na żądanie, w przypadku wykrycia kolizji tras (Local Repair A* lub D*). Jednak takie podejście nie jest zadowalające pod wieloma względami. Na trudnych mapach z wieloma wąskimi korytarzami i wieloma agentami może to prowadzić do zakleszczenia agentów w wąskich gardłach lub do cyklicznego zapętlenia ruchu agentów. [?]

TODO Które algorytmy tylko znajdują drogi, a które rozwiązują problem kolizji

TODO Większość z tych algorytmów wykorzystywana jest w grach. Planowanie musi odbywać się szybko, szczególnie w grach (screen warcraft :))

TODO Poniższe podrozdziały mają na celu wprowadzanie kolejnych modyfikacji i dotarcie do WHCA* *TODO* bedzie to intro przez metody, aby dotrzeć wreszcie do WHCA*

TODO Silver jest sponsorem tego odcinka

TODO Clearance-based Pathfinding and Hierarchical Annotated A* Search - używane, gdy jednostki zajmują różne obszary, ale chyba tylko dla jednej jednostki (nie cooperative)

3.1 Algorytm A*

A* jest algorytmem heurystycznym służącym do przeszukiwania grafu w celu znalezienia najkrótszej ścieżki. Algorytm ten jest powszechnie stosowany w zagadnieniach sztucznej inteligencji oraz w grach komputerowych [?]. Jest modyfikacją algorytmu Dijkstry, wprowadza pojęcie funkcji heurystycznej $h(n)$. Wartość funkcji heurystycznej powinna określać przewidywaną drogę do węzła docelowego. W każdym kroku przeszukiwany jest węzeł o najmniejszej wartości funkcji $f(n)$.

TODO jest optymalny, ale w pewnych warunkach

$$f(n) = g(n) + h(n) \quad (3.1)$$

gdzie:

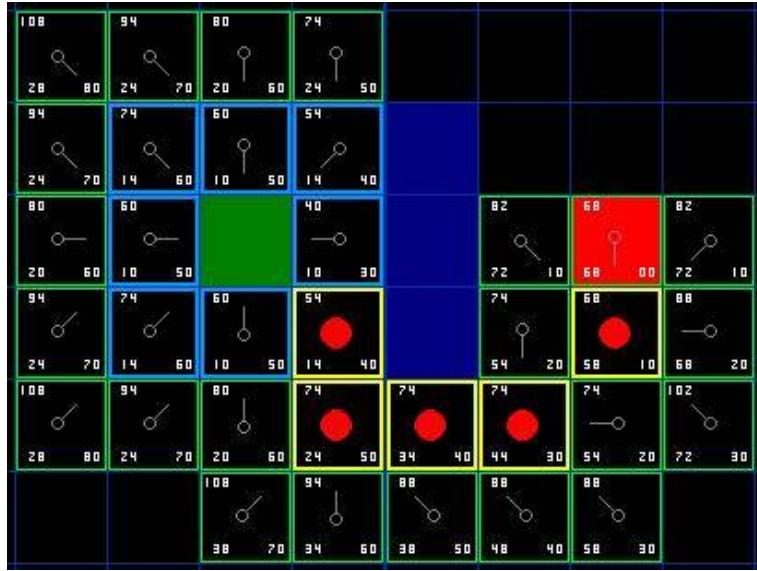
$g(n)$ - dotychczasowy koszt dotarcia do węzła n - dokładna odległość między węzłem n a węzłem początkowym

$h(n)$ - heurystyka, przewidywana pozostała droga od węzła bieżącego do węzła docelowego

$f(n)$ - oszacowanie pełnego kosztu ścieżki od węzła startowego do węzła docelowego prowadzącej przez węzeł n

n - bieżący węzeł, wierzchołek przeszukiwanego grafu

Dzięki takiemu podejściu najpierw sprawdzane są najbardziej "obiecujące" rozwiązania, co pozwala szybciej otrzymać rozwiązanie (w przeciwieństwie do algorytmu Dijkstry). Algorytm kończy działanie w momencie, gdy napotka węzeł będący węzłem docelowym. Dla każdego odwiedzonego węzła zapamiętywane są wartości $g(n)$, $h(n)$ oraz węzeł będący rodzicem w celu późniejszego odnalezienia drogi powrotnej do węzła startowego po napotkaniu węzła docelowego (por. rys. 3.1).



