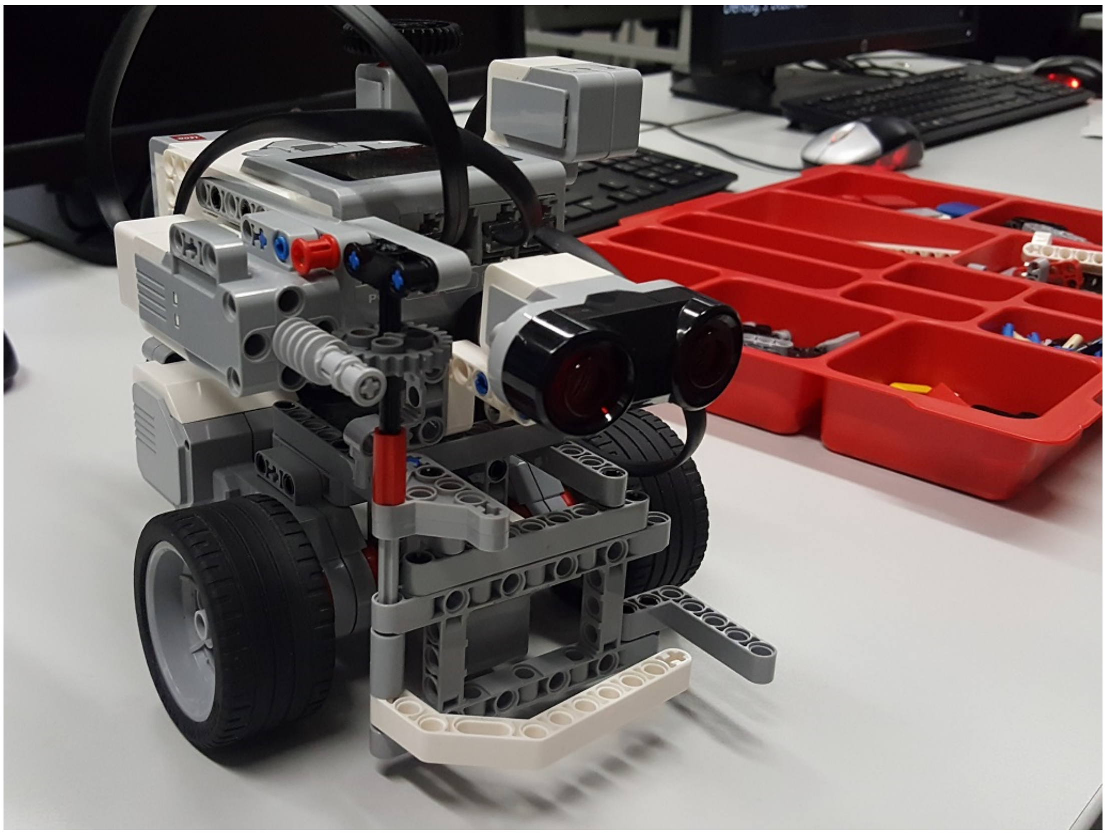
TEC Reutlingen University

Computer Science

Project Documentation



Guillaume COUZINET

Elias NEUMAIER

Group #7

Professor: Dr. T. TAEFI

IPEB3 - Computer Science

WS 2019 - 2020

Abstract

Within in the context of the lecture ”Computer Science” by Pr. TAEFI at Reutlingen University, our task was to program a robot which is able to play a modified version of ”Giana Sisters”. The main goal was an autonomous functioning of the robot. Moreover, it was supposed to master the game at the end of the semester with as few mistakes as possible. The project was approached within teams of two to three students.

This documentation summaries the whole project including technical and methodological details as well as problems that occurred during the project period. Main issues arose in the project’s programming part which led to several changes in our strategy to solve the given challenge. Unfortunately, we could not fulfill all expected requirements due to technical issues. However, we were still able to program an autonomously functioning robot that could participate in the final game.

A special focus in this documentation was laid upon the usage of Scrum as a project management technique which was another essential part of the Computer Science lecture.

Cover page photo: Couzinet, G. (2019). Robot Flying Wall-e [Photo].

