Balthasar Szczepański

Metody sztucznej inteligencji - projekt

Inteligentna identyfikacja bezkolizyjnej trasy

Wprowadzenie

Założenia

Celem projektu było zaimplementowanie algorytmu pozwalającego wirtualnemu robotowi na znajdowanie bezkolizyjnej trasy omijającej przeszkody i poruszanie się po niej. Algorytm musi uwzględniać zmieniające się warunki, np. ruchomy cel. Nie musi znajdować trasy najkrótszej, ważne jest natomiast działanie w czasie rzeczywistym.

Środowisko

Za środowisko do implementacji i testowania algorytmu posłużyła gra Colobot. Jest to gra, w której po trójwymiarowym świecie poruszają się roboty, które można programować w języku CBot. Ject to język o składni podobnej do C++, z elementami obiektowości. Dlatego jest odpowiednim środowiskiem testowym, ponieważ pozwala an skupienie się na samym algorytmie, bez konieczności tworzenia warunków testowych. Język CBot jest językiem interpretowanym, co ogranicza szybkość działania programów. Gra jest dostępna na licencji GNU GPLv3.

Algorytm

Istnieją różne podejścia do problemu znajdowania trasy w grach. Należało wybrać takie, które jest dopasowane do warunków i założeń projekty. Z góry można odrzucić metody polegające na poruszaniu się po predefiniowanym grafie. algorytm ma dopasowywać się do nieznanych warunków otoczenia. Należało więc wybrać metodę polegającą na zbieraniu informacji o otoczeniu i tworzeniu trasy na podstawie tych informacji.

Istnieją metody, w których robot porusza się po siatce, najczęściej prostokątnej, oraz takie, w których może poruszać się swobodnie po całej przestrzeni. Ponieważ podejrzewałem, że zastosowanie metody bez siatki może przerosnąć moje możliwości czasowe, oraz, ze jej implementacja moze być niemożliwa ze względu na ograniczenia środowiska zdecydowałem się na zastosowanie siatki. W mojej implementacji położenie i orientacja siatki w przestrzeni zmienia się iteracyjnie, a sposób sterowania robotem "wygładza" jego trasę, co powoduje, że po ruchach robota nie widać, że porusza się po siatce.

Popularnym i sprawdzonym rozwiązaniem jest algorytm A*, który jest dobrym kompromisem między szybkim działaniem, a znalezieniem najkrótszej trasy. Przy dużych siatkach algorytm wymaga jednak dużo czasu, nie uwzględnia zmieniających się warunków, nie radzi sobie, gdy miejsce docelowe jest niedostępne. Wobec tego postanowiłem podejść do problemu iteracyjnie.

W danej chwili robot wyznacza trasę tylko w swoim najbliższym otoczeniu. Ponieważ zanim robot dojedzie do końca trasy sytuacja moze się zmieniać, nie ma sensu wyznaczać całej trasy, a zmniejszona siatka skraca czas obliczeń. W każdej iteracji robot wyznacza siatkę

wokół siebie i szuka w niej trasy. Do rozwiązania podproblemu stosuje zmodyfikowany algorytm A*. W czasie wyznaczania natępnej iteracji trasy robot porusza się po trasie z poprzedniej iteracji. Po wyznaczeniu trasy, robot zaczyna poruszać się po niej oraz przechodzi do wyznaczenia kolejnego odcinka.

Algorytm A* w podstawowej postaci działa następująco: Każda komórka siatki może znajdować się w jednym z trzech stanów: nieodwiedzony, otwarty i zamknięty. Dodatkowo, każda komórka ma przypisaną wartość liczbową oraz kierunek. Na początku wszystkie komórki są w stanie nieodwiedzonym za wyjątkiem pozycji początkowej, która jest w stanie otwartym, a jej wartość liczbowa jest równa odległości od punktu końcowego.

W każdej iteracji ze zbioru komórek w stanie otwartym wybierana jest komórka o najlepszej (najmniejszej) wartości, a jej stan jest zmieniany na zamknięty. Następnie sprawdzane są wszystkie komórki sąsiadujące. Jeżeli komórka jest w stanie nieodwiedzonym i nie jest zajęta przez przeszkodę jest dodawana do zbioru komórek otwartych. Wpisywany jest do niej kiekunek wskazujący na aktualnie wybraną komórkę oraz wartość będąca sumą drogi przebytej od punktu początkowego i odległości od punktu końcowego.

