EXOTEC

Gladiator features and API

EXOLEGEND - EXOTEC - Version v1.0

Contents

1 Version history

1.1 Version v1.0

- Initial version : Gladiator version 1.9.0
- All functions in different chapters (Game, Maze, Robot, Control, Weapon)
- Remove some functions, and simplify the API
- Add graph representation of the maze
- Correction of the Phase explanation picture
- Add a paragraph that explain the two different modes

2 System overview

The gladiator robot is used by players to participate to the Exolegend games. it is conceived around ESP32S3 microcontroller. The GLadiator Library is written in C++ with the aim of being used with Arduino IDE.

2.1 Definitions

- **Maze** : The field where robots play (playground)
- **World** : Game Arena Struture
- **Arena Screen** : Control screen of the World. It's the small touch screen of the World. It's able to start a new game and identify new players.
- **Game Master** : The software which controls the whole game
- **Gladiator** : The robot used by players.
- **Ghost** : The simulated robot
- **Gladiator Library** : API used to control the robot
- **ID** : Identifier of the robot, it is also the tag ID on the top of the robot. This ID is unique.

2.2 Gladiator Library

Gladiator Library is an API allowing the user to control the Gladiator. This Library provides some functions to control the robot when it's connected to an Arena.

2.2.1 System environment

The Gladiator is built around a ESP32S3 microcontroller.

You need PlatformIO to be able to compile gladiator Library. There is two ways to compile your code :

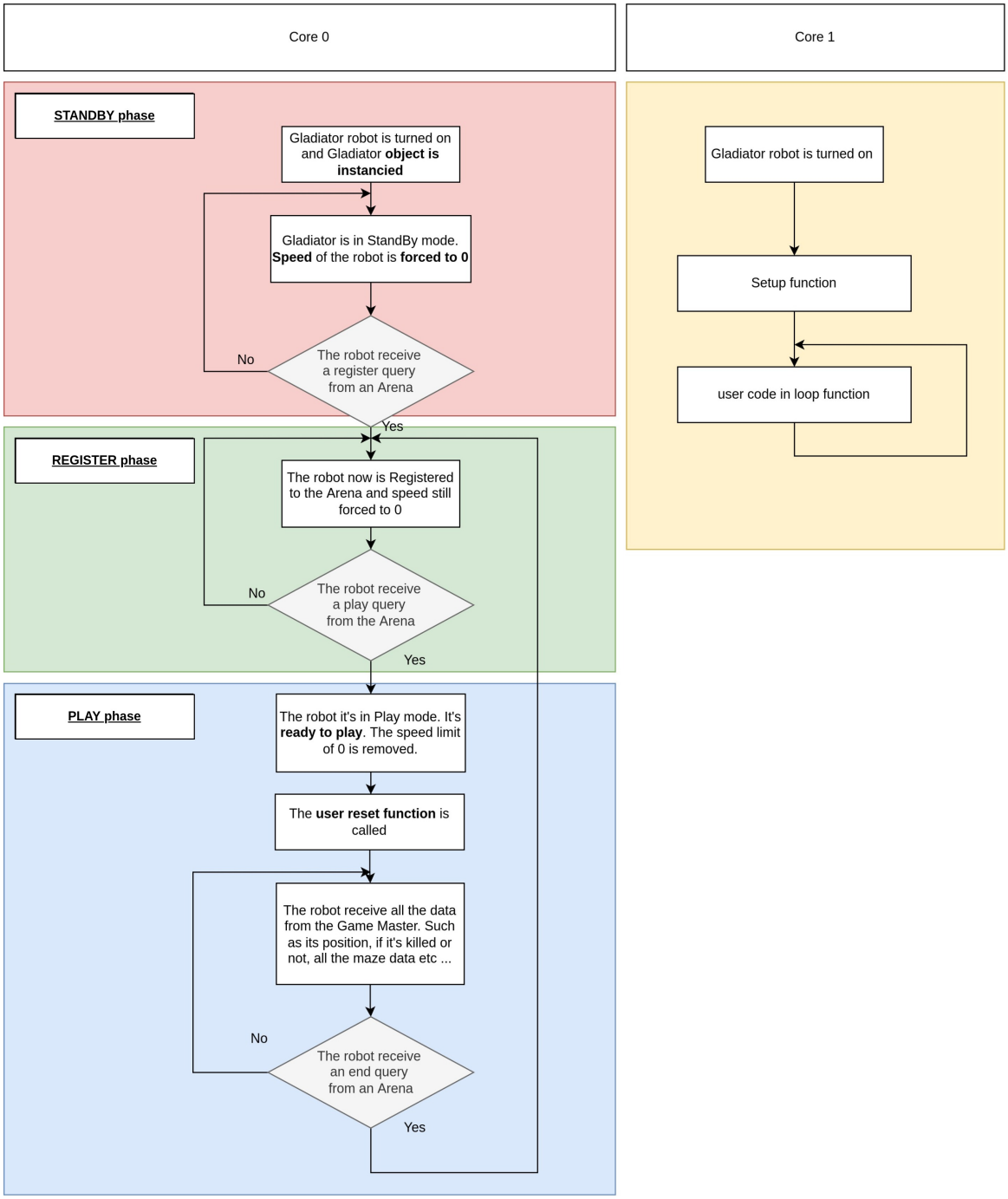
- Compile your code to be run as a Ghost on your computer.
- Compile your code to be flashed in a Gladiator.

