# Implementacja algorytmu A*

Dawid Scechura, Hubert Mucha

21 czerwca 2023

## Spis treści



# 1 Wprowadzenie

Algorytm A* jest jednym z najbardziej popularnych i wpływowych algorytmów w dziedzinie sztucznej inteligencji i robotyki. Jego wyjątkowa zdolność do znajdowania optymalnej ścieżki w grafie lub sieciach złożonych uczyniła go niezwykle ważnym narzędziem w dziedzinach takich jak nawigacja, planowanie tras, robotyka mobilna, grafika komputerowa i wiele innych. Algorytm A* wykorzystuje połączenie heurystyki i przeszukiwania wzdłuż drogi, co pozwala mu znaleźć najbardziej optymalną ścieżkę w bardzo efektywny sposób.

## 2 Zastosowania algorytmu A*

Przykładowe sposoby użycia algorytmu A*:

- Nawigacja: Algorytm A* jest szeroko stosowany w systemach nawigacji, zarówno w nawigacji samochodowej, jak i nawigacji lotniczej. Dzięki swojej zdolności do uwzględniania warunków drogowych, ograniczeń i priorytetów, może znaleźć najkrótszą lub najszybszą trasę pomiędzy dwoma punktami.

- Robotyka mobilna: W robotyce mobilnej algorytm A* jest używany do planowania trajektorii robota w środowisku. Dzięki temu robot może bezpiecznie omijać przeszkody i osiągać zadane cele.

- Planowanie tras: Algorytm A* jest wykorzystywany do planowania tras w różnych dziedzinach, takich jak logistyka, transport, dostarczanie towarów itp. Może zoptymalizować trasę i zmniejszyć czas lub koszty podróży.

- Grafika komputerowa: Algorytm A* jest używany w tworzeniu realistycznych animacji i generowaniu ścieżek dla postaci w grach komputerowych. Może znaleźć najbardziej naturalną trasę dla postaci, aby uniknąć przeszkód i osiągnąć cel.

## 3 Model matematyczny algorytmu

Algorytm A* jest algorytmem przeszukiwania grafu, który wykorzystuje połączenie heurystyki i przeszukiwania wzdłuż drogi w celu znalezienia optymalnej ścieżki pomiędzy dwoma wierzchołkami. Matematycznie, zasada działania algorytmu A* można opisać w następujący sposób:

Rozważamy graf skierowany lub nieskierowany $G(V, E)$, gdzie $V$ oznacza zbiór wierzchołków, a $E$ oznacza zbiór krawędzi. Każda krawędź $(u, v) \in E$ ma przypisaną wagę $w(u, v)$, która reprezentuje koszt podróży między wierzchołkami $u$ i $v$. Dodatkowo, dla każdego wierzchołka $v \in V$ określamy funkcję heurystyczną $h(v)$, która szacuje odległość od wierzchołka $v$ do celowego wierzchołka, zwyczajowo oznaczanego jako *goal*.

Algorytm A* wykonuje przeszukiwanie wzdłuż drogi, przy czym uwzględnia wartość funkcji heurystycznej, aby skierować przeszukiwanie w stronę potencjalnie bardziej obiecujących ścieżek. Dla każdego wierzchołka $v \in V$, algorytm utrzymuje dwie wartości:

- $g(v)$ - koszt dotarcia do wierzchołka $v$ z wierzchołka startowego, gdzie $g(start) = 0$.

- $f(v)$ - szacowany całkowity koszt dotarcia do wierzchołka $v$ z wierzchołka startowego, gdzie $f(v) = g(v) + h(v)$.

Algorytm A* wykonuje następujące kroki:

1. Inicjalizuj zbiór otwartych wierzchołków *Open* i zbiór zamkniętych wierzchołków *Closed*. Dodaj wierzchołek startowy *start* do zbioru otwartych wierzchołków.