Rysunek 3.1: Przykład działania algorytmu A*. Każdy odwiedzony węzeł wskazuje na swojego rodzica, co umożliwia późniejszą rekonstrukcję drogi. Źródło: [?]

3.1.1 Heurystyki

Od wyboru sposobu obliczania heurystyki zależy czas wykonywania algorytmu oraz optymalność wyznaczonego rozwiązania. Poniżej przedstawiono najczęściej wykorzystywane heurystyki.

Funkcja heurystyczna $h(n)$ jest dopuszczalna, jeśli dla dowolnego węzła n spełniony jest warunek $h(n) \leq h^*(n)$ gdzie:

$h^*(n)$ - rzeczywisty koszt ścieżki od węzła n do celu

Przeszukiwanie A* bez eliminacji powtarzających się stanów przy użyciu heurystyki dopuszczalnej znajduje zawsze rozwiązanie optymalne.

TODO Jeśli wartość heurystyki \hat{h} = od rzeczywistej drogi, jaka musi zostać przebyta, to wyznaczona ścieżka jest ścieżką optymalną (najkrótszą) = admissible heuristic An admissible heuristic never overestimates the distance to the goal. A* search with an admissible heuristic is guaranteed to find the shortest path. However, there is a stronger property, known as consistency [Russell03]. A consistent heuristic maintains the property $h(A) \leq cost(A, B) + h(B)$ between all locations A and B. In other words, the estimated distance doesn't jump around between the locations along a path. [?] Heurystyka spójna (gdy dla każdego węzła A). Każda heurystyka spójna jest dopuszczalna.

A* opiera się na heurystyce, która prowadzi przeszukiwaniem. Słaba heurystyka może prowadzić do zbędnego odwiedzania dodatkowych węzłów.

Przeszukiwanie w A* kontroluje heurystyka. To od niej zależy, w kierunku których węzłów będzie podążało przeszukiwanie.

Heurystyka Euklidesowa

Dla dwuwymiarowej mapy heurystyka wykorzystująca metrykę euklidesową wyraża dokładną odległość między przeszukiwanym węzłem (x_n, y_n) a węzłem docelowym (x_g, y_g) :

$$h(n) = \sqrt{(x_n - x_g)^2 + (y_n - y_g)^2} \quad (3.2)$$

Heurystyka Manhattan

W przypadku, gdy robot może poruszać się po mapie jedynie poziomo lub pionowo (nie na ukos) wystarczającym przybliżeniem jest metryka Manhattan:

$$h(n) = |x_n - x_g| + |y_n - y_g| \quad (3.3)$$

TODO Przykład z Pacmanem - screen - siatka podzielona na pola



Rysunek 3.2: Mapa z zaznaczeniem podziału na siatkę pól, zastosowanie heurystyki uwzględniającej zawijanie mapy po bokach, Źródło: własna implementacja gry PacMan

zbliżenie do 0 sprowadza ją do ścieżki optymalnej (Dijkstry) - połowa heurystyki Manhattan daje możliwość wyznaczania trasy z uwzględnieniem zawijania mapy The Manhattan distance is a consistent heuristic. The true distance heuristic is also consistent. (no chyba że zawijanie mapy)

Heurystyka zerowa

Przyjęcie heurystyki równej $h(n) = 0$ powoduje, że algorytm A* sprowadza się do algorytmu Dijkstry.

