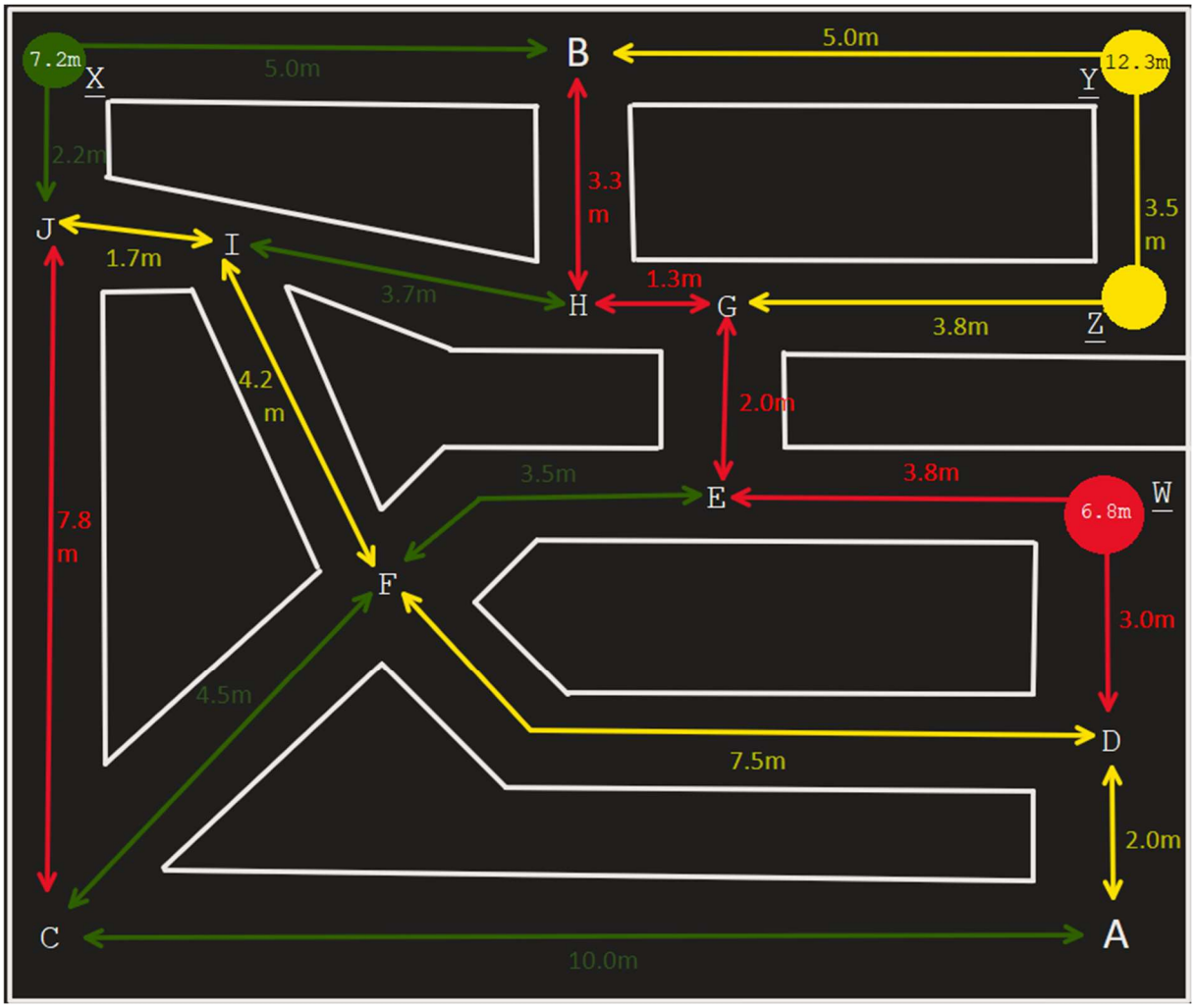
Zadanie na 2-gi etap rekrutacji do działu autonomii PWR Racing Team – Opis rozwiązania