Follow the link to get the online project:

https://github.com/GuillaumeCou/ReHo1920_ComputerScience

Contents

List of Figures

1 Introduction

Within the lecture Computer Science in the third semester of the study program International Project Engineering, offered by Reutlingen University, the given task was to program an autonomous robot that is able to play a modified version of ”The Great Giana Sisters”. Giana Sisters is a 1987 published game, similar to Super Mario Bros., in which ”Gianna” has a nightmare where she travels through 32 stages, collecting diamonds, avoiding monsters and looking for her sister Maria who then awakens Giana to finish the nightmare and also the game.
In our hardware-related version of the game with the ”LEGO Mindstorms EV3 Programmable Robot”, the rules and conditions were respectively adapted:
As soon as the button of the robot was pushed, the robot was supposed to drive around in a square, which was the playing field. In this filed there were circles in three different colours (red, blue and yellow) taped on the ground. In the middle of these circles were little wooden cubes in the appropriate colour, that denoted the diamonds. Each of these diamonds was worth five game points and had to be collected, but these game points were only counted if the diamonds were in the cave (will be explained soon) with the robot by the end of the game. Indeed, the robot was not allowed to somehow find a diamond and directly grab him. The robot first had to surround the diamond respectively had to surround the coloured circle in which the diamond was. As soon as the circle was surrounded, the robot was, if there was no other robot (will be explained soon) inside the circle, allowed to enter the circle and to pick up the diamond. This pick-up-process including the circuit of the circle had to pass in 30 seconds and a timer that counts from 30 to zero should be shown om the screen of the robot. If any of these rules such as surrounding the field, leaving the field and not entering the circle if there was another robot inside was violated, the robot would lose one of his overall three lives.
Furthermore, there were three to six cartons like milk cartons as obstacles stationed in the field. These obstacles must not be moved, not even touched by the robot. In case that the robot touches or even moves an obstacle, the robot would lose another life. Additionally, these obstacles must be counted by the robot, but none of the obstacles must be counted twice. The counted number of obstacles should be repeated loud by the robot every five

seconds but should be paused when the robot enters a ring to fetch a diamond.
The game ends after five minutes. By the end of the game the robot must be in the cave, which is a white piece of paper, taped to the ground. As soon as the robot is in the cave, the number of counted obstacles still must be repeated every five seconds until the button is pressed again to finish the game.
What makes the challenge even more demanding, is a second robot, which must play the same game, at the same time in the same playing field. This robot does not only collect diamonds you then cannot collect anymore or might go into a cave you then cannot go into anymore, the robot is also like an always moving obstacle that must be surrounded and by no means be crashed by the own robot.
Especially for the "avoiding other robots"-part a common strategy, followed by the whole course was required and absolutely necessary.

2 Technical and other preconditions

Challenge area

The game was played in a square, that measures four times four metres. It was framed with green-coloured tape with a width of approximately five centimetres on the floor. The floor itself was grey-black-white mottled. The circles, one of each colour, around the diamonds were also taped on the floor. The little wooden cubes that denoted the diamonds and were in these circles had a side length of approximately 2.5 centimetres. The milk cartons, representing the obstacles were cuboid, had a base that measured approximately six times six centimetres and had a height about approximately twenty centimetres. The cave where the robot should be in by the end of the game was a white sheet of paper, taped on the floor. that measured twenty times twenty centimetres.
All of these elements of the game were positioned in the playing field with at least 40 centimetres of space around each element where no other element could be positioned.
The explained playing field was constructed in the entrance hall of building four at Reutlingen University. Some extra problems that came up with this specific place where the challenge was about to take place were that the floor is, as already mentioned, grey-black-white mottled. Due to that, the colour sensor that was pointed onto the floor, delivered various colour-values when he was moved in little circles above the ground. Furthermore, the green tape that denoted the border of the playing field was measured as black by the colour-sensor under certain light-conditions.
These specific playing-field issues were also preconditions and, in this case, even problems we somehow had to deal with.

The robot

The given robot that should finally be able to accomplish this challenge, was a "LEGO Mindstorms EV3 Programmable Robot".
It is about an assembly kit from LEGO with which you can assemble different types of robots that are able to perform in different situations or in different environments. In the assembly kit there were several components at one's disposal in order to build a good

robot that was adjusted to the explained task. One of these components was the so called "brick". It was like the heart of the robot, capable to work on through the installed code and to manage the execution of all the commands in the code. On this device all the written code was installed, called and executed.
In the brick were several in- and outputs. These ports connected the brick, which was like the centre of the robot, with all the other components such as big and small motors and several sensors.
The two bigger motors and the one smaller motor were driven through the outputs of the brick where you could command to turn the motors about a certain angle or a certain time with a given speed.
The sensors delivered values to the brick. Available sensors were two push-sensors that were like buttons you can push to start or stop something, a colour-sensor that was able to detect different colours and an ultrasonic-sensor, that was able to measure the distance to obstacles. These sensor are not very accurate. For example, the gyroscope has more or less 3 degree tolerance for a measurement. Sometimes, it just don't give the right value.
Furthermore, there were of course multiple LEGO-parts given to mount all these sensors and motors together to a robot. In the beginning we had to mount a robot after a given instruction, so we had a robot to do some tests and to work on.