3.2 Local Repair A*

W algorytmie Local Repair A* (LRA*) każdy z agentów znajduje drogę do celu, używając algorytmu A*, ignorując pozostałe roboty oprócz ich obecnych sąsiadów. Roboty zaczynają podążać wyznaczonymi ścieżkami do momentu, aż kolizja z innym robotem jest nieuchronna. Wtedy następuje ponowne przeliczenie drogi pozostałej do przebycia, z uwzględnieniem nowo napotkanej przeszkody.

Możliwe (i całkiem powszechnie [?]) jest uzyskanie cykli (tych samych sekwencji ruchów powtarzających się w nieskończoność), dlatego zazwyczaj wprowadzane są pewne modyfikacje, aby rozwiązać ten problem. Jedną z możliwości jest zwiększenie wpływu losowego szumu na wartość heurystyki. Kiedy agenci zachowują się bardziej losowo, prawdopodobne jest, że wydostaną się z problematycznego położenia i spróbują podążać innymi ścieżkami.

Algorytm ten ma jednak sporo poważnych wad, które szczególnie ujawniają się w trudnych środowiskach z dużą liczbą przeszkód. Wydostanie się z zatłoczonego wąskiego gardła może trwać bardzo długo. Prowadzi to również do ponownego przeliczania trasy w prawie każdym kroku. Wciąż możliwe jest również odwiedzanie tych samych lokalizacji w wyniku zapętleń.

TODO Jest to przykład rozproszonej struktury organizacyjnej - Podejmowanie decyzji przez roboty lokalnie (+ centralnie)???

3.3 Algorytm D*

D* (*Dynamic A* Search*) jest przyrostowym algorytmem przeszukiwania. Jest modyfikacją algorytmu A* pozwalającą na szybsze replanowanie trasy w wyniku zmiany otoczenia (np. zajmowania wolnego pola przez innego robota). Wykorzystywany jest m.in. w nawigacji robota do określonego celu w nieznanym terenie. Początkowo robot planuje drogę na podstawie pewnych założeń (np. nieznany teren nie zawiera przeszkód). Podążając wyznaczoną ścieżką, robot odkrywa rzeczywisty stan mapy i jeśli to konieczne, wykonuje ponowne planowanie trasy na podstawie nowych informacji. Często wykorzystywana implementacja (z uwagi na zoptymalizowaną złożoność obliczeniową) jest wariant algorytmu *D* Lite* [?].

3.4 Cooperative A*

TODO Cooperative A* jest algorytmem do rozwiązywania problemu kooperacyjnego znajdowania tras. Metoda może być również nazywana czasoprzestrzennym algorytmem A* (*time-space A* search*) Zadanie planowania jest rozdzielone na serię pojedynczych poszukiwań dla poszczególnych agentów. Pojedyncze poszukiwania są wykonywane w trójwymiarowej czasoprzestrzeni i biorą pod uwagę zaplanowane ścieżki przez pozostałych agentów. Akcja wykonania postoju (pozostania w tym samym miejscu) jest uwzględniona w zbiorze akcji możliwych do wykonania. Po przeliczeniu dróg dla każdego agenta, stany zajętości pól są zaznaczane w tablicy rezerwacji (ang. Reservation table). Pozycje w tej tablicy są uważane jako pola nieprzejezdne i w efekcie są omijane podczas przeszukiwania przez późniejszych agentów. [?]

Należy zaznaczyć, że planowanie dla każdego agenta odbywa się sekwencyjnie według przydzielonych priorytetów. Algorytm ten jest podatny na zmianę kolejności agentów. Odpowiedni dobór priorytetów może wpłynąć na wydajność algorytmu oraz jakość uzyskanego wyniku.

3.4.1 Trzeci wymiar

TODO Do rozwiązywania problemu kooperacyjnego znajdowania dróg algorytm przeszukiwania potrzebuje mieć pełną wiedzę na temat przeszkód oraz jednostek na mapie. Aby zapisać tą informację potrzeba rozszerzyć mapę o trzeci wymiar - czas. Pierwotną mapę będziemy nazywać mapą przestrenną, natomiast nową - czasoprzestrenną mapą. [?]