2. Dopóki zbiór otwartych wierzchołków *Open* nie jest pusty:

   (a) Wybierz wierzchołek $v$ o najniższym wartości $f(v)$ z *Open*.

   (b) Jeśli $v$ jest wierzchołkiem celowym *goal*, zakończ algorytm. Otrzymana ścieżka jest optymalna.

   (c) Przenieś wierzchołek $v$ z *Open* do *Closed*.

   (d) Dla każdej krawędzi $(v, u)$ wychodzącej z wierzchołka $v$:

   - Oblicz wartość $g(u)$ jako sumę $g(v)$ i kosztu krawędzi $(v, u)$.
   - Jeśli wierzchołek $u$ jest już w *Open* lub *Closed* i $g(u)$ jest większe niż obecne $g(u)$, pomiń ten krok.
   - Jeśli wierzchołek $u$ nie jest ani w *Open*, ani w *Closed*, dodaj go do *Open* i oblicz $f(u)$.
   - Jeśli wierzchołek $u$ jest już w *Open*, zaktualizuj jego wartości $g$ i $f$ jeśli obecne $g(u)$ jest mniejsze od wcześniejszej wartości.

Algorytm A* kontynuuje powyższe kroki, aż do znalezienia optymalnej ścieżki lub wyczerpania zbioru otwartych wierzchołków *Open*.

# 4 Opis implementacji

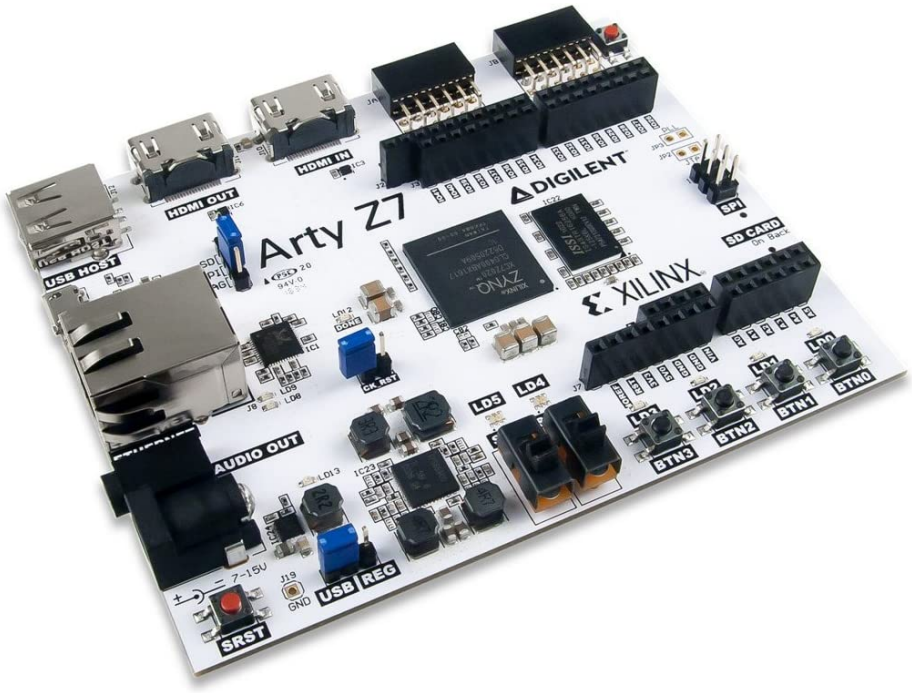
Celem naszego projektu było stworzenie interaktywnego systemu do wizualizacji i wykonywania algorytmu A* na platformie Zynq, konkretnie na płytce rozwojowej Arty Z7. Arty Z7 jest oparta na układzie Zynq-7000 firmy Xilinx, który łączy w sobie procesor ARM Cortex-A9 i programowalną logikę FPGA (Field-Programmable Gate Array).

