EXOTEC

Gladiator features and API

Table des matières

1 Version history

1.1 Version v1.0

— Initial version : Gladiator version 1.5.1

1.2 Version v1.2

— Gladiator version 1.5.2
— 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
— Rename WithoutArena Mode in Free Mode
— 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 microcontroler. The GLadiator LIbrary is written in C++ in aim to be used Arduino IDE.

2.1 Definitions

— **Arena** : The field where robots play

— **Arena Screen** : Control screen of the Arena. It's the small touch screen of the Arena. It's able to start a new game and identify new players.

— **Game Master** : The software which control the whole game

— **Gladiator** : The robot used by players.

— **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.

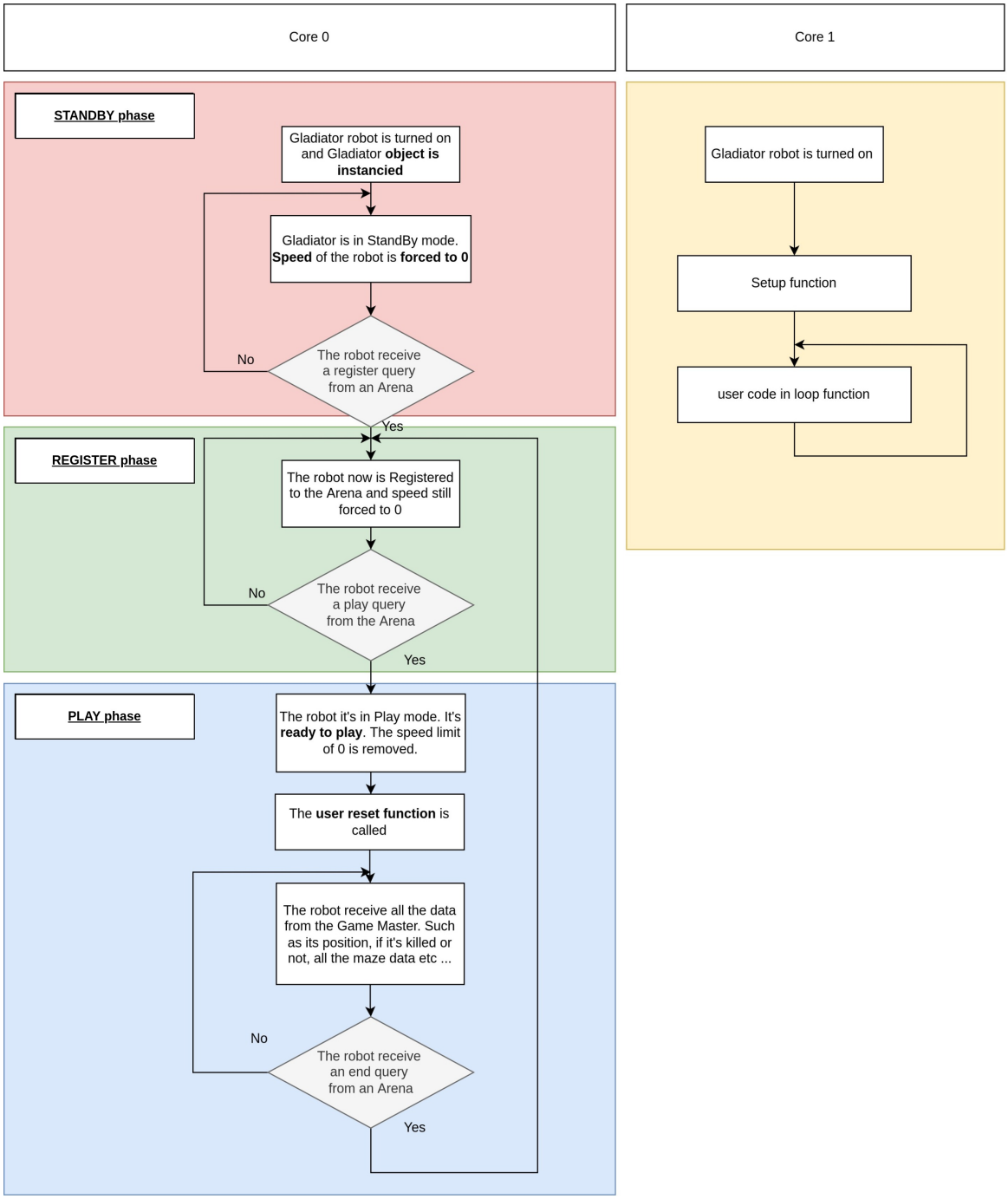
2.2 Gladiator Library

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

2.2.1 Code execution

When you include the Gladiator library and instanciate a gladiator object, a thread start running in the second core of the micro-controller. This thread is running in the background of the user code and it communicates with Arena, it also serve the speed of the robot and filter its position. The code of the user is executed on the first core.