Zagadnienie sprowadza się do przeszukiwania grafu, w którym każdy węzeł ma przypisane 3 wielkości: położenie x, położenie y oraz czas. Podczas gdy zakres wielkości x i y jest znany i wynika z rozmiarów mapy oraz podziału jej wymiarów na dyskretne pola, to jednak określenie wymiaru czasu może być trudnym zagadnieniem. Wymiar czasu możemy również zdyskretyzować i przyjąć, że najmniejsza jednostka jest czasem, jaki zajmuje robotowi przejście z jednego pola na sąsiednie (poziomo lub pionowo). Natomiast górną granicą czasu powinna być maksymalna liczba ruchów potrzebna do dotarcia do celu przez ostatniego robota. Wybór za małej liczby może spowodować, że algorytm nie zdąży znaleźć drogi dla niektórych agentów, z kolei za duża granica czasu mocno wydłuża czas obliczeń. Rozwiążanie tego problemu zostało opisane w późniejszym podrozdziale 3.6.

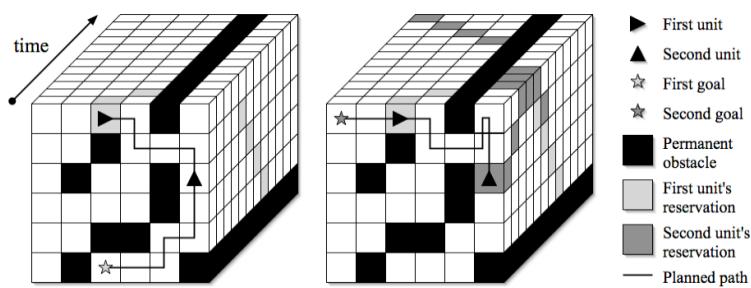
Wprowadzenie trzeciego wymiaru powoduje również konieczność zmian w doborze odpowiedniej heurystyki odpowiedzialnej za oszacowanie drogi pozostałe do celu.

3.4.2 Tablica rezerwacji

Tablica rezerwacji (ang. *Reservation Table*) reprezentuje współdzieloną wiedzę o zaplanowanych ścieżkach przez wszystkich agentów. Jest to informacja o zajętości każdej z komórk na mapie w danym miejscu i określonym czasie. [?]

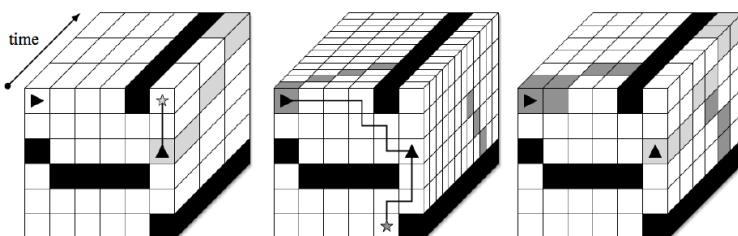
Jak tylko agent zaplanuje trasę, każda komórka odpowiadająca ścieżce zaznaczana jest jako zajęta w tablicy rezerwacji.

W prostej implementacji tablica rezerwacji jest trójwymiarową kostką (dwa wymiary przestrzenne i jeden wymiar czasu). Każda komórka kostki, która jest przecinana przez zaplanowaną przez agenta ścieżkę, jest zaznaczana jako nieprzejezdna przez określony czas trwania. W ten sposób zapobiega to planowania kolizyjnych tras przez pozostałych agentów.



Rysunek 3.3: Dwie jednostki kooperacyjnie poszukujące tras. (A) Pierwsza jednostka znajduje ścieżkę i zaznacza ją w tablicy rezerwacji. (B) Druga jednostka znajduje ścieżkę, uwzględniając istniejące rezerwacje pól, również zaznaczając ją w tablicy rezerwacji. Źródło: [?]

W ogólności poszczególni agenci mogą mieć różną prędkość lub rozmiary, zatem tablica rezerwacji musi mieć możliwość zaznaczenia dowolnego zajętego obszaru. Zostało to przedstawione na rysunku 3.4.



