



DENSO

Crafting the Core

デンソークリエイト 1DAYインターンシップ 迷路走行プログラムの手引き

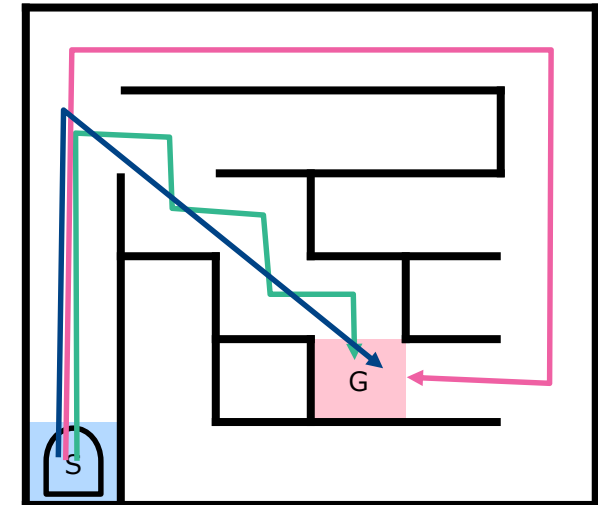
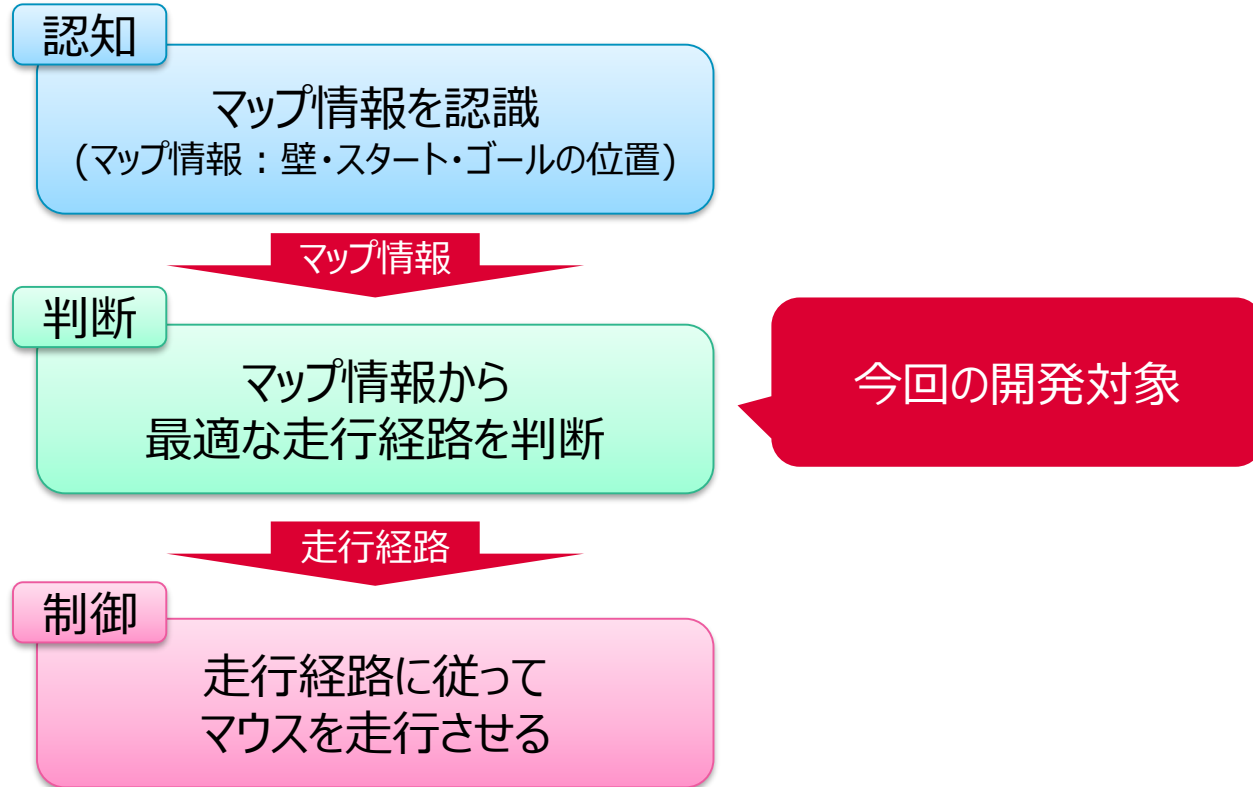
CONFIDENTIAL
関係者外秘

1. マイクロマウスの迷路走行プログラム概要
2. 最短経路探索
 1. 概要
 2. 歩数マップ作成
 3. 走行経路作成
3. 提供するインタフェース
4. 一歩進んだ経路探索