2.2.2 Gladiator life cycle



2.3 How to start ?

This section provides a basic code template wich 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->isReady()) {
        // Write your strategy code here
    }
}
```

2.4 Two modes of controlling the robot

There is two modes available to controlling the robot :

— **Arena mode** This is the default mode of the Gladiator. The user code is automatically executed after registering and stating a new game.

— **Free mode** The user code is executed even if the robot is not registered to achieve some test 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 is already asserved by the Gladiator Library. User can change the parameters of the PID of each wheel.

3.2 Get robot data

User can get the data of the robot such as it's 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 late in a range of [20ms, 100ms].

3.3 Get maze data

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 in live its robot in the maze and save all the data in a csv file.

3.4.1 The Magnifiyk tool

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

User can control up to 3 weapons. There two modes of controlling 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.

4 Gladiator Library - Exolegend API

4.1 Structures and Enumerations

4.1.1 Position

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

The position structure represent 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 on the maze

This structure represent a coin of the maze. If the *value* property is 0, the coin doesn't exists.

4.1.3 GladiatorMsg

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

This structure represents the configuration that user have to set before sending a message to an other robot.

4.1.4 RobotData

properties :
- **position** *(Position)* : position of the robot (filtered)
- **rawPosition** *(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

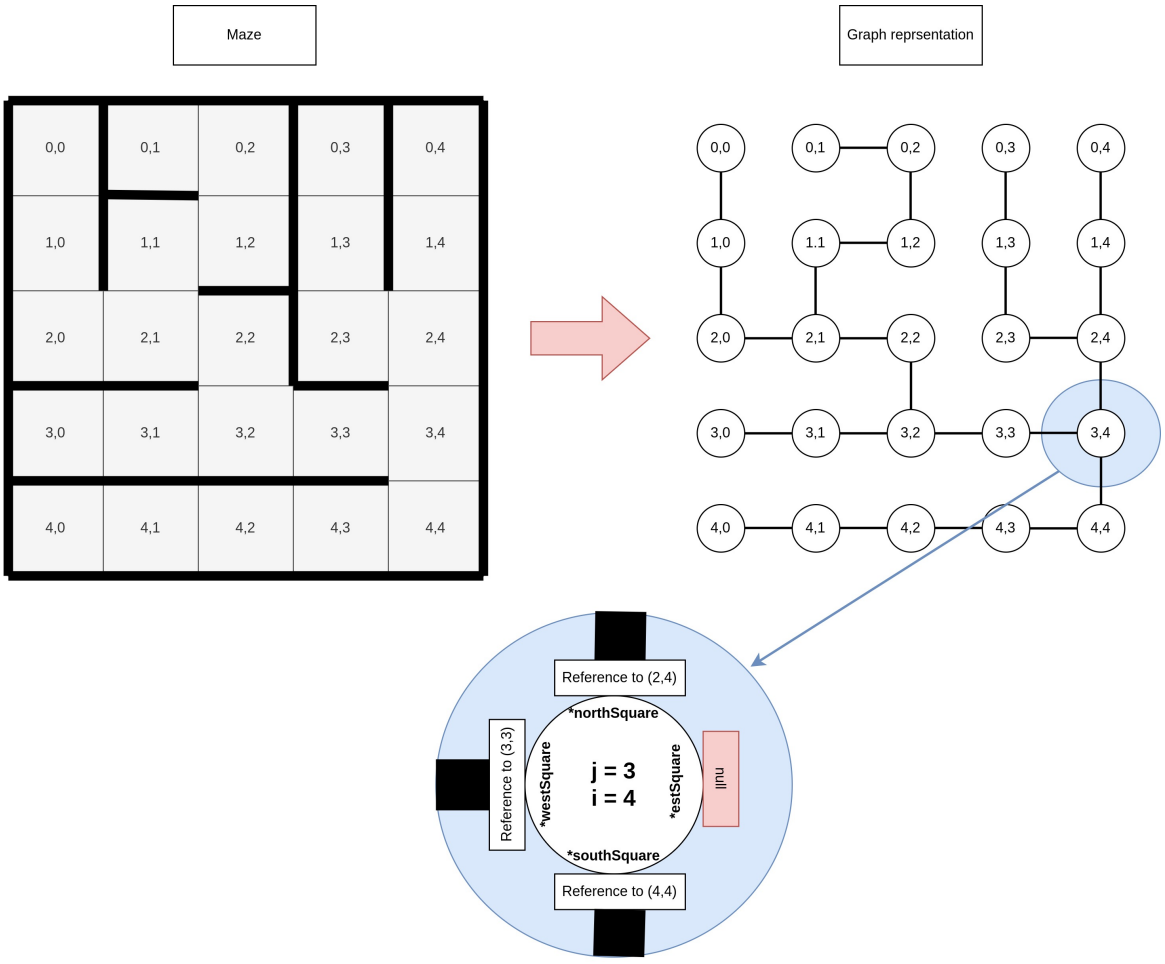
This tructure represents all the data of a robot.

4.1.5 MazeSquare

properties :

- **i** *(byte)* : i index of the square in the maze.
- **j** *(byte)* : j index of the square in the maze.
- **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.
- **estSquare** *(MazeSquare*)* : Reference to the left square. If the pointer is null, it means that there is a wall.
- **coin** *(Coin*)* : Reference to the coin of the square. If this reference is null, there is no coin in the square

This structure represent a square of the maze. The maze is represented as a graph. If the robot can go from a square to another, those two squares will be linked, otherwise there is wall.



4.1.6 WheelAxis

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

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

4.1.7 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.

4.1.8 WeaponMode

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

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

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 is asserved by the Gladiator Library. The value of speed must be in the range $[-1; 1]$.

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.

