**2. Cel projektu**

Celem projektu opracowanie algorytmu **A\*** za pomocą którego wyznaczymy najkrótszą ścieżkę ewakuacji dla grupy ludzi z pomieszczenia o kształcie szachownicy. Na trasie mogą pojawić się przeszkody, a liczba drzwi może być większa niż jeden. Algorytm musi brać pod uwagę te czynniki i znaleźć najkrótszą ścieżkę, do najbliżej położonych drzwi.

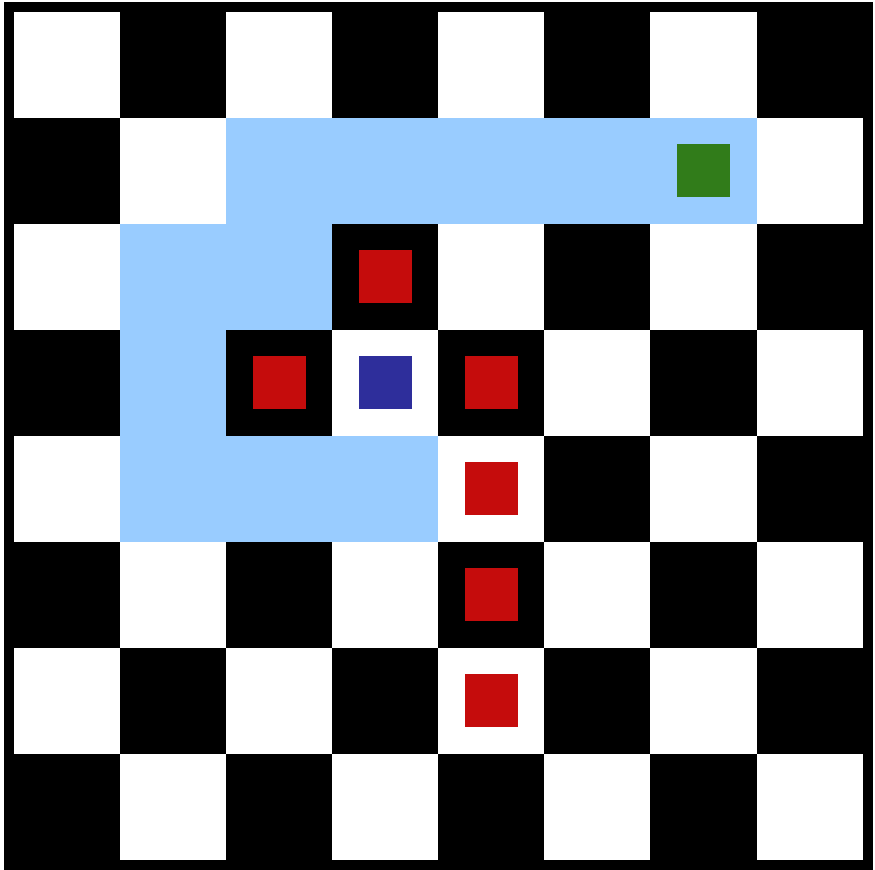
Celem badań podjętych w niniejszej pracy jest wykazanie poprawności działania algorytmu A\* oraz zaprezentowanie jego działania.

Efekty zostaną zaprezentowane w formie graficznej, a także tekstowej. Zostaną wykonane oraz dołączone do tej pracy w formie załączników. Będzie to aplikacja przeglądarkowa.

**3. Sposób realizacji**

Jako algorytm wyszukujący najkrótszą ścieżkę wybraliśmy algorytm A\*, jako jeden z najprostszych algorytmów, który jest powszechnie używany w tego typu zadaniach.

Algorytm zaimplementowaliśmy w aplikacji przeglądarkowej w której na szachownicy o wymiarach 8 x 8 można dowolnie dodawać przeszkody, drzwi oraz ludzi. Po uruchomieniu algorytmu następuje znalezienie najkrótszej ścieżki, lub zakomunikowanie błędem w przypadku braku jakiejkolwiek ścieżki do wyjścia. Najkrótsza ścieżka oznaczana jest jednakowym kolorem dla wszystkich ludzi, dlatego możliwe są nałożenia się ścieżki w przypadku ewakuacji większej liczby ludzi. Ścieżka wskazuje wszystkie pola, wraz z elementem docelowym (drzwiami) które człowiek musi przejść, aby wydostać się z pomieszczenia.



**Rysunek 1** Przykładowa ścieżka ewakuacyjna do drzwi

Na powyższym zrzucie ekranu za pomocą koloru niebieskiego oznaczono najkrótszą ścieżkę. Kolorem czerwonym oznaczono przeszkody. Kolorem fioletowym oznaczono człowieka, a zarazem punkt startowy. Kolorem zielonym oznaczono drzwi, a zarazem punkt docelowy – końcowy.

**4. Schemat algorytmu**

**Szukanie ścieżki w pomieszczeniu bez przeszkód**

Praktycznie bez żadnych problemów możemy wytyczyć najkrótszą ścieżkę z punktu A do punktu B na szachownicy, gdy nie ma na niej żadnych przeszkód.