Program language and project tools

The coding language that was taught and used to program the robot was Python. In advance to the robot-challenge we worked on a few Jupyter-Notebooks, in order to train our Python-skills. Out of that we learned the basics of Python such as handling strings, numbers and tuples, writing selections and functions, using loops and of course combining all these contents.
For the robot system, we uses the `ev3dev-jessie` version of `ev3dev`. It use python3 and few function are already programmed to interacted and use the blocks of the robots.
For developing such a complex product, an organizational framework was needed that directed the way how the development process regarding the task should look like. Since it is meanwhile common to do that in software development projects, in our case scrum was chosen as well. The way how scrum was applied will be quoted in the following.

As usual in scrum all the requirements given by the customer were written down as user stories on little “cards” in Trello and put into the product backlog. For our group the program “Trello” was chosen as scrum board. In Trello we created several lists like the columns on the scrum board. So, there was one called “Product backlog”, the next was “To sprint”, then “On the run”, furthermore “To check” and the last one was “Done”. By these stated means the challenge was finally approached.

3 Analysis: Plan how to tackle the problem

In order to address the challenge in a possibly efficient way we spend the first week on incorporating in the challenge and thinking about all the given requirements. We tried to figure out what problems each requirement contained and what dimensions each requirement had.
The plan was now, to cut the whole task and to formulate user-stories that contained only a small part of the whole challenge. These smaller parts seemed to be easier to finish when we only focused on them instead of the whole challenge. Later we then could merge all the smaller codes together to the final code.
Another reason for cutting down the challenge into smaller separate tasks was that now everybody can work on parts of the code for himself, we do not always have to meet for code-writing and more people can work on different parts at the same time what leads to an increase of our overall group-productivity.
These smaller tasks were written as user stories in our product backlog, so the first version of the product backlog was created.
Some examples of our user stories in the backlog read as follows:

- As robot I want to stay in the field and do not cross the borderline ;
- As robot I do not want to crash another robot ;
- As robot I want to find and keep the diamonds, one of each color ;
- As robot I do not want to touch or move the obstacles

After writing down the user stories, we classified them in the categories "Field", "Bonus points – diamonds" and "Obstacle and the cave". Due to the classification it should later be easier to determine in which category which progress has been done and to know where more effort is needed. So, it was like cutting down the challenge (what we already did in order to formulate the user stories) but not as extensive as at the first time.
Additionally to the classification we did the probably even more important prioritization of the user stories. In the prioritization all the above written user stories were again classified in three levels how important the stories were. The level low implied that the

story of course is a requirement of the customer but there were no consequences if this requirement is not fulfilled. The middle level meant that these requirements brought extra points which were good and needed for the ranking, so these requirements were more important than the low ones. Finally, there were the requirements that were ranked in the highest level. This meant that if these requirements were not fulfilled, the robot would lose one of the three lives and is at risk of being taken out of the challenge and ending up with no points.

Therefore the high prioritized user stories were the ones we definitely were about to focus on.

The whole backlog including the classification of the user stories and the prioritization can be seen in the figure 3 at page 25 in annex A.

4 Design and implementation

Hardware adjustment

At the very beginning of the challenge we thought about the changes and modifications we had to do in order to adapt the standard robot we had so far to the special conditions of the challenge.
At the standard robot the ultrasonic sensor was at the top of the robot, approximately fifteen centimetres above the floor. Because of our plan to use the gyro to detect obstacles and other robots, that was principally good for us so there was no need for change.
The gyro sensor and the button were on top of the robot, which was also okay for us. The colour sensor was attached to one of the grabbing arms and pointed into horizontal direction. Thus, one could measure the colour of things you want to grab but not the colour of the floor. Since we wanted to measure the floor colour with the colour sensor to detect the border line of the field and the circles of the diamonds, we remounted the colour sensor directly on the body of the robot, pointing on the ground.
Furthermore we considered the width of the robot as potential problem. Because the robot will e.g. have to surround diamonds or obstacles and should in the meantime not touch them it would be the best if the robot as narrow as possible.
In order to make the robot narrower we completely dis- and remounted the robot.
A picture of the new constructed robot can be seen on the cover page.