Rysunek 3.4: Czasoprzestrzenna mapa może różnić się od tablicy rezerwacji. (A) Powolna jednostka ma "głębokie" komórki na czasoprzestrzennej mapie. (B) Szybka jednostka ma "pływkie" komórki. (C) Tablica rezerwacji jest współdzielona między wszystkimi agentami, dlatego powinna być odpowiednio dopasowana do wszystkich agentów. Źródło: [?]

Jeśli tylko mała część z całej tablicy rezerwacji będzie markowana jako zajęta, wydajniej jest zaimplementować ją jako tablicę typu *hash table*. Daje to zaletę oszczędności pamięci poprzez pamiętanie jedynie współrzędnych (x, y, t) zajętych pól.

Niestety taki sposób wykorzystania tablicy rezerwacji w pewnych sytuacjach nie zapobiega zderzeniom czołowym jednostek zmierzających w przeciwnych kierunkach. Jeśli jedna jednostka zarezerwowała komórki (x, y, t) i $(x + 1, y, t + 1)$, nic nie stoi na przeszkodzie, aby kolejna jednostka mogła zarezerwować komórki $(x + 1, y, t)$ i $(x, y, t + 1)$. Ten problem może być rozwiązyany poprzez zajmowanie (rezerwowanie) dwóch sąsiednich komórek w tym samym czasie t podczas ruchu robota.

3.5 Hierarchical Cooperative A*

Metoda *Hierarchical Cooperative A** (HCA*) wprowadza pewną modyfikację do algorytmu Cooperative A*. Modyfikacja ta dotyczy heurystyki opartej na abstrakcjach przestrzeni stanu. W tym podejściu abstrakcyjne odległości są obliczane na żądanie. Hierarchia w tym przypadku dotyczy serii abstrakcji przestrzeni stanu, każdej bardziej ogólnej od poprzedniej. [?] HCA* jest również jednym z przykładów rozproszonego podejścia do planowania tras.

Podejście (HCA*) wykorzystuje prostą hierarchię zawierającą pojedynczą abstrakcję dziedziny, która ignoruje wymiar czasu, jak również tablicę rezerwacji. Innymi słowy, abstrakcja jest prostą dwuwymiarową mapą z usuniętymi agentami. Abstrakcyjne odległości mogą być rozumiane jako dokładne oszacowania odległości do celu, ignorując potencjalne interakcje z innymi agentami. Jest to oczywiście dopuszczalna i spójna heurystyka. Niedokładność heurystyki wynika jedynie z trudności związanych z interakcją z innymi agentami (jak bardzo agent musi zboczyć z pierwotnie zaplanowanej ścieżki w celu ominięcia innych agentów).

Opisywane podejście wykorzystuje algorytm przeszukiwania Reverse Resumable A* (RRA*) dla abstrakcyjnej dziedziny. Algorytm ten wykonuje zmodyfikowane przeszukiwanie A* w odwrotnym kierunku. Przeszukiwanie zaczyna się w węźle docelowym agenta i kieruje się do początkowego położenia. Jednak zamiast kończyć w tym punkcie, przeszukiwanie jest kontynuowane do natrafienia na węzeł N , w którym znajduje się agent.

Algorytm HCA* jest więc taki, jak algorytm CA* z bardziej wyszukaną heurystyką, która używa RRA* do obliczania abstrakcyjnych odległości na żądanie.

3.6 Windowed Hierarchical Cooperative A*

Problematyczne zagadnienie związane z wyżej wspomnianymi algorytmami jest takie, że kończą one działanie w momencie, gdy agent osiąga swój cel. Jeśli agent znajduje się już w miejscu docelowym, np. w wąskim korytarzu, to może on blokować części mapy dla innych agentów. W takiej sytuacji agenci powinni kontynuować kooperację z pozostałymi jednostkami, nawet po osiągnięciu swoich celów. Może to zostać zrealizowane np. poprzez usunięcie się z wąskiego

gardła w celu przepuszczenia pozostałych agentów, a następnie powrót do docelowego punktu. [?]