Algorytm A\* polega na znalezieniu wszystkich możliwych ruchów z danego miejsca, a następnie wybranie miejsca najbliższego miejsca docelowego.

Najbliższe miejsce docelowe ustala się licząc kilka parametrów, oraz porównując ich sumę:

- Koszt G - dystans od punktu startowego A, do obecnie przetwarzanego ruchu

- Koszt H - dystans od obecnie przetwarzanego ruchu do punktu docelowego B

- Koszt F - suma obydwu powyższych kosztów G + H

W każdej iteracji powinien zostać wybrany ruch o najmniejszym koszcie F, lub dowolnego ruchu o najmniejszym koszcie (kwestia optymalizacji).

W ogólnym zamyśle koszt G ma zwiększać się wraz z każdym ruchem, oznaczając oddalanie się od punktu początkowego A, a koszt H zmniejszać się, oznaczając przybliżanie się do punktu końcowego B.

Po każdym ruchu sprawdzamy, czy znajdujemy się na punkcie docelowym B, lub w jego bezpośrednim zasięgu (w zależności od wytycznych).

**Szukanie ścieżki w pomieszczeniu z przeszkodami**

Ustalenie najkrótszej drogi na szachownicy z przeszkodami stanowi pewien problem. Powyższy schemat algorytmu nie zadziała, gdyż najprawdopodobniej algorytm zatrzyma się w pętli wybierając ruch który został już wykorzystany. Potrzebna jest zmiana stanu algorytmu oraz implementację kilku innych parametrów dla każdego ruchu wykorzystanego, i możliwego do wykorzystania.

Główny zamysł algorytmu pozostaje bez zmian, musimy natomiast przechowywać informacje o tym które ruchy zostały już wykorzystane, gdyż wykorzystanych ruchów nie ma już sensu ponownie wykorzystywać. Musimy przechowywać informację z jakiego miejsca na szachownicy przesunęliśmy się na dane pole, oraz jaki był tego koszt (G, H, oraz ich suma - F).

Koszt G dojścia na każde pole wyliczamy poprzez sumę wszystkich ruchów wykonanych, aby wejść na to pole.

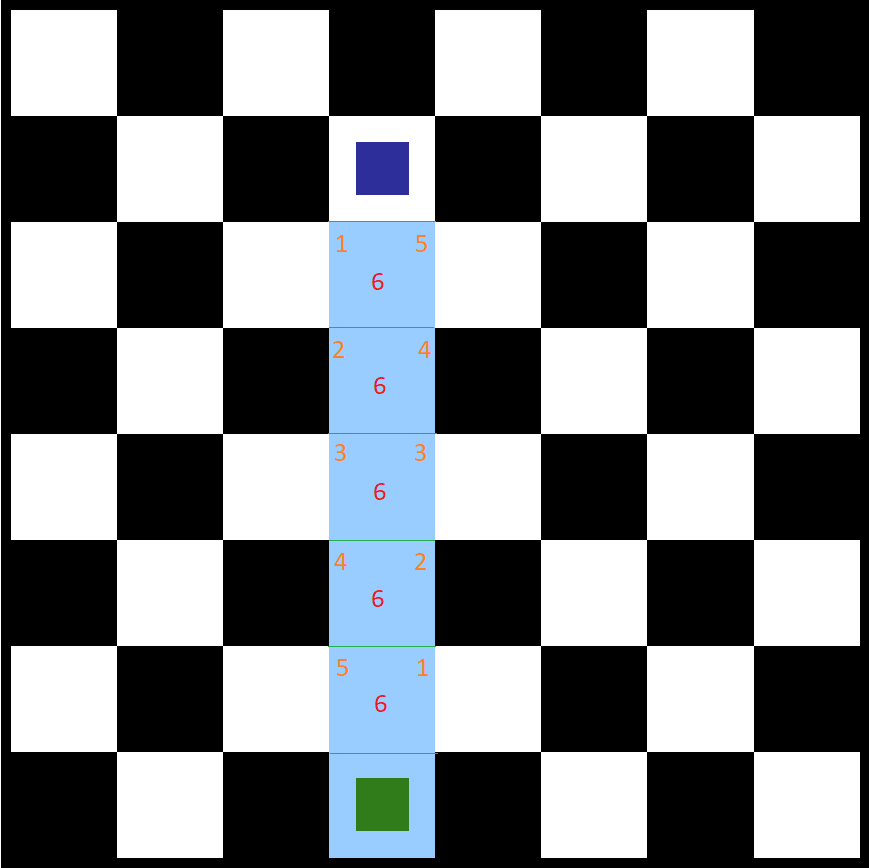
W celu zobaczenia naszej implementacji algorytmu wraz ze szczegółowymi komentarzami odsyłamy do funkcji **findPathToClosestDoor** z klasy **ChesseElement (js/ChesseElement.js)** aplikacji załączonej do tej pracy. Odpowiada ona za znalezienie najkrótszej ścieżki dla danego człowieka do najbliższych drzwi, i usunięcie go z mapy (w celu umożliwienia wydostania się kolejnym osobom).