マイクロマウスの迷路走行プログラム概要

CONFIDENTIAL
関係者外秘

マイクロマウスは、大まかに以下の流れで処理を行いゴールまで走行させます。
今回は、そのうち**走行経路判断**の一部を開発してもらいます。

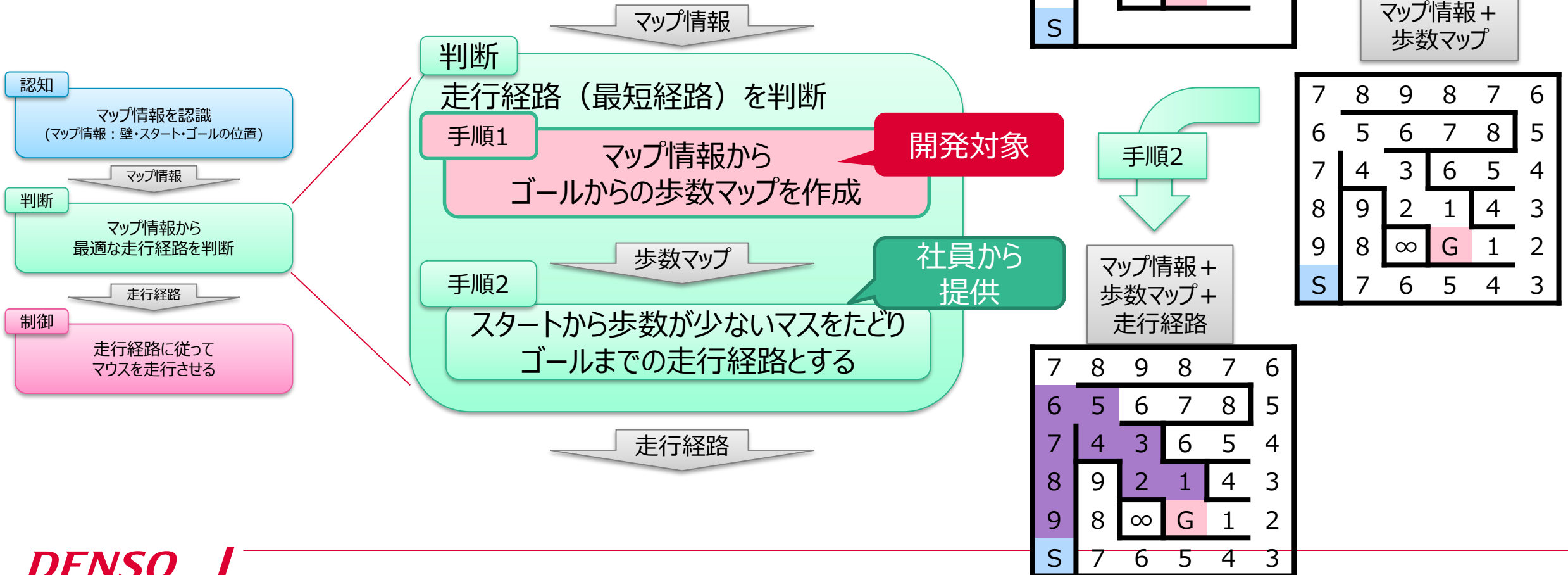


経路の取り方はいろいろ。
最適なコース取りの判断も
マイクロマウス開発の重要な要素の一つ。

最短経路探索（概要）

走行経路判断の方針は様々ありますが、ここでは移動距離が最短となる経路（最短経路）を求めることにします。

最短経路は大きく以下の2手順で求められます。

