State machine: Robocup summary

Map:

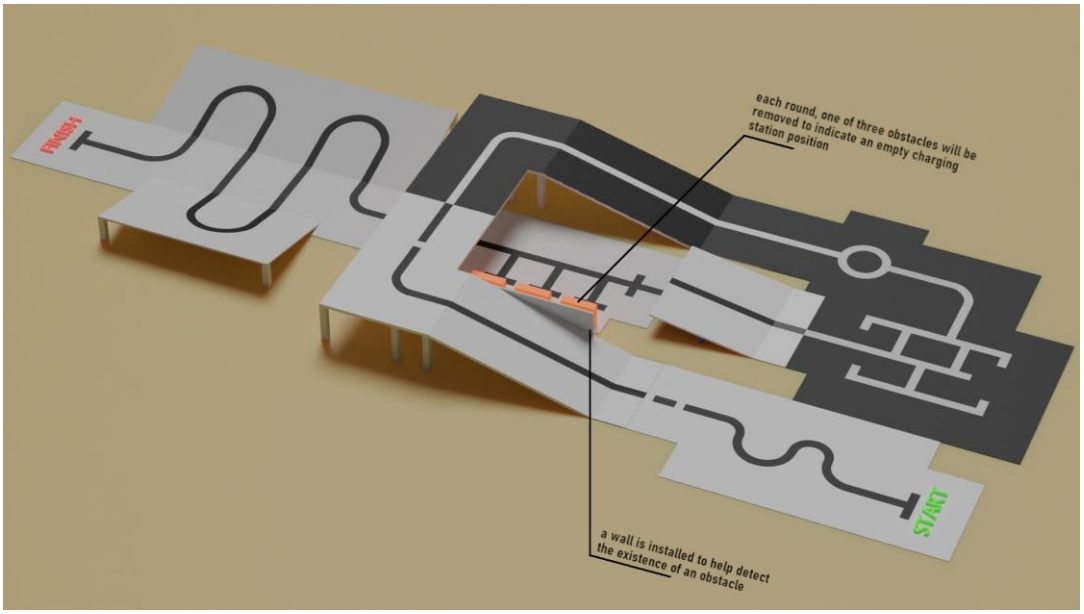

Figure 1: RoboCup map

State machine:

A state machine is a behavior model. It consists of a finite number of states and is therefore also called finite-state machine (FSM). Based on the current state and a given input the machine performs state transitions and produces outputs.

The state machine UML design: RoboCup competition:

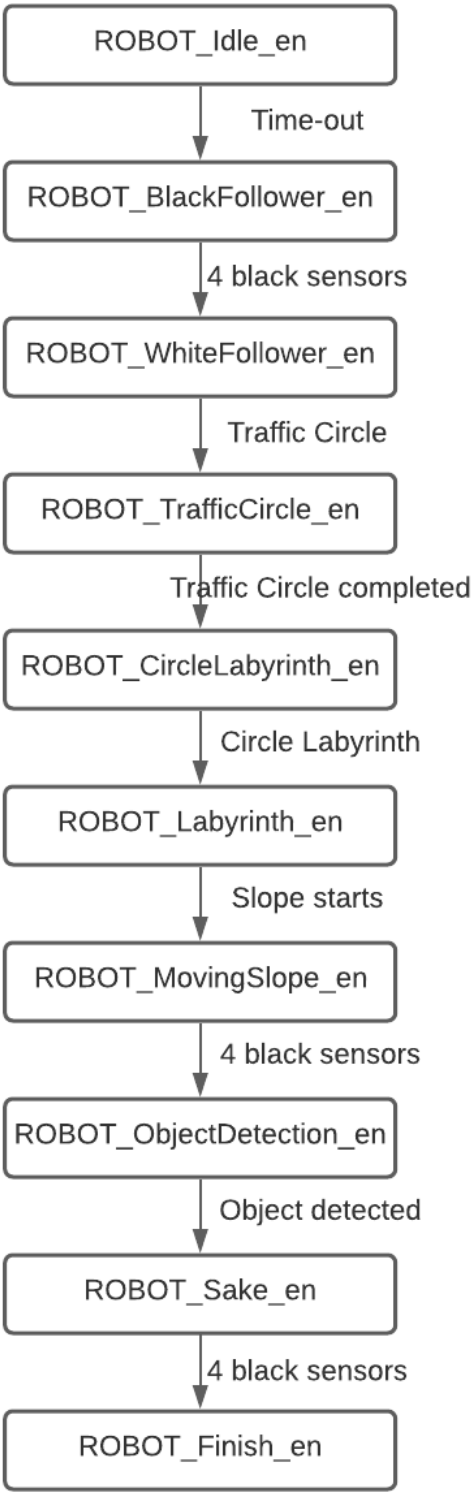

Figure 2: UML presentation of the state machine

Code Implementation:

Macros:

```
#define SENSOR_1     2
#define SENSOR_2     3
#define SENSOR_3     4
#define SENSOR_4     5
#define LED_RED      6
#define LED_GREEN    8
#define TRIGGER_PIN  9
#define ECHO_PIN    12
#define MAX_DISTANCE 400
#define BLACK        1
#define WHITE        0
```