Pętla kończy się w momencie dojścia do punktu końcowego lub gdy w zbiór komórek otwartych jest pusty. Po zakończeniu każda odwiezona komórka ma zapisany kierunek wskazujący na poprzednią komórkę. Trasa zostaje odtworzona od końca, Zaczynając od punktu końcowego w pętli algorytm dodaje aktualnie wybraną komórkę do trasy i przechodzi do następnej, na którą wskazuje wpisany do komórki kierunek. Pętla kończy się po dojściu do punktu początkowego.

Zastosowałem lekko zmodyfikowaną wersję algorytmu. W skład wartości liczbowej oprócz drogi przebytej od punktu startowego i szacowanej odległości od punktu końcowego wchodzi jeszcze wartość zalażna od współrzędnych komórki. Dzieki temu w przypadku jednakowych tras program będzie nieznacznie preferował jedną z nich.

Wprowadzilem też dodatkowe kryterium stopu. Jeżeli przez określoną liczbę poditeracji nie ma postępu (nie udaje sie znaleźć komórek o mniejszej wartości) bieżąca iteracja jest kończona i wybierany jest najlepsza z dotychczas przeszukanych komórek. W podstawowej postaci algorytm A*, gdy nie ma trasy prowadzącej do punktu końcowego, przeszukuje całą siatkę niepotrzebnie tracąc na to czas. Dla sprawdzania postępu stosowane jest inne kryterium niż dla wyznaczania trasy. Odległość od celu jest wliczana dwa razy.

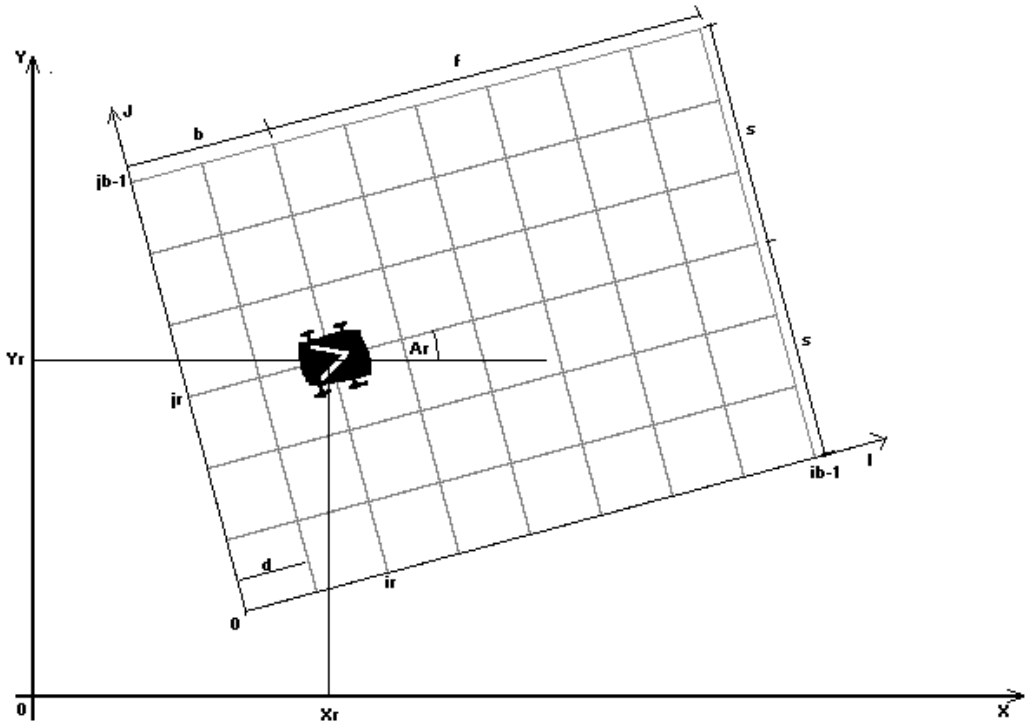
Dodatkowo dopuszczam możliwość szukania następnej komórki wśród komórek zabronionych jeżeli aktualna komórka również jest zabroniona. Pozwala to programowi wyjść z zabronionego obszaru ale nie pozwala do niego wejść. Wartości przypisane takim komórkom są 10 razy wieksze, przez to program będzie próbował jak najszybciej wyjść z zabronionego obszaru.