Project setup and general strategy

4.1 Sprint 1

Sprint planning

The purpose of this sprint was to launch us into the development of our program. We laid the foundations of our methodology and organization. We had to carry out the tasks independently in order to be able to test them individually. For this, we wanted to develop them in independent and unitary files. We had to carry out unit tests for our *DODs* and write all the documentation.

During this sprint, we wanted to develop the functionalities allowing the robot to avoid obstacles (and robots) and to move on the surface. We also wanted the robot to be able to differentiate a simple obstacle from a robot. We also needed to think about and start developing the function that would allow the robot to find, pick up and store diamonds.

Development and implementation

Noted that the basic functions for simple movement had already been written and tested. We are talking here about the functions of rotation with gyroscope, driving in a straight line over a defined distance and on/off of the motors. The distance management and the color recognition still had a rather random operation and still deserved some development time. In order to avoid obstacles, we opted for a bypass. An ultrasonic sensor on the front of the robot was used to determine the distance between the robot and any obstacle in front of it. When this distance reached a value that was arbitrarily too small, the robot had to stop and start the bypass procedure. To do this, the robot would execute a half-square of a few tens of centimeters on each side.
To differentiate a robot from a simple obstacle, we had opted for a comparison of values. When the robot spotted an obstacle, it had to stop and move back a few centimeters. Once completely stopped, it would take about ten distance measurements. If the difference between these values was too large (we had defined a tolerance threshold), then the obstacle had potentially moved. It was then considered as a robot and not counted. Otherwise, we could consider it as a simple obstacle and therefore incremented our obstacle counter.
For the moving strategy, we were looking for the best way to rake the largest possible area. Initially, we were thinking of going back and forth along the length, shifting at each end. This is a fairly simple maneuver. It is activated by the detection of the colored band indicating the surface limit. The robot then performs a 90 degrees rotation, travels about ten centimeters and then a second 90 degrees rotation. It can then continue its course and search for diamonds.
We had tried to develop the functions that would allow the diamonds to be retrieved and kept. However, we had to abandon this part due to lack of time.

Sprint review

This sprint was really tough. We had calculated an average speed per sprint of 27 Story Points. Here we had planned to complete tasks with a total of 29 Story Points. Due to a lack of time, and with later reflection, we finally only reached 21 points for almost completed tasks (missing tests). In the end only 13 task points were actually completed. We left out the Diamond Retrieval task because many technical problems arose and we had to review our move strategy. The gyroscope being absolutely unreliable, we had to think of another method of rotation.
Our robot was officially able to avoid obstacles and robots. It could also differentiate between the two and therefore count the obstacles. Its movements on the surface were still very random and did not match our vision. There were a lot of factors to deviate the robot from its stroke or rotation. For example: differences in the ground between the two wheels, a random power difference between the two motors, a loss of grip for one wheel and a few other factors.
During this sprint review, we realized that our backog did not take into account the difficulties we were encountering and the technical problems. Indeed, having no knowledge about the reliability of the equipment and sensors, we had produced a very optimistic first backlog. We had also not foreseen the personal impediments slowing down the development. At that time, the functionalities were developed in independent files, thus avoiding the risks due to problems related to the other functions.

4.2 Sprint 2

Sprint planning

After the first sprint, we noticed that the way we wrote the given requirements as user stories was in some cases still to general or still a too big subtask we could not work on in an efficient way if we left it the way it was. Consequently, we had to revise the backlog and cut the already given user stories into even smaller subparts of our user stories.
For example, there was the user story no. 2 that was "As robot I do not want to crash another robot". It was a user story that implied to always measure the distance, to determine whether a detected object is a robot or an obstacle in order to count the

obstacles but not count the robots and to avoid the object. Since it was difficult to translate the above written user story into a code, we rewrote that user story into these three sub-parts.
With this proviso we rethought the whole backlog and cut the stories down into smaller parts, if necessary.
The updated version of the backlog can also be seen in the figure 3 at the page 25 in the annex A .
In the meantime, we also noticed that sub-parts of some user stories were the same as sub-parts of other user stories. Therefore, we did not have to write that many codes anymore, because we of course did not write the same thing twice.
Supplementary to our backlog redefinition we also canceled one element out the backlog. It was originally our strategy to "Always know where you are in the field" in order to find the cave easier in the later state of the challenge. Certainly, we even had a strategy how to do that, but the sensors were not accurate enough to contribute reliable values we needed.
Consequently, we estimated this as not doable and decided to change this strategy and delete it from the product backlog.
Then, with the new backlog we started to divide up the tasks and worked on them.
So we also had to review some of our functions to make sure that the changes were properly taken into account. We also developed new functions: ring detection, ring bypassing, aiming at the centre of the rings. The last objective was to make up for the delay accumulated in the previous sprint. So we had to do a lot of tests.