**5. Schemat algorytmu i kod programu**

**6. Wykazanie poprawności algorytmu**

Algorytm A\* jest kompletny, co oznacza, że w każdym możliwym przypadku znajdzie optymalną, czy też najkrótszą drogę i zakończy działanie, o ile taka droga istnieje. Nie istnieje inny algorytm, który wyznaczy krótszą drogę, mogą natomiast istnieć inne algorytmy które znajdą taką samą drogę, ale w krótszym czasie.

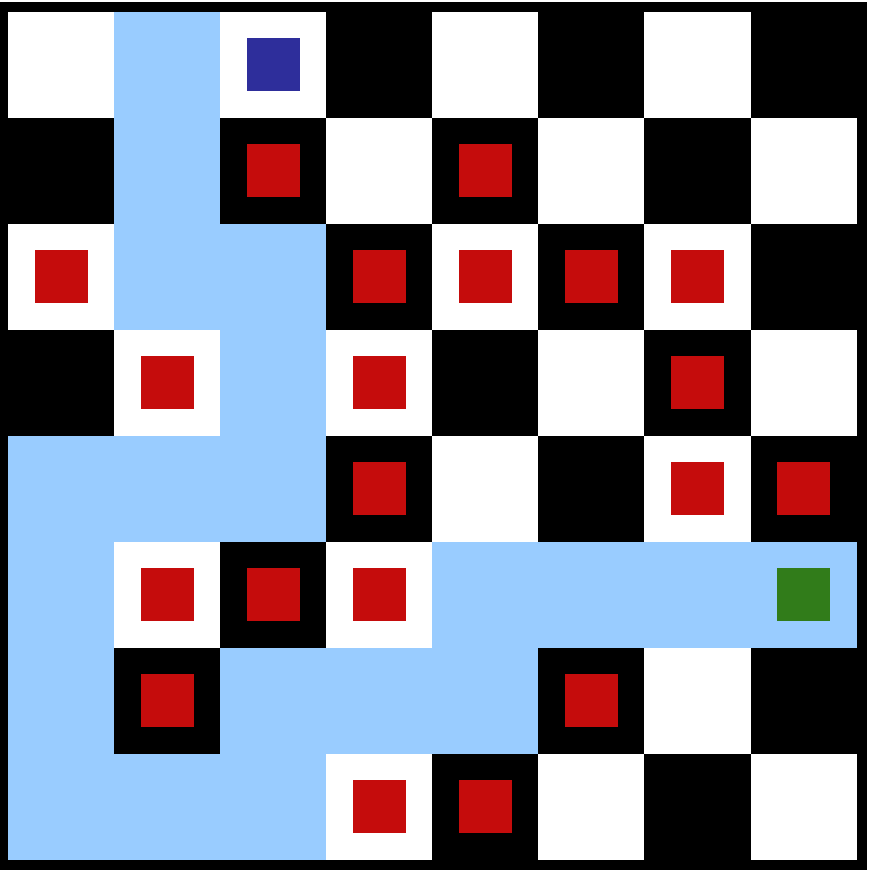
Złożoność czasowa dla algorytmu zależy od liczby przeszkód, oraz liczby drzwi. Najbardziej optymistycznym scenariuszem jest brak przeszkód, oraz jedne drzwi znajdujące się w linii prostej od punktu startowego (człowieka). W takim przypadku algorytm ciągle będzie zbliżał się do drzwi, nie będzie sytuacji w której algorytm sprawdzi ruch którego potem nie wykorzysta w ścieżce końcowej.



Rysunek 2 najbardziej optymistyczny przypadek

Na powyższym obrazku na fioletowo zaznaczyliśmy punkt początkowy, niebieskim kolorem oznaczyliśmy ścieżkę, a zielonym kolorem drzwi. Liczba w lewym rogu każdego ruchu na ścieżce to koszt G, a liczba w prawym rogu to koszt H. Na czerwono oznaczono ich łączny koszt F.

Widać, jak koszt G rośnie, oznacza to oddalanie się od punktu początkowego. Koszt H natomiast z każdym ruchem maleje. Oznacza to przybliżanie się do celu.



Rysunek 3 znacznie trudniejszy przypadek

Przeszkody wymuszają bardziej złożone operowanie algorytmem. W powyższym przypadku algorytm przeanalizował pola które na pierwszy rzut oka wydają się przybliżać człowieka do celu, natomiast w pewnym momencie dalsze ruchy okazywały się niemożliwe ze względu na przeszkody.

Algorytm w takich przypadkach musiał znaleźć kolejny ruch, o najniższym koszcie F (a w przypadku, gdy jest kilka ruchów o takim samym, najniższym koszcie F, algorytm wybrał ten o niższym koszcie H), który nie został jeszcze przeanalizowany.

Złożoność czasowa będzie rosła, gdy algorytm będzie musiał przeanalizować jak najwięcej ruchów. Liczbę maksymalną ruchów możemy obliczyć ze wzoru. W przypadku tej pracy analizujemy pomieszczenie o wymiarach 8x8. Łącza liczba pól wynosi zatem 64.

**7. Wnioski i możliwe ulepszenia**

**8. Bibliografia i przydatne linki**