4.3 Game functions

4.3.1 bool Gladiator::Game::isReady()

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

This function returns a boolean. If the game has started and the robot can play, this function will return true, false otherwise. This function is highly recommended to use before executing the strategy code inside a if statement.
Example :

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

4.3.2 bool Gladiator::Game::enableFreeMode()

Arguments :
— **enableRemote** *(bool)* : enable remote mode is enables
— **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.

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 *Position* argument. The position will be estimated with encoders only. Users can use the Magnifyk tool to see the position of the robot in the Free mode.

If the remote mode is enabled, the user can control it's robot using the remote app, but the code will not be executed if the user is using *isReady()* function.

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::sendRobotMessage()

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 occurs.

— 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 mode or the time before send a new frame is not reached.

4.4 Callback functions

4.4.1 void Gladiator::Game::onReset()

Arguments :
— **resetFunction** *(Function())* : 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::onRobotMessageReceive()

Arguments :
— **receiveFunction** *(Function(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 receive(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

Log a string message.

Warning : A log message is truncated to 120 chars when used with Magifyk tool.

4.5.2 void Gladiator::saveUserConsign()

Arguments :
— **x** *(float)* : x command value
— **y** *(float)* : y command value

Track the position command of the user. The user can see his command and Magnifyk tool remotly (work 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 sent by Arena (Game Master) to the current robot such as it's position (both filtered and not), lifes number, speed limit etc ...)

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.

4.7.2 MazeSquare Gladiator::Maze::getNearestSquare()

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

This function returns the nearest maze square of the robot.

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

Initialize a new weapon pin available on the back of the robot, there is 3 pins available (M1, M2 and M3). The 3 pins can be controlled with 2 different modes to control a weapon : SERVO or PWM

4.8.2 void Gladiator::Weapon::setWeaponPower()

Arguments :
- **pin** *(WeaponPin)* : weapon pins available on the back of the robot (M1, M2 or M3)
- **power** *(float)* : power to set to the weapon between 0 and 1

Set power to a weapon, if it was initialized in PWM mode.

4.8.3 byte Gladiator::Weapon::setWeaponPosition()

Arguments :
- **pin** *(WeaponPin)* : weapon pins available on the back of the robot (M1, M2 or M3)
- **position** *(byte)* : integer in range [0, 180] : ne position to set to a Servomotor used as a weapon

Set a position of a weapon, if it was initialized in SERVO mode.