Development and implementation

In redefining our backlog, it appeared that we had performed most of the functions necessary for the challenge. This made this sprint lighter and allowed us to aim for a better code quality.
Bypassing the rings containing the diamonds was a real challenge. In order to simplify our program, we chose that the robot would always bypass the rings in a counter-clockwise direction. Initially, we chose to modulate the speed of the motors by dichotomy. Before starting the bypass, we had to place the motor on a tangent of the ring. To do this,

we would start the right engine in reverse and it would only be stopped when the colour sensor detected the colour of the ground. Then we would start the left engine at a constant speed, always lower than the speed of the right engine. The speed of the right engine will be modulated by the dichotomy between the minimum speed of the left engine and its starting speed. The choice would be made around the color returned by the sensor. Thus, if the color corresponds to the ring, the engine will be slowed down. If the color is the color of the ground, the engine will be accelerated. This way, the robot will follow the contour of the ring.
We had also developed the function to retrieve a diamond and keep it. Initially, we wanted to deposit the collected diamonds in a cave of our choice. This meant knowing our position. The solution was to make a list of all the robot's movements and another of the positions of the elements encountered. To go back to their location, we just had to take the list of movements and make them upside down. This idea was finally abandoned because the sensors and the movements of the robot were not at all precise. So we made the decision, in order not to lose our points, to walk the diamonds with the robot.
After walking around the ring, the robot had to position itself towards the center where the diamond was. The robot would stop after 7s and rotate 90 degrees to the left to aim at the center. Once in position, all you had to do was move back a little, open the arm, move forward, close the arm and then resume the exploration. We encountered some difficulties in calibrating the opening and closing of the arm because the motor speed fluctuated according to the robot's battery charge.

Sprint review

By the end of the second sprint we reviewed what and how we worked. The "What we worked" focused on the progress in programming and the "How we worked" was the question that should be thought about in the retrospective.
The first thing to mention here is that, due to the cancellation of one task in the backlog, we had a decrease in our overall effort we had to bring. Thus, the effort we brought in the first sprint was enough to stick to the now required velocity and we were not in debt anymore. Since we also brought a quite good effort in this week, we were up to date in our plan.

The burn-down chart of the new backlog after the 2nd sprint can be seen in the figure 1 at page 24 in annex A.
In the retrospective we were very happy about the "What". It was a sprint in which a lot of the coding has been done so we had a good basis to work on and to perfect the code and could stick to our velocity.
The "How" was unfortunately not that satisfying. We wanted to meet more often because we noticed that more meetings result in a noticeable better efficiency in the code writing, but however did not manage to implement that plan.
That was kind of disappointing because we could have performed better but at least we knew which measures to take in order to improve our processes and code-writing in the following weeks.
We had almost all the functions necessary for the challenge. So far we had a lot of independent files. This was a problem for us, because setting up a naming convention would have been counterproductive in relation to the size of our project. Also, in our new features we were using our older, more basic features. But this made it very difficult to understand the programs and updates. To simplify our work, we decided to work in one single code file with all our functions.

4.3 Sprint 3

Sprint planning

Among the main functions we had selected, the only ones missing were those that would allow the robot to get to a cave before the end of the allotted time and the one that would allow us to give the number of obstacles encountered. We also wanted to optimize our already written functions such as the one allowing us to pick up diamonds.
We also had to start implementing all these functions together in one program file. So we had to spend some time cleaning up and harmonizing the code so that it would all be compatible. Finally, we had to start the final test phase.

Development and implementation

In order to get the robot into a cave before the end of time, we decided to give up the search for diamonds after a certain period of time. Once this time had elapsed, the robot would no longer react to the coloured rings but only to the (white) cave areas. This function was implemented in the heart of our main loop. Thus a boolean flag indicated if the defined time had elapsed. If yes, this boolean could neutralize the diamond search function and unlock the cave search function. Once a cave was found, the robot would enter it and would be immobilized until the end of the challenge.
For the obstacle count, we were faced with a problem. Indeed, if the count was wrong, we would lose hard-earned points. We thought of a way not to count the same obstacle twice, without success. We decided to abandon this feature. Indeed, if no account is made, no points are gained, but no points are lost! You could find the former program in annex D on page 42
We also optimized and refactored our functions in order to increase their reliability and accuracy. For example, rotations will be performed at the minimum speed of the motors.