Your code is compiled with gnu++17. All the functions needed to control the robot are in the Gladiator library. You can use std functions, but be aware that all dynamic allocation mechanisms can cause memory overflows on esp32 (like vectors, strings, etc).

2.2.2 Code execution

When you include the Gladiator library and instantiate a gladiator object, a thread runs in the second core of the microcontroller. This thread runs in the background of the user code and communicates with Arena, it also calculates the speed of the robot and filters its position. The user code is executed on the first core.

2.2.3 Gladiator life cycle



2.3 How to start ?

This section provides a basic code template which user can copy to start using Gladiator:

```
#include <gladiator.h>
void reset();
Gladiator* gladiator;
void setup() {
    gladiator = new Gladiator();
    gladiator->game->onReset(&reset);
    // setup your data after turning on the robot
}
void reset() {
    gladiator->log("Reset function called");
    // reset your data before a game start
}
void loop() {
    if(gladiator->game->isStarted()) {
        // Write your strategy code here
    }
}
```

2.4 The two ways of using a Gladiator

There are two modes available to control the robot:

- **Arena mode** This is the default mode of the Gladiator. The user code is automatically executed after registering and starting a new game.
- **Free mode** The user code is executed even if the robot is not registered to achieve some tests without connecting the robot to an Arena.

3 Features

3.1 Control the speed of the robot

The user can control the speed of each wheel. The max speed allowed by Gladiator Library is 1m/s. The speed control loop is handled by the Gladiator library. User can change the parameters of the PID of each wheel.

3.2 Get robot data

The user can get the data of the robot such as its position, it's ID, life status etc ... The user is able to know the data of the other robots in the field but with an estimate delay in a range of [20ms, 100ms].

3.3 Get maze data

The user can get the data of the maze such as the position of each wall and each remaining coins in the field.

3.4 Tracking data

The robot allows the user to log data in the Serial monitor or to track all the data in a live mode to see them in their computer screen while the robot is playing on the maze. The user can track and see his robot live in the maze and save all the data in a csv file.

3.4.1 The Minotor tool

Minotor is a python tool which allows the user to monitor all the data of the robot remotely from his computer. Minotor shows the position of the robot in the maze and all the data like the speed of the robot, the robot speed and user's logs. All the data can be saved in a csv file in order to be studied later.

3.5 Communication between robots

The robot can communicate with other robots by knowing their ID.