Kolejny problem związany jest z wrażliwością na kolejność agentów (przydzielone priorytety). Chociaż czasem możliwe jest skuteczny, globalny przydział priorytetów [?], to jednak dobrym rozwiązaniem może być dynamiczne modyfikowanie kolejności agentów. Wtedy rozwiązania mogą zostać znalezione w tych przypadkach, w których zawiodło przydzielanie niezmiennych priorytetów. [?]

Rozwiązaniem powyższych kwestii jest zamknięcie algorytmu przeszukiwania w oknach czasowych. Kooperacyjne planowanie jest ograniczone do ustalonej głębokości. Każdy agent szuka częściowej ścieżki do celu i zaczyna nią podążać. W regularnych okresach (np. gdy agent jest w połowie drogi) okno jest przesuwane naprzód i wyznaczana jest nowa ścieżka.

Aby zapewnić, że agenci podążają do prawidłowych punktów docelowych, ograniczana jest tylko głębokość przeszukiwania kooperacyjnego (związanego z wieloma agentami), podczas gdy przeszukiwanie abstrakcyjnych odległości (heurystyki opisanej w podrozdziale 3.5) odbywa się bez ograniczeń głębokości. Okno o rozmiarze w może być rozumiane jako pośrednia abstrakcja, która jest równoważna wykonaniu w kroków w rzeczywistym środowisku (z uwzględnieniem pozostałych agentów) a następnie wykonaniu pozostałych kroków zgodnie z abstrakcją (bez uwzględnienia innych agentów). Innymi słowy, pozostali agenci są jedynie rozważani dla w pierwszych kroków (poprzez tablicę rezerwacji) a dla pozostałych kroków są ignorowani [?].

Rozmiar okna jest wielkością ustalaną arbitralnie. Dobrą praktyką jest przyjęcie wartości równej liczbie agentów na mapie, gdyż to właśnie z ich powodów mogą wystąpić ewentualne zmiany w planowaniu tras.

Porównanie HCA* i WHCA*

Algorytm HCA* wybiera ustaloną kolejność agentów i planuje trasy dla każdego agenta po kolei, unikając kolizji z poprzednio wyznaczonymi ścieżkami. Natomiast użycie przeszukiwania z przesuwanym oknem w WHCA* poprawia skuteczność algorytmu oraz przyspiesza proces wyznaczania rozwiązania [?].

Fakt wykonywania przeszukiwania w oknie oznacza, że planowanie algorytmem WHCA* wykonywane jest zawsze z ustaloną liczbą kroków w przyszłości i wybierany jest najbardziej obiecujący węzeł na granicy tego okna [?]. W metodach Hierarchical Cooperative A* oraz Cooperative A* wybór granicy wymiaru czasu (głębokości przeszukiwania w liczbie kroków) stanowi balans pomiędzy wydajnością a zupełnością algorytmu.

W obu podejściach HCA* i WHCA* zastosowano dodatkowy algorytm przeszukiwania wstecz (RRA*) wspomagający heurystykę. Służy on wyznaczeniu dokładnej odległości z węzła

do celu, pomijając wpływ innych agentów. Jest to często heurystyka wysokiej jakości (bardzo dobre oszacowanie), gorzej sprawdza się jedynie w środowiskach z dużą ilością wąskich gardeł i zatorów. [?]

Chociaż takie przeszukiwanie wstecz prowadzi do początkowego wykonania większej ilości obliczeń (wykonanie pełnego przeszukania A* z punktu docelowego do punktu startowego, jak również do innych węzłów), to jednak koszt obliczeniowy w kolejnych krokach jest już zdecydowanie niższy. [?]

Rozdział 4

Podsumowanie

TODO Większość algorytmów Cooperative Pathfinding opiera się o A*

Algorytmy wprowadzają ograniczenie (błędne założenie), że ruchy trwają tyle samo. Można robić inaczej: podzielić dyskretnie i zaznaczać zajętość pól w wielu kratkach - ale wtedy będzie więcej obliczeń.