Platforma Zynq oferuje zalety obu światów: procesora, który zapewnia możliwość wykonywania złożonych obliczeń i obsługi interfejsów, oraz FPGA, który umożliwia elastyczną konfigurację sprzętową dla specjalizowanych operacji. Dzięki temu, mogliśmy wykorzystać moc obliczeniową procesora ARM do implementacji interfejsu użytkownika i komunikacji, a także FPGA do przyspieszenia algorytmu A*.

## 4.1 FPGA

### 4.1.1 Co-procesory w architekturze FPGA

W architekturze FPGA (Field-Programmable Gate Array) istnieje możliwość wykorzystania tzw. co-procesorów, które są dedykowanymi blokami logiki programowalnej, zaprojektowanymi do przyspieszania konkretnych operacji lub algorytmów. Co-procesory są implementowane jako część struktury FPGA i mogą

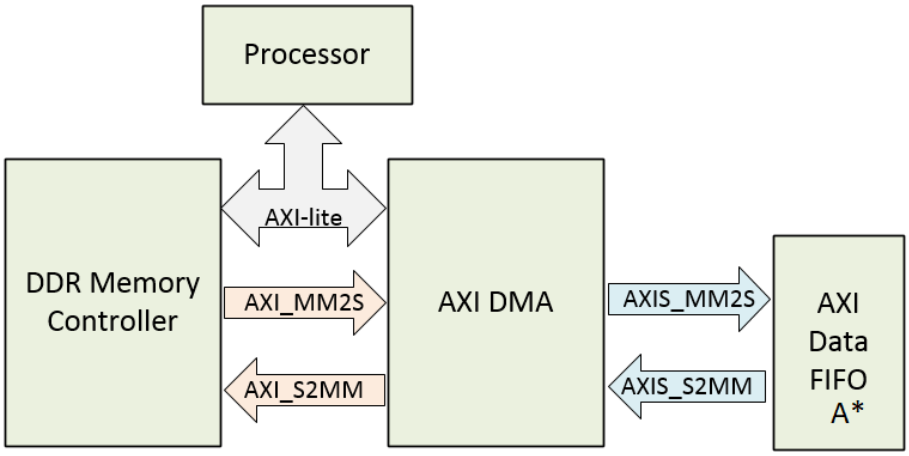



Rysunek 1: Platforma ARTY Z7

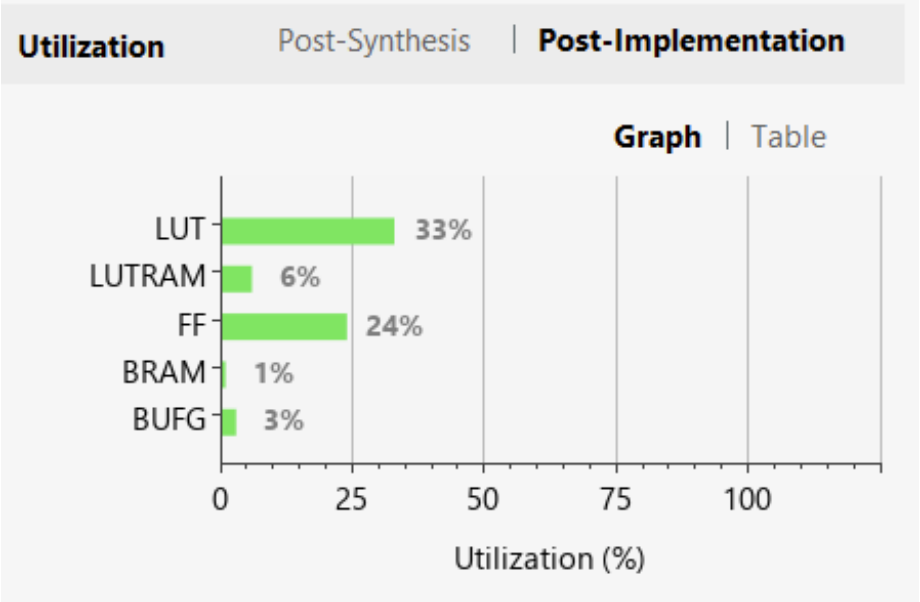
być wykorzystywane razem z głównym procesorem lub mikroprocesorem w celu zwiększenia wydajności systemu.