3.6 Control external weapon

The user can control up to 3 weapons. There are two modes to control them. The SERVO which allows the user to control a servo-motor and set a position and PWM mode which allows the user to set a PWM value to an actuator like a motor. The pins to which users can connect weapons are located on the back of the robot. The pins are called M1, M2 and M3.

4 Gladiator Library - Exolegend API

4.1 Structures and Enumerations

4.1.1 Position

properties :

- **x** *(double)* : x position (m)
- **y** *(double)* : y position (m)
- **a** *(double)* : alpha angle (rad)

The position structure represents the position x, y and alpha of any object.

4.1.2 Coin

properties :

- **value** *(byte)* : Value of the coin
- **p** *(Position)* : position of the coin in the maze

This structure represents a coin in the maze.

4.1.3 GladiatorMsg

properties :

- **type** *(char)* : Message type, this field is not required by user
- **id** *(byte)* : Id of the robot to which the message will be sent
- **message** *(char[30])* : message to send

This structure represents the configuration that users have to set before sending a message to another robot.

4.1.4 RobotData

properties :

- **position** *(Position)* : position of the robot (filtered)
- **cposition** *(Position)* : position of the robot (non filtered position, received by camera without any processing)
- **speedLimit** *(double)* : speed limit of the robot
- **vl** *(double)* : speed of the left wheel
- **vr** *(double)* : speed of the right wheel
- **score** *(short)* : score of the robot
- **lifes** *(byte)* : lifes of the robot, if the this value is 0, it means that the robot is dead
- **id** *(byte)* : id of the robot
- **teamId** *(byte)* : id of the team (0 or 1)
- **macAddress** *(String)* : MAC address of the robot
- **remote** *(bool)* : If the robot is in remote mode

This structure represents all the data of a robot.

4.1.5 RobotList

properties :

- **ids** *(byte[4])* : List of robot ids playing currently in the maze.