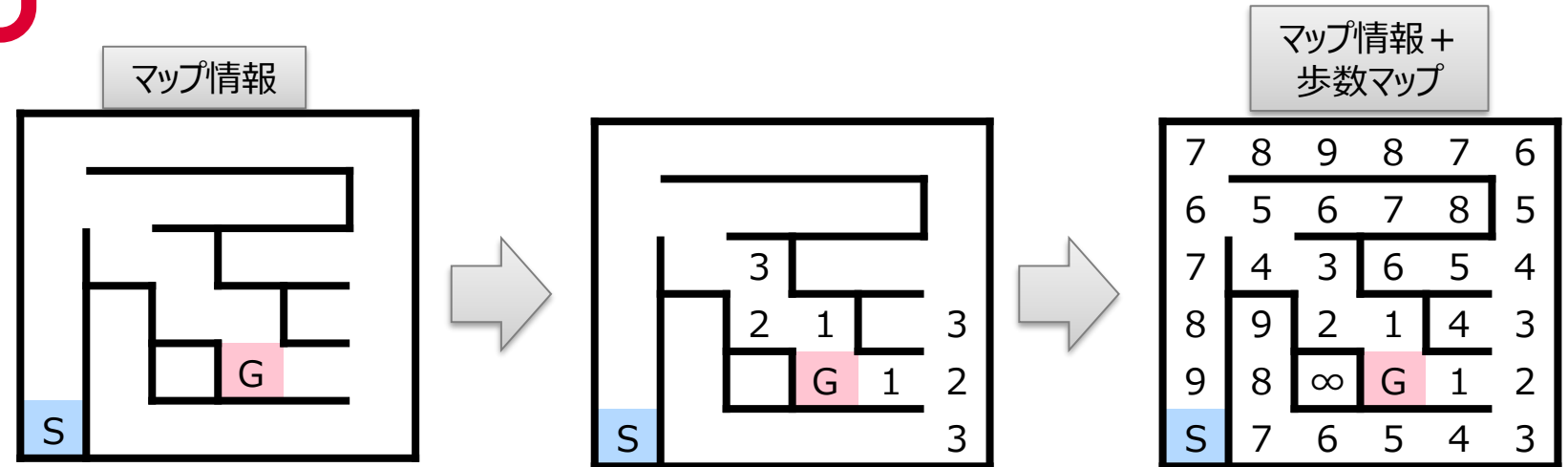
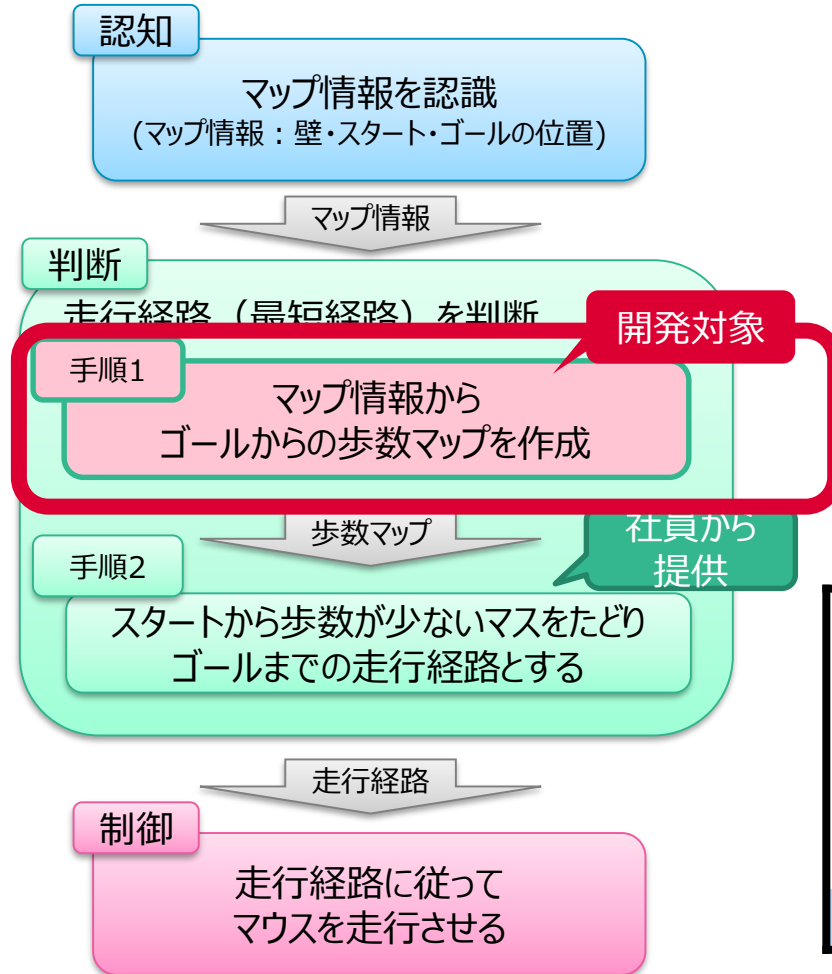


最短経路探索（歩数マップ作成）

CONFIDENTIAL
関係者外秘

手順1では、マップ情報をもとに、各マスの中のゴールからの歩数を求めます。

ゴールのマスを0とし、そこから1歩ですすめるマスを1、さらにそこから1歩ですすめるマスを2、というように進めていきます。これをすべての経路で繰り返すと、歩数マップが完成します。複数の経路でたどり着けるマスについては、歩数が少ない方が採用されるようにしましょう。