Co-procesory na FPGA są programowalne, co oznacza, że można je skonfigurować do realizacji określonych funkcji zgodnie z potrzebami aplikacji. Mogą one być zoptymalizowane pod kątem konkretnych operacji, takich jak obliczenia matematyczne, przetwarzanie sygnałów, kompresja danych czy analiza obrazów. Dzięki temu, co-procesory mogą efektywnie wspierać działanie głównego procesora, przyjmując na siebie bardziej złożone i czasochłonne obliczenia.

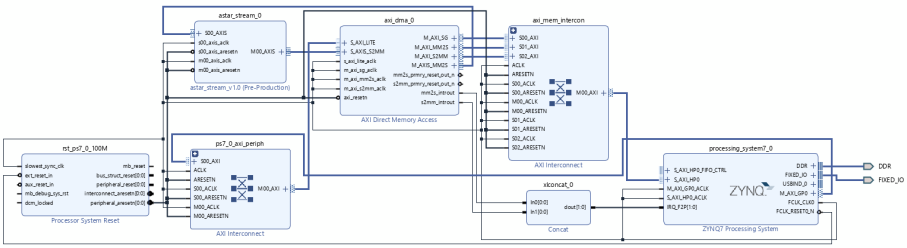
Wykorzystanie co-procesorów na FPGA przynosi wiele korzyści. Po pierwsze, pozwala to na przerzucenie obliczeń związanych z konkretnymi operacjami na dedykowane bloki logiki, co skutkuje znacznym przyspieszeniem i wydajnością systemu. Ponadto, dzięki elastycznemu charakterowi programowalności FPGA, co-procesory mogą być dostosowane do specyficznych wymagań aplikacji, co pozwala na osiągnięcie optymalnych wyników.




Rysunek 2: Diagram integracji AXI DMA z A*




Rysunek 3: Wykorzystane zasoby




Rysunek 4: Schemat blokowy

### 4.1.2 Schemat blokowy

- Blok ZYNQ Processing System:
  - Odpowiedzialny za poprawną konfigurację platformy Zynq.
  - Aktywuje magistralę AXI-HP (High-Performance) do komunikacji z innymi blokami.
- Blok AXI-DMA:
  - Odpowiada za obsługę transferu danych przy użyciu magistrali AXI.
  - Zwalnia procesor od obsługi transferów, umożliwiając mu skoncentrowanie się na innym zadaniu.
- Blok Astar Stream:
  - Zawiera kolejkę FIFO (First-In-First-Out), która służy do buforowania danych.
  - Blok Astara współpracuje z kolejką FIFO, wykorzystując możliwości magistrali AXI Stream, umożliwiając szybkie przesyłanie danych.
- Blok AXI-Interconnect:
  - Wykorzystany do połączenia magistrali AXI platformy Zynq z blokiem AXI-DMA.
  - Jego zadaniem jest zapewnienie kompatybilności między magistralą AXI platformy Zynq a DMA.
- Blok Concat:
  - Odpowiada za scalanie dwóch pojedynczych sygnałów w jeden.
  - Wykorzystany do generowania przerwań związanych z DMA na platformie Zynq.

Ten diagram blokowy przedstawia inicjalizację DMA na platformie Zynq i ukazuje interakcje między poszczególnymi blokami, które są odpowiedzialne za konfigurację platformy, obsługę transferu danych, wykorzystanie magistrali AXI Stream, zapewnienie kompatybilności między magistralami oraz generowanie przerwań związanych z DMA.