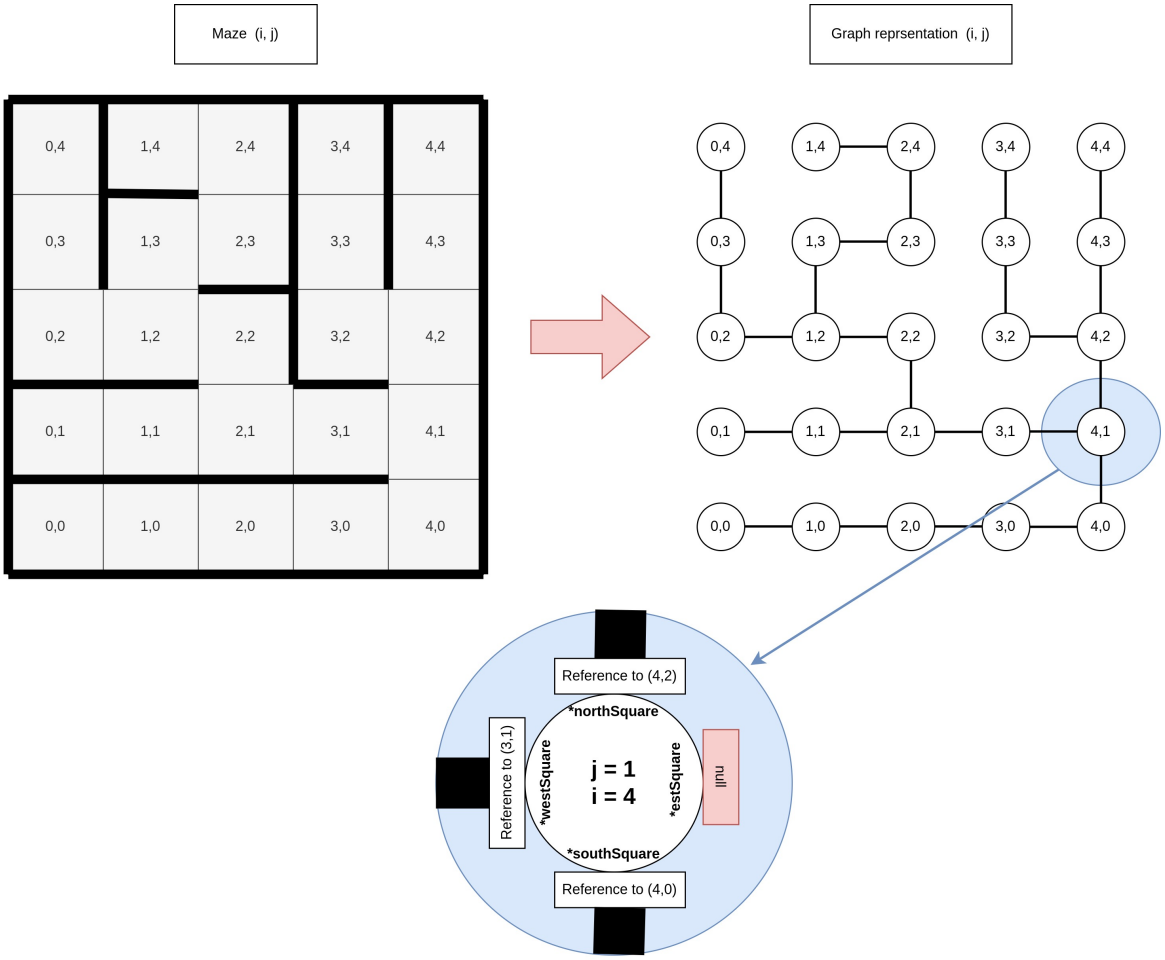
4.1.6 MazeSquare

properties :

- **i** *(byte)* : i index of the square in the maze (see image below).
- **j** *(byte)* : j index of the square in the maze (see image below).
- **northSquare** *(MazeSquare*)* : Reference to the top square. If the pointer is null, it means that there is a wall.
- **southSquare** *(MazeSquare*)* : Reference to the bottom square. If the pointer is null, it means that there is a wall.
- **westSquare** *(MazeSquare*)* : Reference to the right square. If the pointer is null, it means that there is a wall.
- **eastSquare** *(MazeSquare*)* : Reference to the left square. If the pointer is null, it means that there is a wall.
- **coin** *(Coin)* : The coin of the square. If the coin's value is 0, there is no coin in the square

This structure represent a square of the maze. The maze is represented as a graph of 14x14 squares.

If the robot can go from a square to another, those two squares will be linked, otherwise there is a wall.



4.1.7 WheelAxis

Values :

- `WheelAxis::LEFT` : left wheel
- `WheelAxis::RIGHT` : right wheel

This enumeration represents a wheel axis (left or right).

4.1.8 WeaponPin

Values :

- `WeaponPin::M1` : pin M1 of the robot
- `WeaponPin::M2` : pin M2 of the robot
- `WeaponPin::M3` : pin M3 of the robot

This enumeration represents the pin of available weapons of the robot.

4.1.9 WeaponMode

Values :

- `WeaponMode::PWM` : pwm output mode
- `WeaponMode::SERVO` : servomotor mode

This enumeration represents the pin mode to use for the weapon.

4.1.10 RemoteMode

Values :

- `RemoteMode::ON` : Control the robot with remote is enabled
- `RemoteMode::OFF` : Control the robot with remote is disabled

Remote mode to ON or OFF

4.2 Robot Control functions

4.2.1 void Gladiator::Control::setWheelSpeed()

Arguments :

- **axis** *(WheelAxis)* : Axis of the wheel for which the speed will be applied. (right or left)
- **speed** *(float)* : speed value to be set to the wheel
- **reset** *(bool) [optional]* : Reset the integrator (default false).

Set the speed of a wheel of the robot in m/s. The speed control loop is handled by the Gladiator library.. The value of speed must be in the range $[-1; 1]$. If the game has not started the speed of the robot is forced to 0.