Variables:

```
static uint8_t ROBOT_LeftSensorState_u8 = 0,
               ROBOT_LeftCenterSensorState_u8 = 0,
               ROBOT_RightCenterSensorState_u8 = 0,
               ROBOT_RightSensorState_u8 = 0,
               ROBOT_CenterSensorState_u8 = 0;
static uint32_t ROBOT_StartTime_u32 = 0;
static float SOUND_Distance_f = 0.0;
static float SOUND_Duration_f = 0.0;
static float SOUND_SPEED_M_f = 0.0;
static float SOUND_SPEED_CM_f = 0.0;
float ROBOT_Distance_f = 0.0;
static bool ROBOT_TrafficCircle_b = false;
static bool ROBOT_TrafficCircleComplete_b = false;
static bool ROBOT_ObjectDetected_b = false;
static bool ROBOT_CircleLabyrinth_b = false;
static bool ROBOT_MovingSlopeStart_b = false;
const uint8_t ULTRASONIC_ITERATIONS = 5;

static uint32_t previousTime = 0;
static bool timeOut = false;
```

Define and declare the state machine:

```
typedef enum
{
  ROBOT_Idle_en,
  ROBOT_BlackFollower_en,
  ROBOT_WhiteFollower_en,
  ROBOT_TrafficCircle_en,
  ROBOT_CircleLabyrith_en,
  ROBOT_Labyrinth_en,
  ROBOT_MovingSlope_en,
  ROBOT_ObjectDetection_en,
  ROBOT_Snake_en,
  ROBOT_Finish_en
} ROBOT_States_ten;

ROBOT_States_ten ROBOT_CurrentState_en = ROBOT_Idle_en;
```

Declare functions:

```
void ROBOT_vReadSensors();

void ROBOT_vForward();
void ROBOT_vRight();
void ROBOT_vLeft();
void ROBOT_vBackward();

void ROBOT_vForward1();
void ROBOT_vRight1();
void ROBOT_vLeft1();
void ROBOT_vBackward1();
void ROBOT_vForward5();
void ROBOT_vRight5();
void ROBOT_vLeft5();
void ROBOT_vBackward5();

void ROBOT_vBlackFollower();
void ROBOT_vWhiteFollower();
void ROBOT_vTrafficCircle();
void ROBOT_vCircleLabyrinth();
void ROBOT_vLabyrinth();
void ROBOT_vMovingSlope();
void ROBOT_vObjectDetection();
float ROBOT_fReadDistance();
void ROBOT_vWhiteFollower1();
```

Execute the state machine:

```
Switch (ROBOT_CurrentState_en)
{
    Case ROBOT_Idle_en:
    {
        ROBOT_vForward();
        If Time-out:
            ROBOT_Current_en = ROBOT_BlackFollower_en;
    } break;
    Case ROBOT_BlackFollower_en:
    {
        ROBOT_vBlackFollower();
        If 4 black sensors:
            ROBOT_Current_en = ROBOT_WhiteFollower_en;
    } break;
    Case ROBOT_WhiteFollower_en:
    {
        ROBOT_vWhiteFollower1();
        If Traffic Circle:
            ROBOT_Current_en = ROBOT_TrafficCircle_en;
    } break;
    Case ROBOT_TrafficCircle_en:
    {
        ROBOT_vTrafficCircle ();
        If Traffic Circle Complete:
            ROBOT_Current_en = ROBOT_CircleLabyrith_en;
```

```
} break;
Case ROBOT_CircleLabyrith_en:
{
        ROBOT_vCircleLabyrinth ();
        If Circle Labyrinth:
                ROBOT_Current_en = ROBOT_Labyrinth_en;
} break;
Case ROBOT_Labyrinth_en:
{
        ROBOT_vLabyrinth ();
        If Moving slope starts:
                ROBOT_Current_en = ROBOT_MovingSlope_en;
} break;
Case ROBOT_MovingSlope_en:
{
        ROBOT_vMovingSlope ();
        If Moving slope starts:
                ROBOT_Current_en = ROBOT_MovingSlope_en;
} break;
Case ROBOT_MovingSlope_en:
{
        ROBOT_vMovingSlope ();
        If 4 black sensors:
                ROBOT_Current_en = ROBOT_ObjectDetection_en;
} break;
```

```
    Case ROBOT_ObjectDetection_en:
    {
        ROBOT_vMovingSlope ();
        If Object is detected:
            ROBOT_Current_en = ROBOT_Snake_en;
    } break;
    Case ROBOT_Snake_en:
    {
        ROBOT_vBlackFollower();
        If 4 black sensors:
            ROBOT_Current_en = ROBOT_Idle_en;
    } break;
    Default:
    {
    }
}
```

Some coding rules:

- Avoid hard coding:
  WRONG: if (a < 10) …
  RIGHT:
  a- Declare a macro: #define MAX_NUMBER 10
  b- if (a < MAX_NUMBER) …
- Each variable should begin with the system name: ROBOT_xxxxx or LINE_FOLLOWER_xxxxx or 3JEJA_xxxxx
- Each variable should end with suffix indicating the variable type: ROBOT_xxxxxx_u8 to say that this variable's type is uint_8
- Each function should precise the return type: XXXX_vXXXXX() to say that the return type of this function is void