## 4.2 Mikroprocesor

Kod dla mikroprocesora został napisany w języku C i składa się z trzech części. Pierwsza część jest odpowiedzialna za konfigurację DMA, która umożliwia efektywne zarządzanie transferem danych między pamięcią a urządzeniami peryferyjnymi. Dla platformy Zynq, konfiguracja DMA może obejmować ustawienie trybu dwukierunkowego, inicjalizację odpowiednich rejestrów konfiguracyjnych oraz przypisanie deskryptorów DMA, które przechowują informacje o transferach danych, takie jak adres źródłowy, adres docelowy i rozmiar bloku danych.

Kolejna część kodu zajmuje się obsługą komunikacji UART, która umożliwia interakcję mikroprocesora z innymi urządzeniami poprzez port szeregowy. W celu obsługi komunikacji UART, używamy funkcji `scanf` do odczytu danych wejściowych z portu szeregowego oraz funkcji `printf` do wysyłania danych wyjściowych przez ten sam port. Komunikacja UART może być skonfigurowana z odpowiednimi parametrami, takimi jak prędkość transmisji, bity danych, kontrola parzystości itp.

Ostatnia część kodu odpowiada za wyświetlanie danych w konsoli, umożliwiając monitorowanie i prezentację informacji na wyjściu mikroprocesora. Dzięki funkcji `printf`, można wygodnie formatować i wyświetlać różne rodzaje danych, komunikatów i informacji diagnostycznych w konsoli.

Podział kodu na trzy części ułatwia zarządzanie i rozwijanie projektu. Każda część ma swoje unikalne zadanie, które współpracuje z innymi modułami, tworząc kompletną funkcjonalność mikroprocesora. Dzięki odpowiedniej konfiguracji DMA, obsłudze komunikacji UART oraz wyświetlaniu danych w konsoli, mikroprocesor może efektywnie zarządzać transferem danych, komunikować się z innymi urządzeniami i prezentować informacje użytkownikowi w czytelny i zrozumiały sposób.

### 4.2.1 Komunikacja UART

Do komunikacji między GUI a mikrokontrolerem ARM, który jest zaimplementowany na platformie Zynq, zastosowaliśmy interfejs UART (Universal Asynchronous Receiver-Transmitter). UART to popularny standard komunikacji szeregowej, który umożliwia transmisję danych między różnymi urządzeniami. W naszym przypadku, UART został wykorzystany do przesyłania informacji z GUI do mikrokontrolera ARM, takich jak konfiguracja planszy, punkt startowy i końcowy.

## 4.3 GUI

W celu stworzenia interaktywnego interfejsu użytkownika dla naszego projektu zdecydowaliśmy się skorzystać z biblioteki Tkinter w języku Python. Tkinter jest popularną biblioteką do tworzenia aplikacji graficznych w Pythonie, która oferuje wiele narzędzi i elementów interfejsu do łatwego projektowania interaktywnych aplikacji. W naszym przypadku, Tkinter został wykorzystany do stworzenia graficznego interfejsu użytkownika (GUI), który umożliwia

manipulację planszą dla algorytmu A*. Link do dokumentacji biblioteki Tk `https://docs.python.org/3/library/tk.html`.

### 4.3.1 Implementacja interfejsu języku Python

W naszym projekcie zaimplementowaliśmy interfejs użytkownika w bibliotece Tkinter. Stworzyliśmy okno aplikacji, które wyświetlało macierz o wymiarach 20 na 20. Macierz ta reprezentuje planszę, na której można umieszczać przeszkody oraz zdefiniować punkt startowy i końcowy. Dzięki temu użytkownik może wizualnie manipulować planszą, ustalając konfigurację dla algorytmu A*.

Znaczenia kolorów:

- biały - pole neutralne
- czarny - przeszkoda, algorytm nie może użyć tego pola w wyznaczaniu ścieżki
- pomarańczowy - pole startu
- różowy - pole końca
- zielony - wyznaczona ścieżka
- żółty - pola listy otwartej
- szary - pola listy zamkniętej

Przycisk z napisem **CLEAR** czyści plansze pozwalając na nowo zdefiniować punkt startu, stopu oraz nowe przeszkody. Przycisk z napisem **SOLVE** wywołuje wysłanie danych do mikroprocesora oraz wyświetlenie wyników.