4.2.2 double Gladiator::Control::getWheelSpeed()

Arguments :

- **axis** *(WheelAxis)* : Axis of the wheel for which speed id measured. (right or left)
- **@return** *(double)* : Speed of a wheel in m/s.

Get the speed of a wheel measured by its encoder, the returned value is in m/s.

4.2.3 void Gladiator::Control::setWheelPidCoefs()

Arguments :

- **axis** *(WheelAxis)* : Axis of the wheel for which the PID coeffcient will be applied. (right or left)
- **kp** *(float)* : proportional coefficient
- **ki** *(float)* : integral coefficient
- **kd** *(float)* : derivative coefficient

Set coefficients of the PID for each wheel (right or left). Default values are set for each coefficient in the initialization of the Library. Users are free to change those values.
By default $K_p = 1$, $K_i = 5$, $K_d = 0$

4.3 Game functions

4.3.1 bool Gladiator::Game::isStarted()

Arguments :

- **@return** *(bool)* : boolean to know if the game started

If the game has started and the robot can play, this function will return true, false otherwise. This function is supposed to be used before executing the strategy code inside a if statement.
Example :

```
1    void loop () {
2        if(gladiator->game->isStarted()) {
3         //All your strategy code goes here
4        }
5    }
```

4.3.2 bool Gladiator::Game::enableFreeMode()

Arguments :

- **enableRemote** *(RemoteMode)* : enable remote mode during free mode
- **initPosition** *(Position) [optional]* : Position of the robot to be set when free mode is enabled

This function enables the player to use the robot without connecting it to an Arena. The user's code will be executed without any restriction in speed. The user can also measure the speed of each wheel and play with a simulated arena. This function can only be called in the setup function"

For maze simulation, the estimated position of the robot is set to (0,0). Users are free to change this position by setting a new value to *initPosition* argument. The position will be estimated with encoders only. Users can use the Minotor tool to see the position of the robot in the Free mode.

If the remote mode is enabled, the user can control its robot using Minotor. The code will not be executed if the user is using *isStarted()* function.

The user has to call this function in the setup function to enable the Free mode, by default the robot is in Arena mode. If the free mode is enabled, it's impossible to change the mode during the execution of the code.

4.3.3 RobotData Gladiator::Game::getOtherRobotData()

Arguments :

- **id** *(byte)* : Id of the robot to get data from
- **@return** *(RobotData)* : Data of the robot

This function returns the data of an other robot playing in the maze. If the *id* property is not an id of a robot playing currently in the maze, the function will return an empty robot (with id 0).

4.3.4 byte Gladiator::Game::sendOtherRobotMessage()

Arguments :

- **msg** *(GladiatorMsg)* : Message to send
- **@return** *(byte)* : Status

Send a message to another robot, the function only works when robot is in Play mode. The user

can send a message every 5s. If the function returns a number higher than 0, it means that an error occured.

- if returned value is 1 : The robot is trying to send a message to itself
- if returned value is 2 : The robot is trying to send data to a robot that is not playing in the same Arena
- if returned value is 3 : The robot is not allowed to send a message, because it is not in Play phase or the delay since the previous frame is too short.

4.3.5 RobotList Gladiator::Game::getPlayingRobotsId()

Arguments :

- **@return** *(RobotList)* : List of robot ids

This function returns a list which contains all the Ids of the robots currently playing in the maze.

4.3.6 void Gladiator::Game::setPin()

Arguments :

- **@return** *(int)* : new Pin code

Set a pin code for the Gladiator. This pin code is used when the Minotor attempts to connect to the Gladiator.

4.4 Callback functions

4.4.1 void Gladiator::Game::onReset()

Arguments :

- **resetFunction** *(void*(void))* : Function to be run before a game start

Set a function to be run before a game start. A reset function is highly recommended to reset all the variables of the user code before each game.