Windowed Hierarchical Cooperative A*: • Cooperative A* • Hierarchical Heuristic • Windowed cooperation

The cooperative pathfinding methods are more successful and find higher quality routes than A* with Local Repair. Unfortunately, the basic CA* algorithm is costly to compute,

The size of the window has a significant effect on the success and performance of the algorithm. With a large window, WHCA* behaves more like HCA* and the initialisation time increases. If the window is small, WHCA* behaves more like Local Repair A*. The success rate goes down and the path length increases. The window size parameter thus provides a spectrum between Local Repair A* and HCA*. An intermediate choice appears to give the most robust overall performance. In general, the window size should be set to the duration of the longest predicted bottleneck. In Real-Time Strategy games groups of units are often moved together towards a common destination. In this case the maximum bottleneck time with cooperative pathfinding (ignoring units in other groups) is the number of units in the group. If the window size is lower than this threshold, bottleneck situations may not be resolved well. If the window size is higher, then some redundant search will be performed.

Local Repair A* may be an adequate solution for simple environments with few bottlenecks and few agents. With more difficult environments, Local Repair A* is inadequate and is significantly outperformed by Cooperative A* algorithms.

By introducing Hierarchical A* to improve the heuristic and using windowing to shorten the search, a robust and efficient algorithm, WHCA*, was developed. WHCA* finds more successful routes than Local Repair A*, with shorter paths and fewer cycles.

Although this research was restricted to grid environments, the algorithms presented here apply equally to more general pathfinding domains. Any continuous environment or navigational mesh can be used, so long as each agent's route can be planned by discrete motion elements. The grid-based reservation table is generally applicable, but reserving mesh intersections may be more appropriate in some cases. Finally, the cooperative algorithms may be applied in dynamic environments, so long as an agent's route is recomputed whenever invalidated by a change to the map.

Cooperative pathfinding is a general technique for coordinating the paths of many units. It is appropriate whenever there are many units on the same side who are able to communicate their paths. By planning ahead in time as well as space, units can get out of each other's way and avoid any conflicting routes.

Many of the usual enhancements to spatial A* can also be applied to space-time A*. Moreover, the time dimension gives a whole new set of opportunities for pathfinding algorithms to explore.

Bibliografia

- [1] Bennewitz M.; Burgard W.; Thrun S. *Optimizing Schedules for Prioritized Path Planning of Multi-Robot Systems*. 2001.
- [2] Hart P. E.; Nilsson N. J.; Raphael B. *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*. IEEE Transactions on Systems Science and Cybernetics SSC4, 1968.
- [3] Koenig S.; Likhachev M. *D* Lite*. Proceedings of the AAAI Conference of Artificial Intelligence, 2002.
- [4] Latombe J. *Robot Motion Planning*. Boston, MA: Kluwer Academic, 1991.
- [5] Mówinski K.; Roszkowska E. *Sterowanie hybrydowe ruchem robotów mobilnych w systemach wielorobotycznych*. Postępy Robotyki, 2016.
- [6] Siemiątkowska B. *Uniwersalna metoda modelowania zachowań robota mobilnego wykorzystująca architekturę uogólnionych sieci komórkowych*. 2009.
- [7] Silver D. *Cooperative Pathfinding*. Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference, 2005.
- [8] Zhu Q.; Yan Y.; Xing Z. *Robot Path Planning Based on Artificial Potential Field Approach with Simulated Annealing*. Intelligent Systems Design and Applications, 2006.
- [9] Standley T.; Korf R. *Complete Algorithms for Cooperative Pathfinding Problems*. Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, 2011.
- [10] Toresson A. *Real-time Cooperative Pathfinding*. 2010.
- [11] Narendra S. Chaudhari Le Minh Duc, Amandeep Singh Sidhu. *Hierarchical Pathfinding and AI-Based Learning Approach in Strategy Game Design*. International Journal of Computer Games Technology, 2008.