Sprint review

In principle, we had written down all the functions we wanted to have for the challenge. Only, due to lack of time, we hadn't been able to implement them all together and we hadn't always been able to test their efficiency on all possible situations.
So all our functions were ready but not our final program!
We still had a little time left before the challenge, enough time to solve as many problems as possible.

4.4 Latest concessions and changes

As said before, we had all our functions, but we hadn't tested them all together to make the robot work under the conditions of the challenge. So here are the last changes made in the program between sprint 3 and the challenge.
Due to feedback from other groups, the blue ring was difficult to detect by the robots. So we decided to do without this diamond and focus on the two others: red and yellow.

Indeed, the robot would have wasted time trying to bypass or detect the blue ring. The sensor regularly confuses the blue colour of the ring with the black of the ground.
We have also greatly simplified the obstacle avoidance function. Rather than wasting time going around it, remember that the sensors are not very accurate, we decided to simply turn around at an angle greater than 90 degrees randomly and continue in a random direction as well. So here again we decided that the robot would drive in random directions on the surface. This removes the problems of accuracy. You could find this former program in annex D on page 44.
For the ring bypass, we also chose to simplify. We found that the dichotomy method was not as reliable as it was complex. We then gave a low speed to the left motor and a higher speed to the right. We calculated the time it took the robot to go around and we inserted it in a `sleep()` function. So the robot will have 7 seconds to complete the tour of the ring and then the motors will stop. The program continues on the recovery of the diamond. You could find this former program in annex D on page 43.

5 Discussions and Reflections

5.1 Results of the sprint retrospectives

The sprint retrospectives were a for us very important item ("Punkt" auf der Tagesordnung), because in this stage of the whole development process, we only thought about how we were working and not about the coding of the robot itself. We were about to investigate, which ways of working were the most efficient in the past week in order to desist from the inefficient ways of working.
The topic that came up probably the most times and was discussed the most was about our direct collaboration. What is not meant with this is, that we somehow tried to accomplish the challenge by ourselves. It is rather that, due to full schedule books, we could only barely manage to find dates were both of us had time, although we were only a team of two people.
Thus, we were like forced to work on the challenge separately, focusing on ones tasks and trying to accomplish them till the next meeting. We then of course had a few meetings to discuss the latest improvements and to work on them.
We noticed that we were quite efficient in that sessions where we worked together on the same thing, but nevertheless we did not manage to have such sessions more often.
This "lack of meetings" is maybe one of the main reasons, why we finally could not meet all the requirements of the challenge .
Due to this way of working on the challenge, we needed special tools in order to organize us in an appropriate way to still stay connected and updated about the done coding progresses. A software we chose for that purpose was GitHub which is a platform adapted to software development and therefore the right tool for us.
The problem that came up with this was that one of us was not that familiar with GitHub, so there was a lot time spend on understanding how to use the program and trying to organize it as understandable as possible. Hence there was also a lot of time spend in which we were not coding and within having a lower efficiency.

5.2 Discussion of our solution

Our solution has the advantage of being modular. Indeed, we have a loop that runs permanently and in a few thousandths of a second as long as the time condition is not reached. Thus, we can perform verification functions such as distance quickly and without consequence. The loop pauses so that the robot can perform more complex tasks. You could find the *Activity Diagram* on figure 4 in annex B on page 26.
The advantage of independent functions is that we can modify them without affecting the proper functioning of the other functions. Moreover, they are easily replaceable and allow for better code readability.
To make our robot functional, we had to draw a line under several features that could not be implemented because they were too complex or required too much precision.
We quickly realized that we couldn't get the robot to do specific tasks. So we decided to play with this inaccuracy by adding randomness. This greatly simplifies our code and is much better suited when there are many different situations where the resolution does not require a high degree of precision. This is particularly the case for bypassing, and U-turns.
Still to overcome this problem of accuracy and knowing that the challenge would only be bonus points, we preferred to focus on tasks where we were sure we could gain points or at least not lose any.