### 4.3.2 Komunikacja przez UART przy użyciu biblioteki *serial* w Pythonie

Aby umożliwić komunikację między mikrokontrolerem ARM a interfejsem użytkownika w Pythonie, zastosowaliśmy bibliotekę *serial*. *Serial* jest popularną biblioteką w języku Python, która umożliwia obsługę komunikacji szeregowej, w tym transmisję danych za pomocą interfejsu UART.

W naszym projekcie, biblioteka *serial* została wykorzystana do ustanowienia połączenia między GUI a mikrokontrolerem ARM. W GUI, po otwarciu odpowiedniego portu szeregowego za pomocą biblioteki *serial*, możemy wysyłać dane w formacie tekstowym, takie jak konfiguracja planszy, punkt startowy i punkt końcowy, do mikrokontrolera. Mikrokontroler odbiera te dane i może na ich podstawie podjąć odpowiednie działania.

# 5 Prezentacja działania

## 5.1 Środowiko testowe Vivado

Projekt został zrealizowany przy użyciu środowiska Vivado, które jest potężnym narzędziem do projektowania, implementacji i weryfikacji układów FPGA. W ramach projektu, opracowano klarowny i optymalny kod dla części FPGA, zapewniając efektywne wykorzystanie zasobów i wysoką wydajność systemu.


Rysunek 5: Przebiegi cyfrowe algorytmu A*

Rysunek 5 przedstawia przebiegi sygnałów cyfrowych pracy algorytmu A*.

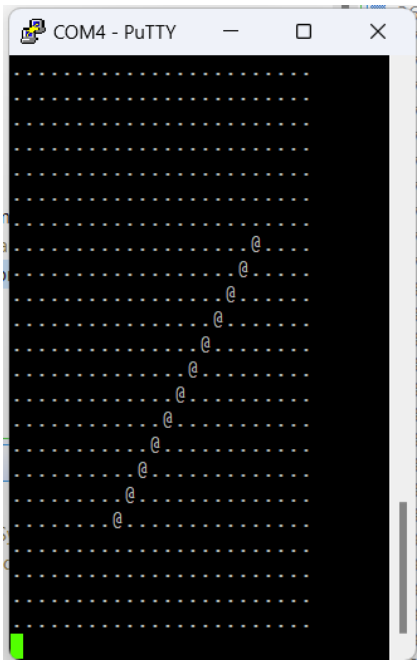
## 5.2 Testowa wizualizacja algorytmu w konsoli UART

Istnieje również możliwość obserwacji działania algorytmu nie tylko poprzez interfejs graficzny, ale również za pomocą konsoli UART (rysunek 6, 7). Dzięki tej funkcjonalności można monitorować logi i wyniki działania algorytmu bez konieczności korzystania z interfejsu graficznego. Jest to szczególnie przydatne w przypadku uruchamiania projektu w środowiskach, gdzie interfejs graficzny nie jest dostępny lub nie jest wymagany.