- [12] P; Vokrinek J.; Pechoucek M. Cap, M; Novak. *Asynchronous Decentralized Algorithm for Space-Time Cooperative Pathfinding*. Workshop Proceedings of the European Conference on Artificial Intelligence (ECAI 2012), 2012.
- [13] Chubak P. Geramifard A. *Efficient Cooperative Path-Planning*. Computing Science Department, University of Alberta, 2005.
- [14] . . .
- [15] A* pathfinding for beginners. <http://homepages.abdn.ac.uk/f.guerin/pages/teaching/CS1013/practicals/aStarTutorial.htm>. Dostęp: 2018-01-02.
- [16] Amazon warehouse demand devours robots and workers. https://www.roboticsbusinessreview.com/supply-chain/amazon_warehouse_demand_devours_robots_and_workers. Dostęp: 2018-01-05.
- [17] Choset H. Robotic motion planning: Potential functions. https://www.cs.cmu.edu/~motionplanning/lecture/Chap4-Potential-Field_howie.pdf. Dostęp: 2018-01-02.
- [18] Roboty TUG i HOMER firmy Aethon. <http://www.aethon.com/tug/tughealthcare/>. Dostęp: 2018-01-02.
- [19] Searching using a*. <http://web.mit.edu/eranki/www/tutorials/search/>. Dostęp: 2018-01-02.

Wykaz skrótów

CA*	Cooperative A*
HCA*	Hierarchical Cooperative A*
LRA*	Local Repair A*
MAS	Multi-Agent Systems
RTS	Real Time Strategy
WHCA*	Windowed Hierarchical Cooperative A*

Spis rysunków

1.1	Przykładowe środowisko z dużą liczbą przeszkód i rozmieszczonymi robotami. Źródło: własna implementacja oprogramowania symulacyjnego	4
1.2	Roboty Kiva pracujące w magazynie firmy Amazon. Źródło: [?]	5
1.3	Popularny problem zakleszczania się jednostek. Źródło: gra komputerowa Age of Empires II HD	6
2.1	Zasada działania metody pól potencjałowych. Źródło: [?]	9
2.2	Ciągła przestrzeń mapy zdyskretyzowana do siatki pól, Źródło: edytor map z gry Warcraft III.	10
2.3	Sytuacja, w której żadne rozwiązanie nie zostanie znalezione, jeśli robot 1 ma wyższy priorytet niż robot 2. Źródło: [?]	11
2.4	a) Niezależne planowanie optymalnych tras dla 2 robotów; b) suboptymalne rozwiązanie, gdy robot 1 ma wyższy priorytet; c) rozwiązanie, gdy robot 2 ma wyższy priorytet. Źródło: [?]	11
3.1	Przykład działania algorytmu A*. Każdy odwiedzony węzeł wskazuje na swojego rodzica, co umożliwia późniejszą rekonstrukcję drogi. Źródło: [?]	14
3.2	Mapa z zaznaczeniem podziału na siatkę pól, zastosowanie heurystyki uwzględniającej zawijanie mapy po bokach, Źródło: własna implementacja gry PacMan .	15
3.3	Dwie jednostki kooperacyjnie poszukujące tras. (A) Pierwsza jednostka znajduje ścieżkę i zaznacza ją w tablicy rezerwacji. (B) Druga jednostka znajduje ścieżkę, uwzględniając istniejące rezerwacje pól, również zaznaczając ją w tablicy rezerwacji. Źródło: [?]	18
3.4	Czasoprzestrzenna mapa może różnić się od tablicy rezerwacji. (A) Powolna jednostka ma "głębokie" komórki na czasoprzestrzennej mapie. (B) Szybka jednostka ma "pływkie" komórki. (C) Tablica rezerwacji jest współdzielona między wszystkimi agentami, dlatego powinna być odpowiednio dopasowana do wszystkich agentów. Źródło: [?]	18