5.3 The suitability of Scrum for this task

Scrum is typically used as framework of development processes where a high agility is of advantage and the requirements and maybe also approaches can change more often. Especially the agility and thus the possibility to change things in the process is a characteristic in Scrum-organized development processes.
This is a property of Scrum that was during the development of the robot also essential and facilitated the sometimes-necessary alteration of our approaches.
When we tried to approach this challenge that was quite demanding for us rather unexperienced students, we made a Project Setup in which we planed how we wanted to address the task and how we wanted to proceed for the first week.

It was a plan that pointed out on which parts of the challenge and which subtask we wanted to focus in this week. Therefore, we also already thought about a strategy and possibilities how to complete the task in order to make the plan.
After beginning to work and trying to incorporate our thoughts and strategies into a code, we quickly noticed that some of the subtasks we created in our plan respectively in our "To Sprint" list on our scrum board were partly not doable in the way we defined them. Thus, these subtasks had to be rethought and anew defined. Sometimes they were that complicate and could not be cut down anymore into easier parts, so we decided to change the requirements on our own and to cross some of them completely out.
This implies that the requirements towards the robot and within the backlog was continuously changing.
To be able to deal with these changes of the requirements in a proper way an agile framework such as Scrum was perfectly suitable.
Furthermore, Scrum is a framework in which you stagnate the development process for a short time in order to evaluate our working performance of the past week, called the Retrospective.
This was also very efficient for the quality improvement of our procedure. Since the code-writing was a given task in which we were not that experienced, the weekly retrospectives helped us or even forced us to figure out how we performed the best and of course we build upon these good ways of working. A result of that was, that we continuously improved our approaches and over time were able to address the tasks in a much more efficient way.
Summarizing one can say that Scrum is a very easy and efficient framework for such a software development process. Due to the usual elements such as the just mentioned Retrospective there was also a steady and quick process-quality improvement and we could learn a lot in this rather short time.

5.4 The project as didactical method

The whole project can be evaluated as didactical method for several things.
At first, the didactical method used to teach how to program can be discussed. Ahead one must consider that as it came to the project phase of the lecture, we already had some basic knowledge and knew the principals of Python.

Then, in the following project and challenge phase we applied and extended our skills by researching solutions for our problems and sometimes asking our professor for help.
The first part of this, applying the already learned content and working with it was a very good didactical method. Thereby we consolidated our knowledge and found ways to program things as smart as possible.
So for the project as didactical method was very good to strengthen already existing skills.
The second part of this, researching in case you do not know what to do was sometimes very exhausting and frustrating because although there are probably all the necessary code parts somewhere in the internet, it was hard to find ways how to implement some ideas and therefore costed a lot of time.
But still, there was always the possibility to ask the professor in the lectures. So, it would have been nice to get some more input on some topics, but you were never lost or left alone.
The second part, the whole project can be divided into and where we were taught about some topics was the Scrum part.
In the first section of the lecture (not the project phase) there was some input given about Scrum. We learned the basic concept, the roles that are taken in a scrum team and the typical elements of a with scrum organized project. Of course, we thereby had a little notion of what scrum could be, but what it really was, was not that clear to us.
After that it came to the project phase in which we were supposed to organize our project in the scrum framework. Thus, we applied our knowledge about scrum and managed our project in this agile way, had sprint plannings and retrospectives, redefined the backlog, if necessary, and always improved our processes.
Applying scrum was a as we perceived very efficient way of learning the sequences and getting used to this way of managing a project. Therefore, this can in any case be experienced as an effective didactical method.
Finally, one can also discuss the project as didactical method for imparting of soft skills such as overcoming frustration, staying motivated, staying focused and not losing the orientation to work towards the bigger goal.
Summarizing one can say that the whole project is a very good didactical method. Compared to a classic lecture, where you just sit there and only listen to the professor, you

rather immediately applied what you just learned and, in this way, consolidated your knowledge. Thereby you also made your first experiences in a programming project, which is very good and something you can build upon.
But besides the progress we made in code-writing there was, as already mentioned, the progress and first experiences with scrum and, what is probably the most important to us, the personal progress. Especially the personal progress we made in dealing with problems in projects will very likely push us further in following projects.