Implementacja

Siatki

W każdej iteracji potrzebne są dwie siatki, aktualna i następna. Na każdej z nich zaznaczane są komórki zablokowane oraz wyznaczana jest trasa. Siatka została

zaimplementowana jako klasa o nazwie `path`. Ponieważ w języku CBot klasy mogą być tylko publiczne, również ta klasa jest publiczna. Siatka tworzona jest wokół robota i jest określona



przez następujące parametry.: początkowe położenie robota (xr, yr), początkową orientację robota (ar), wymiary siatki w kierunkach w przód(f), w tył(b) i na boki (s). Siatka ma długość ib=f+b+1 i szerokość jb=2s+1. Początkowe współrzędne robota to ir=b i jr=s.

Dodatkowo, klasa przechowuje informacje o komórkach: zajętość , stan, kierunek i odległość od punktu startowego. dodatkowo klasa przechowuje punkty składające się na trasę (wp).

Przeliczanie współrzędnych względnych na bezwzględne odbywa się według następującego wzoru:

x=xr+(i-ir)·d·cos(ar)-(j-jr)·d·sin(ar); y=yr+(i-ir)·d·sin(ar)-(j-jr)·d·cos(ar).

Podobnie wygląda przeliczanie współrzędnych bezwzględnych na względne:

i=((x-xr)·cos(ar)+(y-yr)·sin(ar))/d+ir; j=((y-yr)·cos(ar)-(x-xr)·sin(ar))/d+jr

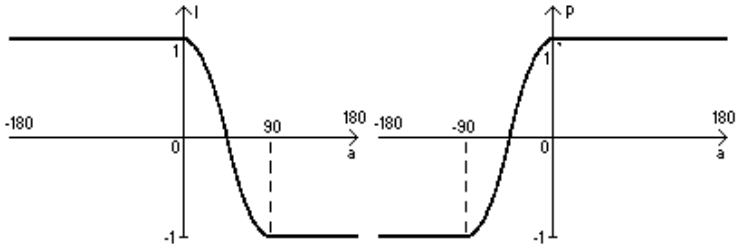
Ponieważ sinius i cosinus kąta ar powtarzają się we wzorach, są obliczane tylko raz, przy inicjalizacji siatki.

Sterowanie robotem

Robot porusza się po wyznaczonych punktach znajdujących się na siatce. Jednak nie ma potrzeby, żeby jeździł od punktu do punktu dokładnie po liniach lub przekątnych siatki. Gdy robot zbliży się do punktu na odległość mniejszą niż 1,5 szerokości komórki, zaczyna jechać w kierunku następnego.

Sterowanie silnikami robota jest oparte na sprzężeniu zwrotnym. Im bardziej kierunek ruchu różni się kierunku w stronę wybranego punktu tym mocniej robot skręca, żeby to skorygować. Ustawienia silników przyjmują wartości od -1 (pełna moc w tył) do 1 (pełna moc w przód). Ustawienia silnika lewego i prawego opisane są wzorami:

$$l=\begin{cases} 1 & a\in\langle -180,0)\\ \cos(2a) & a\in\langle 0,90)\\ -1 & a\in\langle 90,180\rangle \end{cases} \quad p=\begin{cases} -1 & a\in\langle -180,-90)\\ \cos(2a) & a\in\langle -90,0)\\ 1 & a\in\langle 0,180\rangle \end{cases}$$



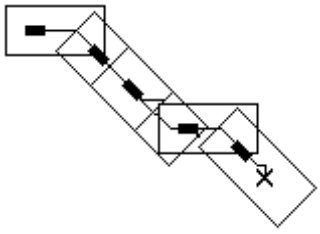
Zbiór komórek w stanie otwartym

Zbiór komórek w stanie otwartym został zaimplementowany w klasie o nazwie `pf_front`. komórki zapamiętywane są w tablicach. Ponieważ dodawane są różne komórki, ale usuwana jest zawsze najlepsza, komórki są umieszczane w kolejności od najlepszej do najgorszej. Kolejność nie ma nic wspólnego z indeksem tablicy, który się nie zmienia. Jednym z pól klasy jest indeks najlepszej komórki. Każda komórka posiada indeks kolejnej, najlepszej po niej komórki.