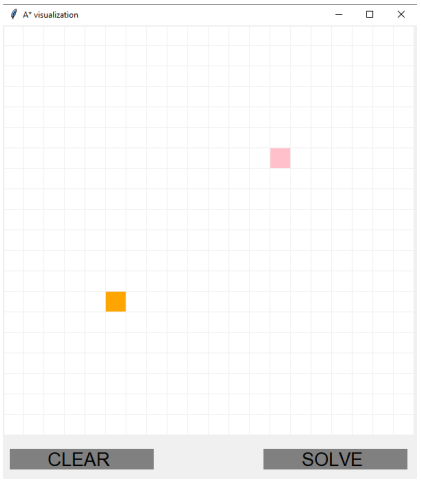
## 5.3 GUI

W ramach prezentacji projektu, zaprezentowano wizualizacje działania algorytmu A* za pomocą interfejsu graficznego. Do tego celu wykorzystano cztery rysunki 8, 9, 10 oraz 11 , które ilustrują różne przypadki. Rysnuki 8 i 9 przedstawiają podstawowy scenariusz bez przeszkód, gdzie algorytm znajduje optymalną ścieżkę. Rysnuki 10 oraz 11 przedstawiają bardziej skomplikowany przypadek z obecnością przeszkody, w którym algorytm musi znaleźć alternatywną ścieżkę wokół przeszkody.

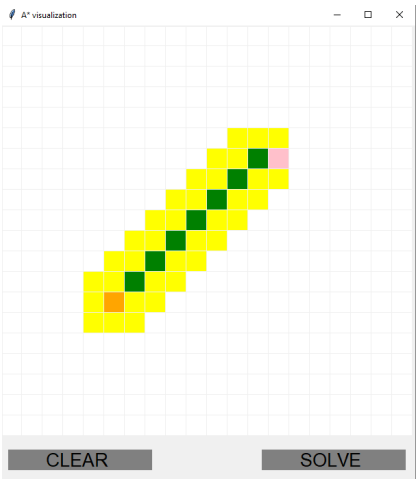

Rysunek 6: Testowa wizualizacja algorytmu w konsoli UART, przed



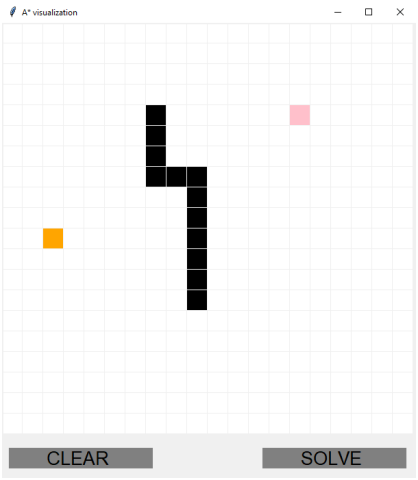

Rysunek 7: Testowa wizualizacja algorytmu w konsoli UART, po



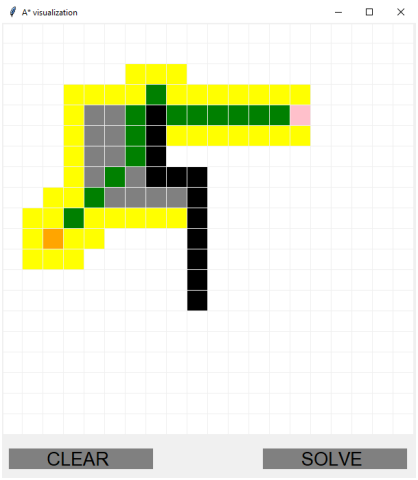

Rysunek 8: Przykład prosty, przed




Rysunek 9: Przykład prosty, po




Rysunek 10: Przykład prosty, przed




Rysunek 11: Przykład prosty, po

# 6 Podsumowanie

W ramach naszego projektu udało nam się zrealizować interaktywny system do wizualizacji i wykonywania algorytmu A* na platformie Arty Z7 opartej na układzie Zynq-7000. Mimo kilku trudności, z którymi się spotkaliśmy, udało nam się skutecznie rozwiązać wszystkie problemy i osiągnąć zamierzone cele.

Nasz projekt dostarcza praktyczny dowód na wykorzystanie platformy Zynq w implementacji algorytmów i interfejsów użytkownika. Interaktywny system do wizualizacji i wykonywania algorytmu A* na platformie Arty Z7 otwiera możliwości dla dalszego rozwoju i zastosowania w różnych dziedzinach, takich jak robotyka, systemy nawigacji czy analiza danych.