Metody rozwiązywania labiryntu

Oskar Wiśniewski, 198058
Mikołaj Wiszniewski 197925
Franciszek Fabiński 197797

Politechnika Gdańska, WETI

28 maja 2025

1 Wstęp

Celem tego sprawozdania jest porównanie różnych, zaimplementowanych wcześniej metod rozwiązywania labiryntu: depth-first search, A* oraz Q-learning. W analizie skupiono się na takich aspektach jak skuteczność znajdowania rozwiązania, czas działania algorytmu oraz liczba odwiedzonych pól. Dodatkowo oceniono, jak poszczególne metody radzą sobie w labiryntach o różnym rodzaju zagęszczenia.

2 Generowanie labiryntu

Wygenerowanie labiryntu opiera się na algorytmie opartym na algorytmie Kruskala ze strukturą zbiorów rozłącznych (disjoin set), który zapewnia utworzenie spójnej i acyklicznej siatki przejść – klasycznego labiryntu z jedną możliwą ścieżką pomiędzy dowolnymi dwoma punktami.

Najpierw tworzona jest siatka ścian o rozmiarze `2*szerokość+1` na `2*wysokość+1`, w której wszystkie komórki są domyślnie oznaczone jako ściany. Następnie generowana jest lista możliwych ścian do usunięcia, tzn. są to wszystkie sąsiadujące komórki w siatce reprezentujące potencjalne przejścia między komórkami. Lista ta jest losowo tasowana, aby zapewnić różnorodność generowanych labiryntów.

Dla każdej pary sąsiednich komórek sprawdzana jest przynależność do różnych zbiorów. Jeżeli tak – oznacza to, że usunięcie ściany między nimi nie utworzy cyklu. W takim przypadku ściana jest usuwana (komórki połączone są "korytarzem"), a zbiory są scalane.

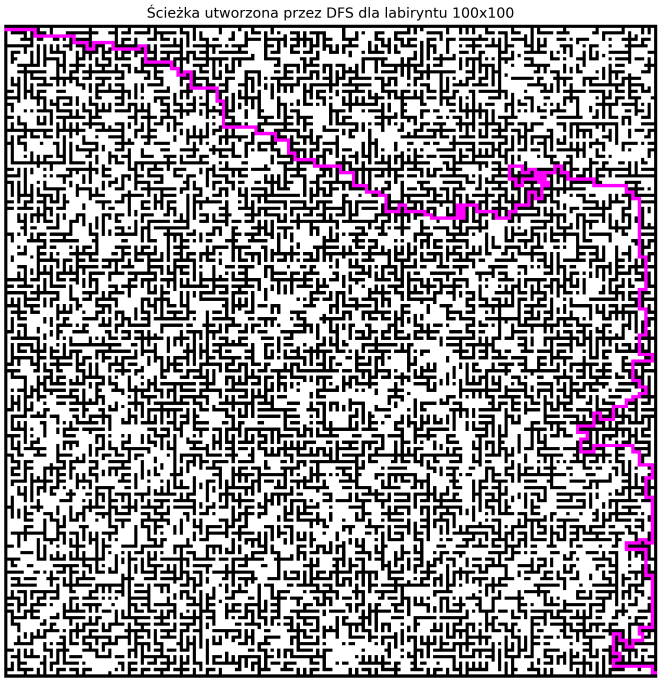
Dodatkowo, z prawdopodobieństwem `X`, usuwane są dodatkowe ściany boczne przylegające do tworzonego przejścia, co wprowadza rozgałęzienia i alternatywne ścieżki. Nadaje to labiryntowi bardziej złożony charakter.

Na zakończenie, siatka jest obudowywana ścianą zewnętrzną oraz ustawiane są jedno wejście i wyjście w przeciwległych rogach labiryntu (lewy górny - wejście, prawy dolny - wyjście). Labirynt jest gotowy do wizualizacji oraz dalszego przetwarzania, np. przeszukiwania.

3 Wyniki dla typowego labiryntu

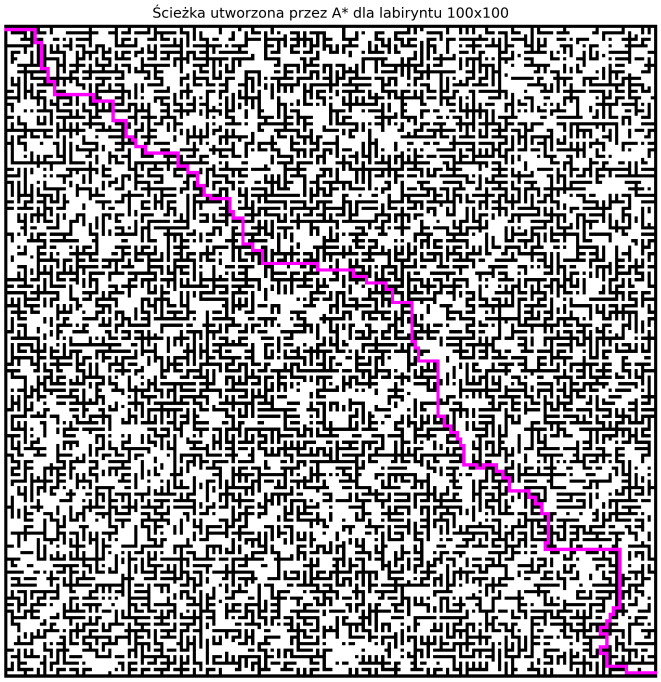
3.1 Depth-first search

Ze względu na swoją prostotę, implementacja algorytmy DFS nie przysporzyła wielu problemów. Podczas testów dla większych labiryntów konieczna okazała się zmiana wersji rekurencyjnej na iteracyjną. Zauważono również silny wpływ kolejności sprawdzania wolnych pól na parametry wyjściowe programu, w szczególności kształt ścieżki. Wiąże się to z tym, że algorytm będzie wybierał stosunkowo częściej pola pod niższymi indeksami, ponieważ przeszukujemy "w głąb". Po testach najkorzystniejszą kolejnością okazała się $[(0,-1),(-1,0),(0,1),(1,0)]$. Wynikowe ścieżki zwykle przebiegają w przybliżeniu wzdłuż górnej i prawej krawędzi labiryntu, co pokazano poniżej.



3.2 A*

W algorytmie A* wykorzystano heurystykę Manhattan, która polega na obliczaniu odległości pomiędzy dwoma punktami jako sumy wartości bezwzględnych różnic ich współrzędnych poziomych i pionowych. Jest to podejście szczególnie dobrze dopasowane do środowisk reprezentowanych w postaci siatki, gdzie możliwy jest wyłącznie ruch w czterech kierunkach: góra, dół, lewo i prawo. Z tego powodu był to pierwszy wybór przy implementacji. Z tego powodu wynikowe ścieżki zwykle biegną w przybliżeniu wzdłuż przekątnej od punktu wejściowego do wyjściowego, co pokazano poniżej.



3.3 Q-learning

TODO: MIKOŁAJ

3.4 Porównanie

TODO: ???