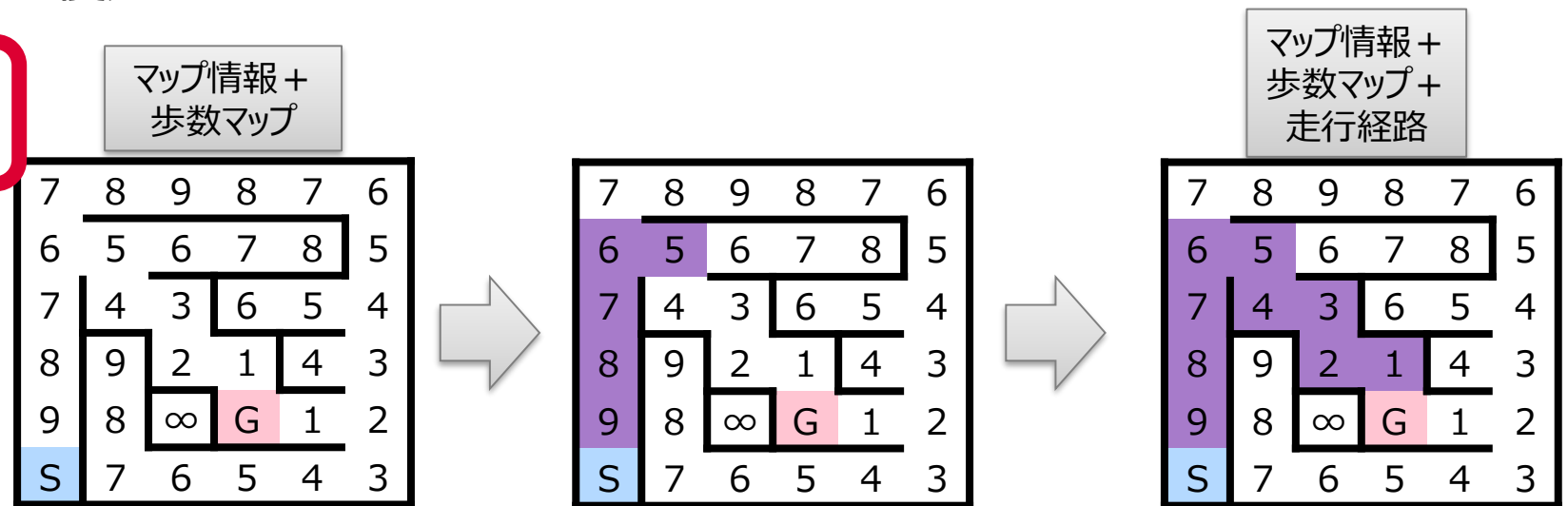
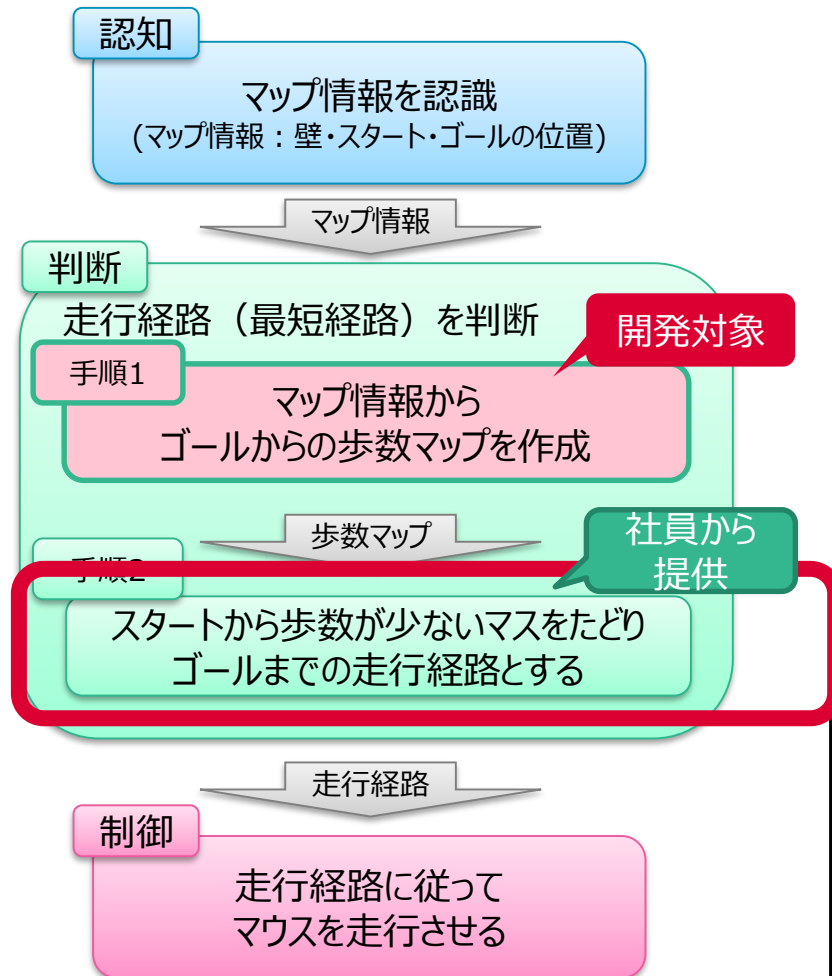
最短経路探索（走行経路作成）