W czasie dodadawania nowej komórki sprawdzane są komórki od najlepszej, aż do znalezienia miejsca w którym należy umieścić nową komórkę. Usunięcie najlepszej polega na zamianie pola wskazującego na najlepszą komórkę. Żadne elementy tablic nie są rzeczywiście usuwane i pozostają w pamięci do końca iteracji.

Główna pętla

Główna pętla została zaimplementowana w funkcji `mainloop` w klasie `pathfinder`. W głównej pętli aktualizowana jest pozycja celu. Tworzona jest nowa siatka wokół aktualnej pozycji robota. W tej siatce wyznaczana jest nowa trasa. Równocześnie robot prousza się po poprzednio wyznaczonej trasie.



Badanie terenu

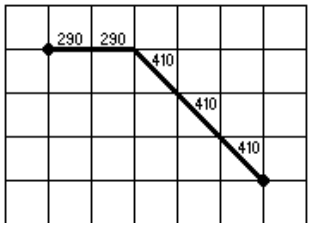
Zostały stablicowane najmniejsze promienie w których mieszczą się obiekty każdego typu. Za pomocą wbudowanej funkcji retobject sprawdzane są wszystkie obirkty w grze. Jeżeli obiekt znajduje się na siatce, a promień jest większy od zera, wówczas odpowiednie komórki (stanowiąca kwadrat o boku równym promieniowi) zostają oznaczona jako zablokowane.

Wyznaczenie trasy

W każdej iteracji jest wyznaczana trasa dla danej siatki. Odbywa się to w sposób opisany we wprowadzeniu. Jeżeli punkt docelowy znajduje się poza siatką, jest ustawiany na granicy siatki. Wartość każdej komórki jest sumą następujących elementów: przebytej drogi od punktu

startowego, szacowanej odległości od punktu , i wartości uzależnionej od współrzędnych.

Droga przebyta od punktu początkowego jest wyznaczana przez dodanie odpowiediej wartości do odległości wyznaczonej dla poprzedniej komórki. dla ruchu wzdłuż linii siatki wartość wynosi 290. Dla ruchu po przekątnej siatki wartość wynosi 410. W ten sposób uzyskuje się przybliżenie $\sqrt{2}\approx41/29$. Szacowana odległość od punktu końcowego jest najmniejszą sumą odcinków leżących na liniach i przekątnych łączących dany punkt z punktem końcowym. Te odciki również mają wartości 410 i 290. Wartość zależna od współrzędnych wynosi j-2i.

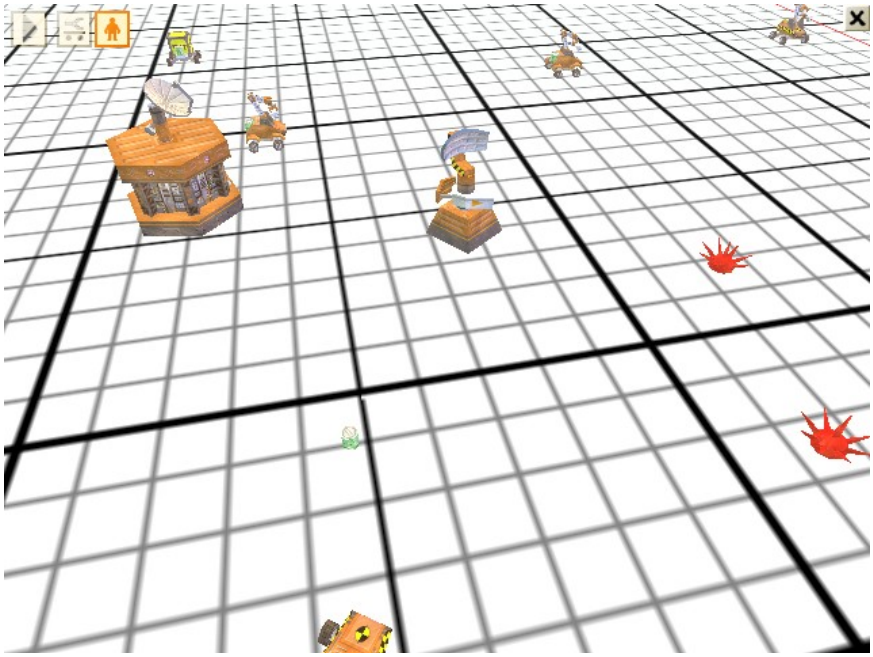


Po zakończeniu przeszukiwania siatki tworzona jest trasa. Na początku trasy dostawiane są punkty z poprzedniej trasy. Jeżeli została wyznaczana trasa niepokryająca się z trasą z poprzedniej iteracji, a robot przejechał już część drogi, powinien wiedziec, któredy wroić. Na podstawie odległości ustalane jest w którym punkcie znajduje się robot. Kryterium stopu w razie braku postępu wynosi 72 poditeracje.