4.4.2 void Gladiator::Game::onOtherRobotMessageReceive()

Arguments :

- **receiveFunction** *(void*(GladiatorMsg msg))* : Function to be run when value is received from another robot

The receiveFunction is called when the robot receives a new message from another robot in the maze. The msg argument contains the new message.
Example :

```
1   void MessageReceived(GladiatorMsg msg);
2   void setup () {
3
4       // ... init gladiator
5
6       //set the reset function :
7       gladiator->game->onRobotMessageReceive(&MessageReceived);
8   }
9   void MessageReceived(GladiatorMsg msg) {
10      /*This function will be called only if a
11      message has been sent by another robot*/
12      gladiator->log("Message received from " + String(msg.id) );
13  }
```

4.5 Debug functions

4.5.1 void Gladiator::log()

Arguments :

- **msg** *(String)* : String message

Logs a message as a string. Logging messages are sent to a Minotor if there is one connected to the current Gladiator.

> **Warning :** A log message is truncated to 120 chars when used with Minotor tool.

4.5.2 void Gladiator::saveUserWaypoint()

Arguments :

- **x** *(float)* : x command value
- **y** *(float)* : y command value

Track the position waypoint of the user. The user can see his command remotely from the Minotor tool (works in Free mode and Arena mode).

4.6 Robot functions

4.6.1 RobotData Gladiator::Robot::getData()

Arguments :

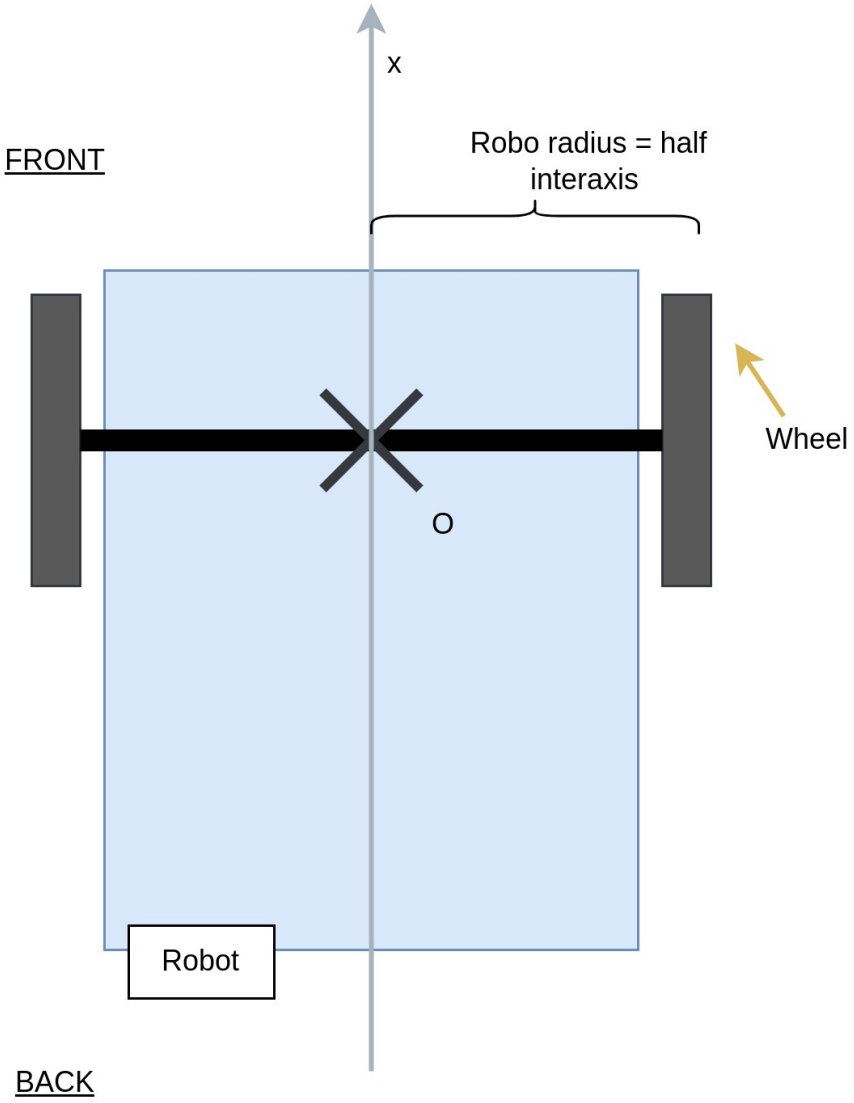
- **@return** *(RobotData)* : Get all the data sent by the gameMaster concerning the current robot