手順2では、歩数マップをもとに、最短経路となる走行経路を求めます。

スタート地点から順に、移動できるマスのうち、ゴールまでの歩数が一番少ないマスへ移動します。

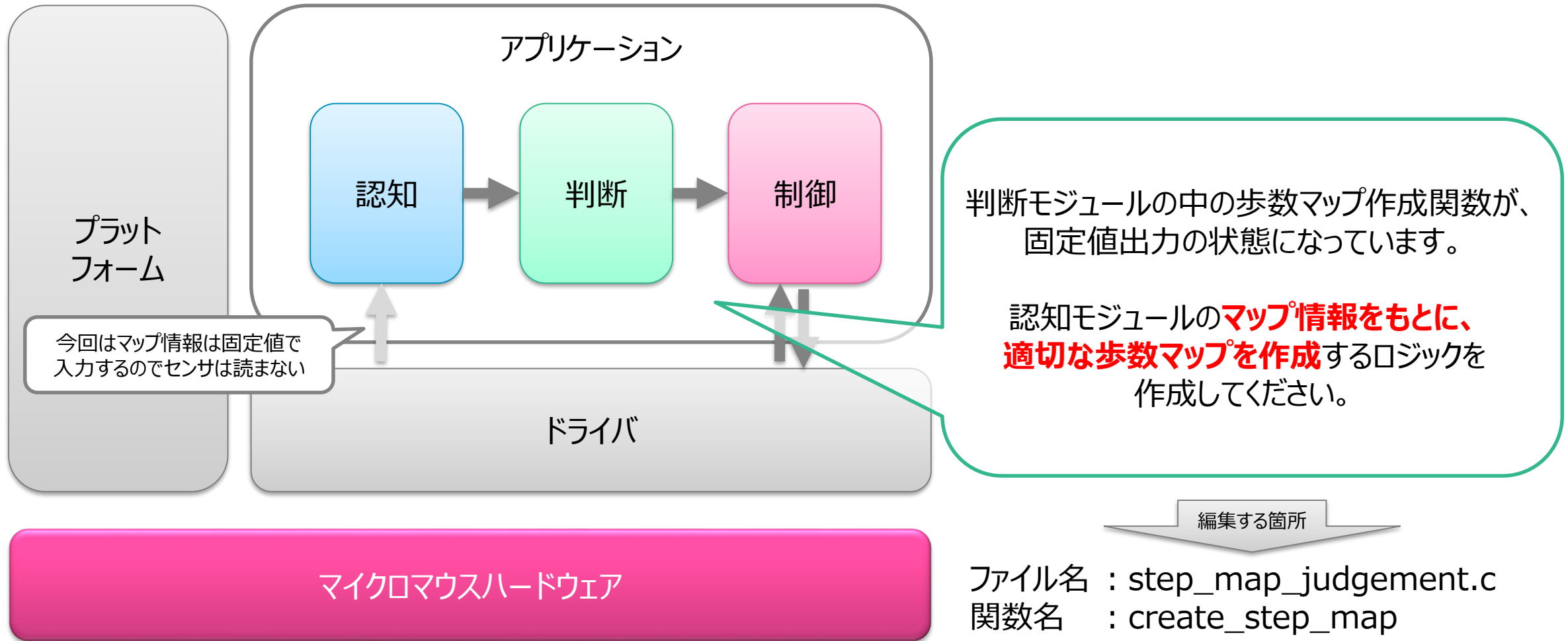
下記例では出てきませんが、コース次第で、上記条件を満たすマスが複数になる場合もあるため、注意が必要です。