1. **Przygotowanie mapy hangaru**

Rozwiązanie zadania rozpocząłem od przygotowania mapy hangaru, wszystkie skrzyżowania dróg oznaczyłem literami od A do J (pozostawiając oryginalnie oznaczone skrzyżowania A oraz B), a miejsca w, których trasa tylko skręcała literami od W do X. Następnie zmodyfikowałem trzy fragmenty trasy:

- połącznie D – W – E zastąpiłem połączeniem D – E o łącznej długości 6.8m
- połączenie B – X – J zastąpiłem połączeniem B – J o łącznej długości 7.2m
- połączenie B – Y – Z – G zastąpiłem połączeniem B – G o łącznej długości 12.3m

Takie modyfikacje pozwalają na optymalizację działania programu, który musi rozpatrzyć połączenia między 10 a nie 14 punktami.

2. Sposób rozwiązania problemu

Do rozwiązania problemu znalezienia optymalnej trasy użyłem algorytmu Dijkstry. Algorytm składa się z kilku kroków:
1) Nadanie węzłowi początkowemu odległości równej zero a pozostałym odległości równej nieskończoność, oznaczenie węzła początkowego jako aktualny;
2) Dla aktualnego węzła wyznaczenie odległości do jego sąsiadów i porównanie jej z odległością aktualnie przypisaną, jeżeli nowa wartość jest mniejsza od aktualnie przypisanej, nadpisanie wartości;
3) Po wyznaczeniu odległości do wszystkich sąsiadów oznaczenie aktualnego węzła jako odwiedzony, odwiedzony węzeł nie będzie więcej rozpatrywany;
4) W przypadku oznaczenia końcowego węzła jako odwiedzony zakończenie pracy algorytmu;
5) Jeżeli węzeł końcowy nie był jeszcze odwiedzony wybranie jednego z nieodwiedzonych węzłów, oznaczenie go jako aktualny i powrót do kroku 2).

3. Opis programu

Program jest napisany w Pythonie i składa się z trzech funkcji:

- funkcja **load()** służy do pobrania od użytkownika koszt przejazdu jednego metra po danym rodzaju nawierzchni, a następnie wartość podaną przez użytkownika przypisuje do odpowiedniej zmiennej globalnej. Funkcja obsługuje przypadki takie jak podanie ujemnej wartości (komunikat informujący, że koszt nie może być ujemny), podanie ciągu znaków nie będącego liczbą (komunikat informujący, że koszt musi być liczbą) oraz podanie liczby zmiennoprzecinkowej ze znakiem przecinka zamiast kropki (program zastępuje przecinek kropką).

- funkcja **make_hangar(g, y, r)** zwraca słownik, który jest grafową reprezentacją mapy hangaru. W słowniku tym kluczem jest skrzyżowanie (oznaczone literą), a wartością zagnieżdżony słownik w, którym kluczami są osiągalne skrzyżowania, a wartościami koszty przejazdu między danymi skrzyżowaniami. Funkcja przyjmuje trzy argumenty, które są kosztami przejazdu po każdym rodzaju nawierzchni i na ich podstawie wyznacza wagę każdego połączenia według wzoru: *koszt * odległość,* gdzie *koszt* to koszt przejazdu dla każdej nawierzchni (odpowiednio g dla zielonej, y dla żółtej, r dla czerwonej), a *odległość* to odległość podana w metrach.

- funkcja **set_min(graph, start, end)** jest implementacją algorytmu Dijkstry, wyznacza minimalny koszt przejazdu z punktu startowego do pozostałych punktów. Przyjmuje trzy argumenty: *graph* - słownik utworzony przez funkcję make_hangar(), *start* – punkt początkowy (w programie domyślnie ustawiony punkt A), *end* – punkt końcowy (w programie domyślnie ustawiony jako punkt B). Po wyznaczeniu kosztów połączeń funkcja wypisuje koszt połączenia z punktem *end* zaokrąglony do trzeciego miejsca po przecinku.