Testowanie

Warunki testowania

Przygotowałem poziom testowy zawierający przypadkowo rozmieszczone obiekty stanowiące przeszkody, robota wykonującego program, oraz kosmonautę którym można sterować, a który stanowi cel dla robota. Testowanie polegało na przemieszczaniu się kosmonauty i obserwowanie, jak robot radzi sobie z odnajdowaniem drogi.



Dodatkowo, w czasie wstępnego testowania były generowane pliki tekstowe pokazujące wyznaczoną trasę.

Obserwacje

Ze względu na charakter zagadnienia nie da się przedstawić wyników o charakterze ilościowym. Obserwując zachowanie robota mozna stwierdzić, że prawidłowo omija przeszkody i nie próbuje przez nie przejeżdżać, Trasa po której porusza się robot nie jest trasą najkrótszą. Jednak wyznaczenie najkrótszej możliwej trasy nie było założeniem programu. W niektórych przypadkach robot potrafi się "zgubić". "Nie może się zdecydować", z której strony ominąć przeszkodę. W jednej iteracji wybiera ominięcie przeszkody z jednej strony. Lecz zanim to zrobi, w następnej iteracji wybiera ominięcie przeszkody z drugiej strony.

Wnioski

Zaproponaowany przeze mnie projekt nie jest idealny. W niektórych przypadkach nie radzi sobie optymalnie z problemem. Można to usprawnić przez stosowanie zmiennego rozmiaru i gęstości siatki oraz wprowadzanie dodatkowych czasów między iteracjami. Nie umiem jednak wskazać kryterium, na podstawie którego miały by być ustalane te wartości.

Od strony algorytmicznej nie ma przeszkód, żeby wyznaczyć trasę najkrótszą. Wystarczy w tym celu zwiększyć siatkę, aby objęła cały obszar i zrezygnować z szacowania odległości do punktu końcowego, zamiast tego przesukiwać wszystkie punkty. Jednak wówczas ze względu na duże zwiększenie liczby potrzebnych obliczeń algorytm nie nadawałby się zastosowania on-line. Dlatego lepszym rozwiązaniem jest szukanie trasy trochę dłuższej, ale możliwej do policzenia w czasie rzeczywistym.

Algorytm nie nadaje się do praktycznego zastosowania w grze ze względu na ograniczenia tej gry. Język jest interpretowany i dlatego działa mało wydajnie. Mój program ledwo nadąża z obliczeniami. Istnieje możliwość zmiany liczby wykonywanych instrukcji na jedną klatkę gry, ale jest to wartość globalna, dotycząca wszystkich robotów. Powoduje to, że program będzie działał odpowiednio szybko tylko wtedy, gdy w grze żaden inny robot nie wykonuje żadnego programu.

Dużym problemem w czasie testowania był słabo udokumentowany język Cbot i bardzo ograniczona możliwość debuggingu.

Uruchomienie

Do uruchomienia projektu potrzebna jest gra Colobot. Grę można legalnie pobrać ze strony http://colobot.info. katalog projektu należ skopiować do podkatalogu **user** w katalogu gry. Jezeli do projektu była dołączona gra, nie trzeba niczego kopiować. Po włączeniu gry należy klikać następujące przyciski: Poziomy > AI_pathfinder > Graj. Po wczytaniu program autowatycznie wystartuje. za pomocą klawiszy strzałek można sterować kosmonautą.

Kod programu

Ze względu na długość (788 linii) kod nie został umieszczony w niniejszym tekście. Kod programu znajduje się w katalogu projeku w pliku o nazwie **pathfinder.txt**.

Literatura

- *Pathfinding in Computer Games* - Ross Graham, Hugh McCabe and Stephen Sheridan
- *Amit's Thoughts on Pathfinding* - http://theory.stanford.edu/~amitp/GameProgramming/
- Toward More Realistic Pathfinding - Marco Pinter - http://www.gamasutra.com/view/feature/131505/toward_more_realistic_pathfinding.php?page=2