提供する走行経路作成関数ではその場合、直進を優先し、次に右折を優先します。

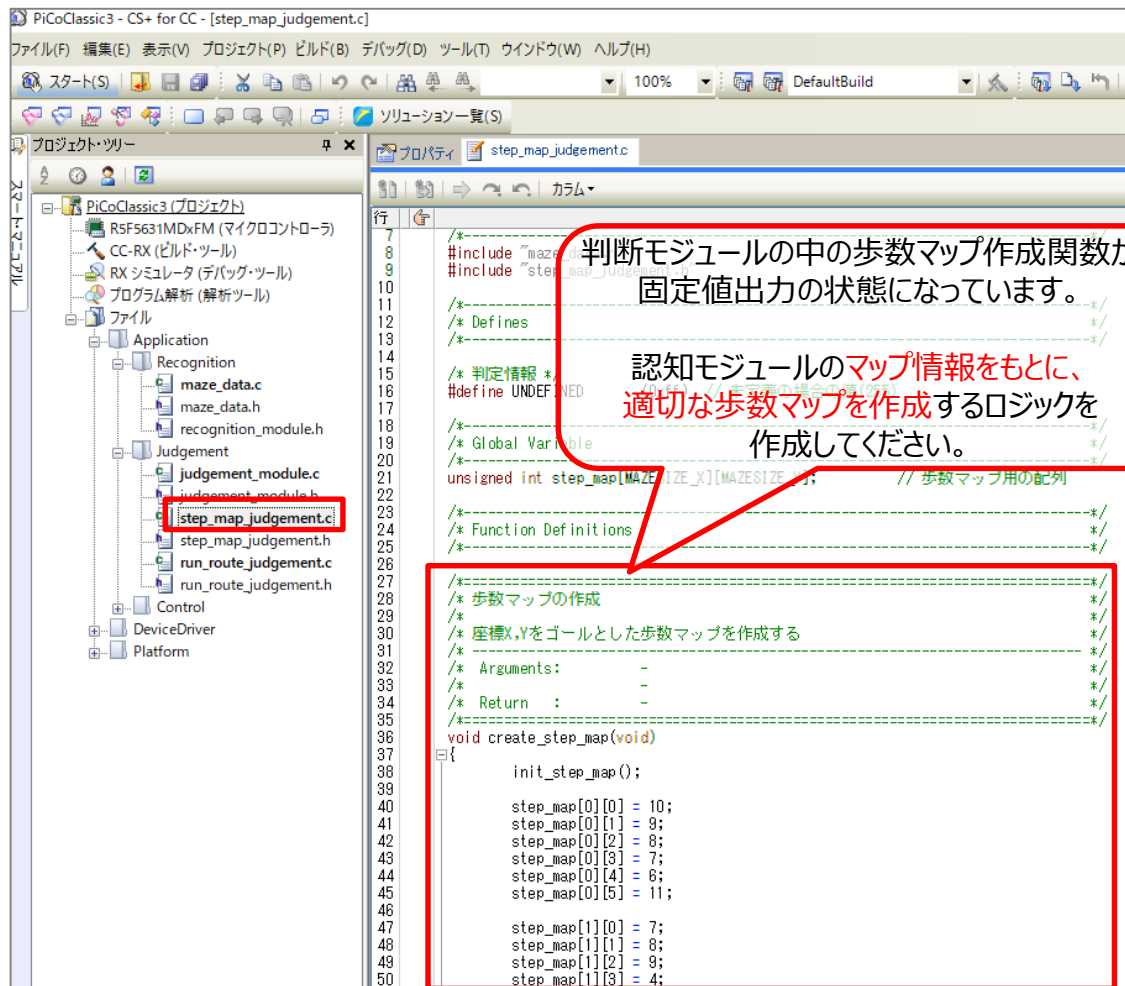


ベースプログラムの構成

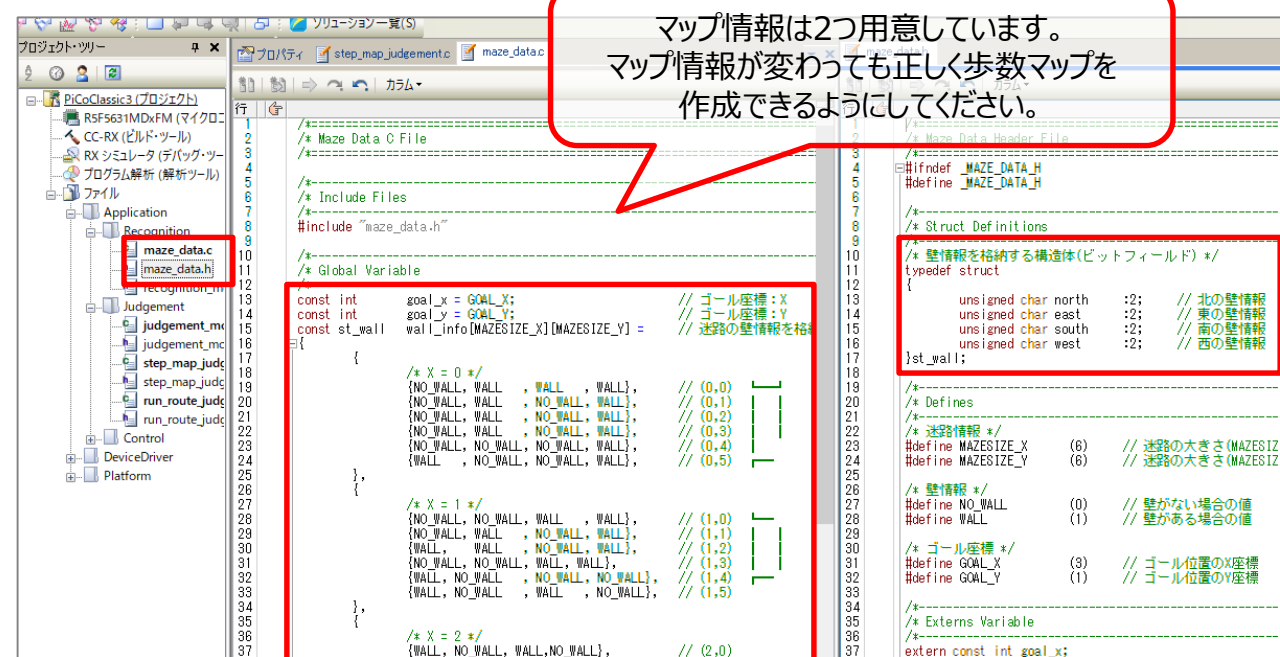
CONFIDENTIAL
関係者外秘