This functions returns a struct that contains all the data send by Arena (Game Master) to the current robot, notably its position and lifes number

4.6.2 const float Gladiator::Robot::getRobotRadius()

Arguments :

- **@return** *(float)* : the radius of the robot in meter



4.6.3 const float Gladiator::Robot::getWheelRadius()

Arguments :

- **@return** *(float)* : the radius of the robot's wheel in meter

4.7 Maze functions

4.7.1 MazeSquare Gladiator::Maze::getSquare()

Arguments :

- **i** *(byte)* : i index
- **j** *(byte)* : j index
- **@return** *(MazeSquare)* : returned Maze Square

This function returns the Maze Square object at the (i, j) index. If i and j are greater than 13 or lower than 0 this function will return an empty MazeSquare object.

4.7.2 MazeSquare Gladiator::Maze::getNearestSquare()

Arguments :

- **@return** *(MazeSquare)* : returned Maze Square

This function returns the maze square where the robot is.

4.7.3 const float Gladiator::Maze::getSize()

Arguments :

- **@return** *(float)* : Size of the maze in meter

4.7.4 const float Gladiator::Maze::getSquareSize()

Arguments :

- **@return** *(float)* : Size of a square of the maze in meter.

4.8 Weapon Control functions

4.8.1 void Gladiator::Weapon::initWeapon()

Arguments :

- **pin** *(WeaponPin)* : Weapon pins available on the back of the robot to be initialized (M1, M2 or M3)
- **mode** *(WeaponMode)* : Mode of the pin (PWM or SERVO)

Initialize a new weapon pin available on the back of the robot, there are 3 pins available (M1, M2 and M3). The 3 pins can be controlled with 2 different modes to control a weapon : SERVO or PWM. This function must be called only once per pin. If it's called several times, only the first mode set will be taken.

4.8.2 void Gladiator::Weapon::setTarget()

Arguments :

- **pin** *(WeaponPin)* : weapon pins available on the back of the robot (M1, M2 or M3)
- **value** *(float)* : value in range [0; 255] to set for the weapon

If the weapon mode is set to SERVO, the value is the position of the servo in degree to set. If the weapon mode is set to PWM. the value is the dutyCycle of the Pwm. **Example :**

```
1
2     void setup () {
3
4         // ... init gladiator
5
6         //set M1 as SERVO
7         gladiator->weapon->initWeapon(WeaponPin::M1, WeaponMode::SERVO);
8         //set M2 as PWM
9         gladiator->weapon->initWeapon(WeaponPin::M2, WeaponMode::PWM);
10    }
11    void loop() {
12        if(gladiator->game->isStarted()) {
13
14          //set the servomotor to 95
15          gladiator->weapon->setTarget(WeaponPin::M1, 95);
16
17          //create a pwm signal with a duty cycle of 0.5 on pin M2 (alpha = 0.5)
18          gladiator->weapon->setTarget(WeaponPin::M1, 128);
19
20          delay(1000);
21
22          //create a pwm signal with a duty cycle of 1 on pin M2 (alpha = 1)
23          gladiator->weapon->setTarget(WeaponPin::M1, 255);
24        }
25
26    }
```

EXOTEC

This document has been digitally signed by

Signatory	Signature
Valentin De Bruyne	Valentin De Bruyne Fri, 31 Mar 2023 06:42:06 GMT