■ 歩数マップ作成関数



■ マップ情報



■ 使用例

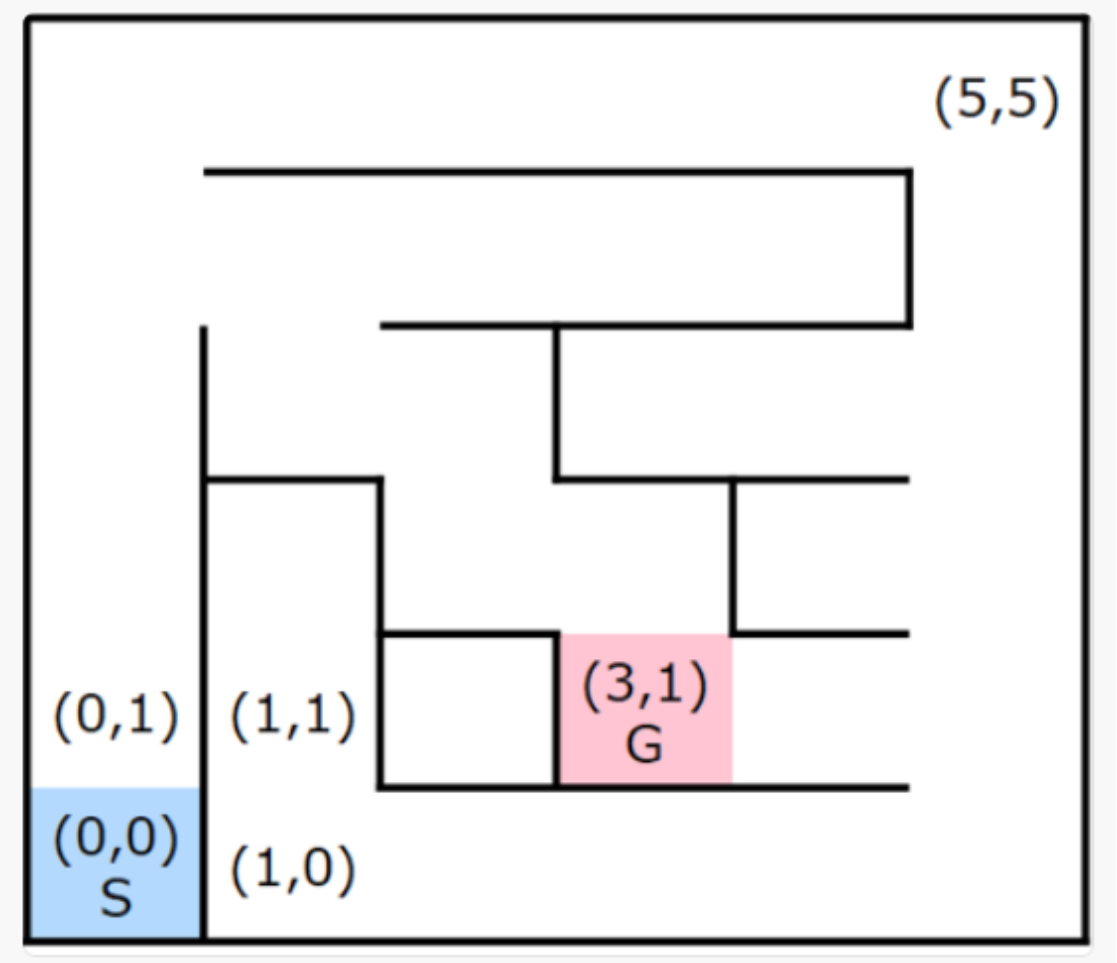
```

if(wall_info[goal_x][goal_y].north == NO_WALL){
    // ゴール座標のマスの上に壁がないときの処理
}
    
```


探索対象のマップ情報

CONFIDENTIAL
関係者外秘

x,yの原点は左下です



(応用編)一歩進んだ経路探索

CONFIDENTIAL
関係者外秘



最短経路が必ず最速になるの？



そうとも限りません。

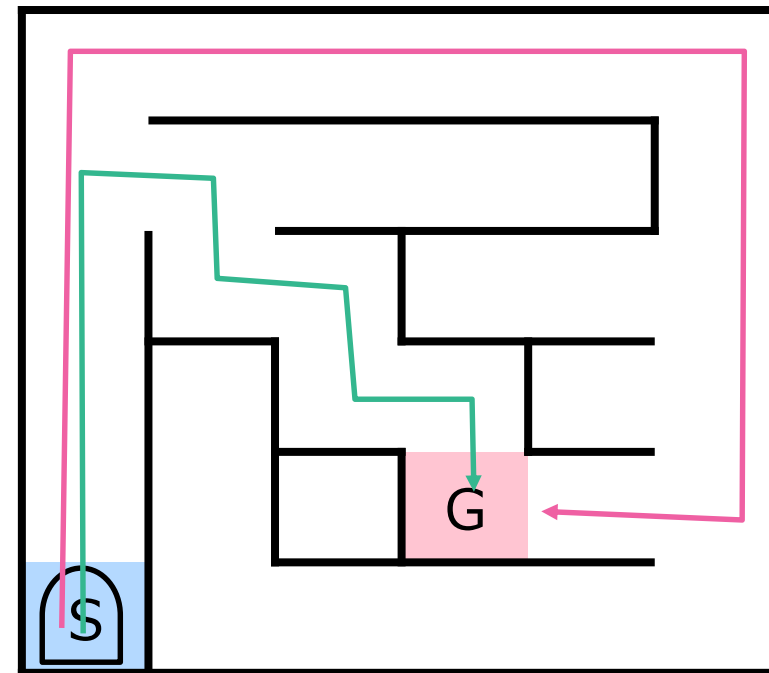
サンプル動画をみてみると、提供するインタフェースでの直進・旋回は、1マス進む時間は、旋回より直進のほうが早いです。

よって、より旋回回数の少ない経路を進んだほうが早くゴールする可能性もあります。



歩数マップの作り方次第で、そのほかの部分を変更せずに

旋回回数の少ない経路にすることもできるので、挑戦してみましょう！



(応用編)提供するインタフェース(1/4)

■ 一覧

概要	シンタックス	機能
直進	<code>void run_straight(int distance, float target_speed)</code>	指定した距離を、指定した速度になるように加減速しながら直進する。
旋回	<code>void run_rotate(int rotate_direction, int rotate_angle)</code>	指定した方向に指定した角度だけ旋回する。
ブザー鳴動	<code>void sound_buzzer(int frequency, int ms_time)</code>	指定した周波数で、指定した時間ブザーを鳴らす。
LED点灯/消灯	<code>void display_led(int led_mode)</code>	指定した値の下位4ビットに合わせて4つのLEDを点灯/消灯させる。
スイッチ状態取得	<code>int get_switch_state(int switch_kind)</code>	指定したスイッチの状態を取得する。

(応用編)提供するインタフェース(2/4)

■ 直進

- シンタックス
 - void run_straight(int distance, float target_speed)
- 機能
 - 指定した距離を、指定した速度になるように加減速しながら直進する。

• 引数

型	名前	値域	概要
int	distance	0 – 1000[mm]	距離
float	speed	0 – [mm/sec]	目標速度

- 戻り値
 - なし

- 必要なヘッダ
 - tire_control.h

■ 旋回

- シンタックス
 - void run_rotate(int rotate_direction, int rotate_angle)
- 機能
 - 指定した方向に指定した角度だけ旋回する。

• 引数

型	名前	値域	概要
int	rotate_direction	TURN_RIGHT TURN_LEFT	旋回方向
int	rotate_angle	0 – 180[度]	旋回角度

- 戻り値
 - なし

- 必要なヘッダ
 - tire_control.h

(応用編)提供するインタフェース(3/4)

■ ブザー鳴動

- シンタックス

- void sound_buzzer(int frequency, int ms_time)

- 機能

- 指定した周波数で、指定した時間ブザーを鳴らす。

- 引数

型	名前	値域	概要
int	frequency	0 – 1000[Hz]	周波数
int	ms_time	0 – [ms]	鳴動時間

- 戻り値

- なし

- 必要なヘッダ

- buzzer_driver.h

■ LED点灯/消灯

- シンタックス

- void display_led(int led_mode)

- 機能

- 指定した値の下位4ビットに合わせて4つのLEDを点灯/消灯させる。

- 引数

型	名前	値域	概要
int	led_mode	0x00 – 0x0F	下位4ビットのビット毎に4つのLEDの点灯/消灯に対応 1:点灯 0:消灯

- 戻り値

- なし

- 必要なヘッダ

- tire_control.h

(応用編)提供するインタフェース(4/4)

■ スイッチ状態取得

- シンタックス
 - int get_switch_state(int switch_kind)
- 必要なヘッダ
 - switch_driver.h
- 機能
 - 指定したスイッチの状態を取得する。
- 引数

型	名前	値域	概要
int	switch_kind	MODE_SW_RED : 赤いスイッチ MODE_SW_YELLOW : 黄色いスイッチ MODE_SW_BLUE : 青いスイッチ	状態を取得したいスイッチ種別

• 戻り値

型	値域	概要
int	SW_ON : 押されている SW_OFF : 押されていない	スイッチの状態

DENSO

Crafting the Core