A Burn-down chart and Backlogs

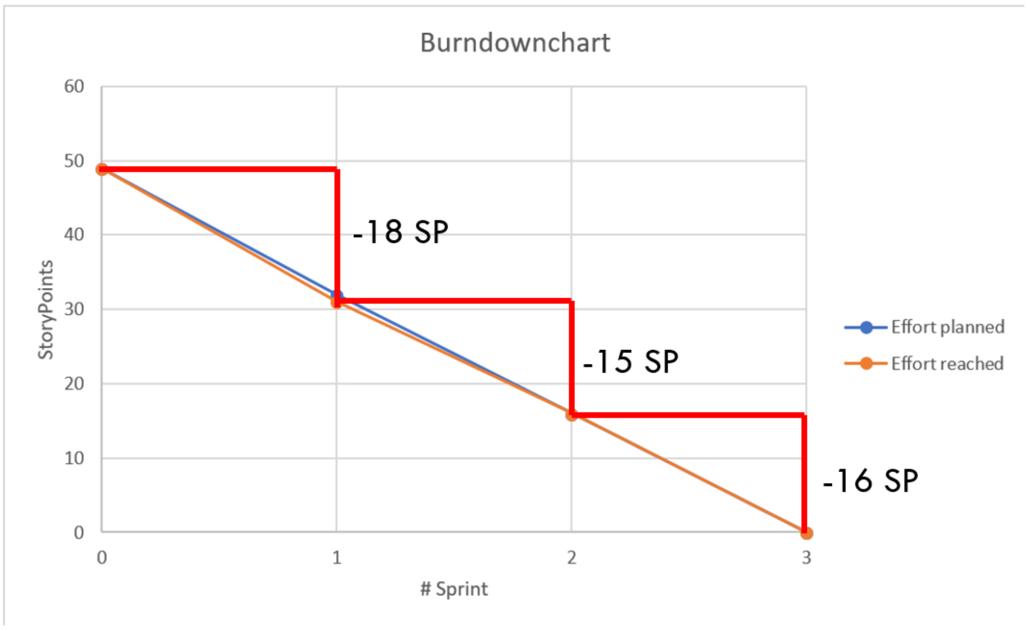
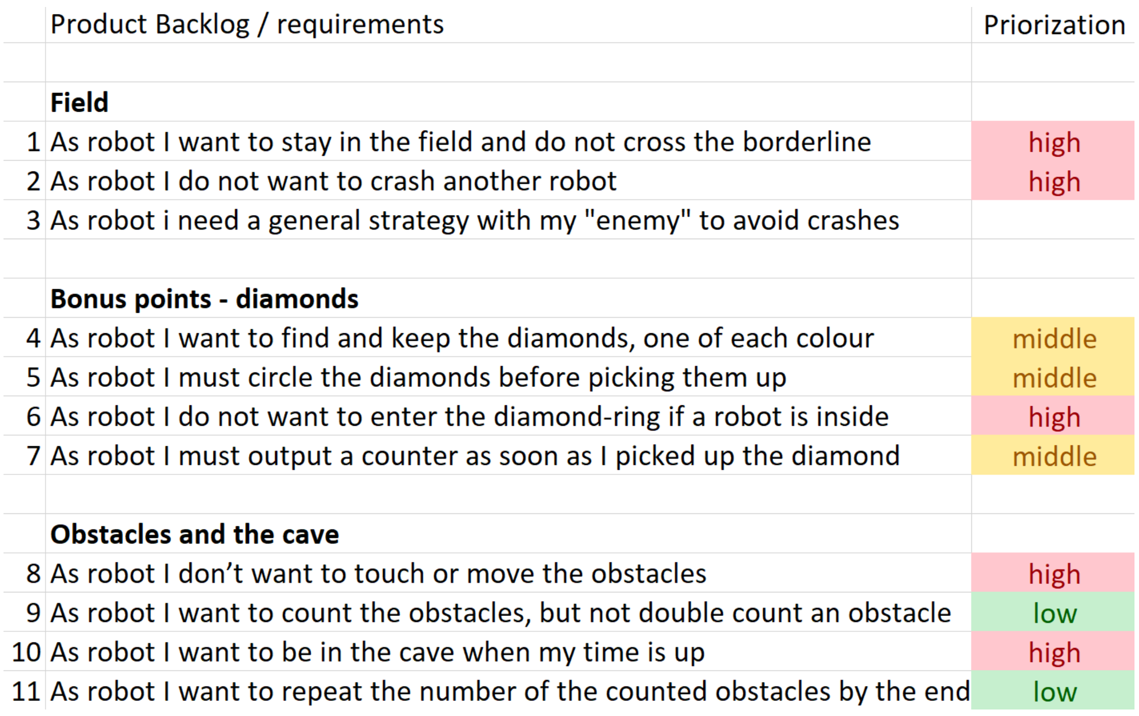
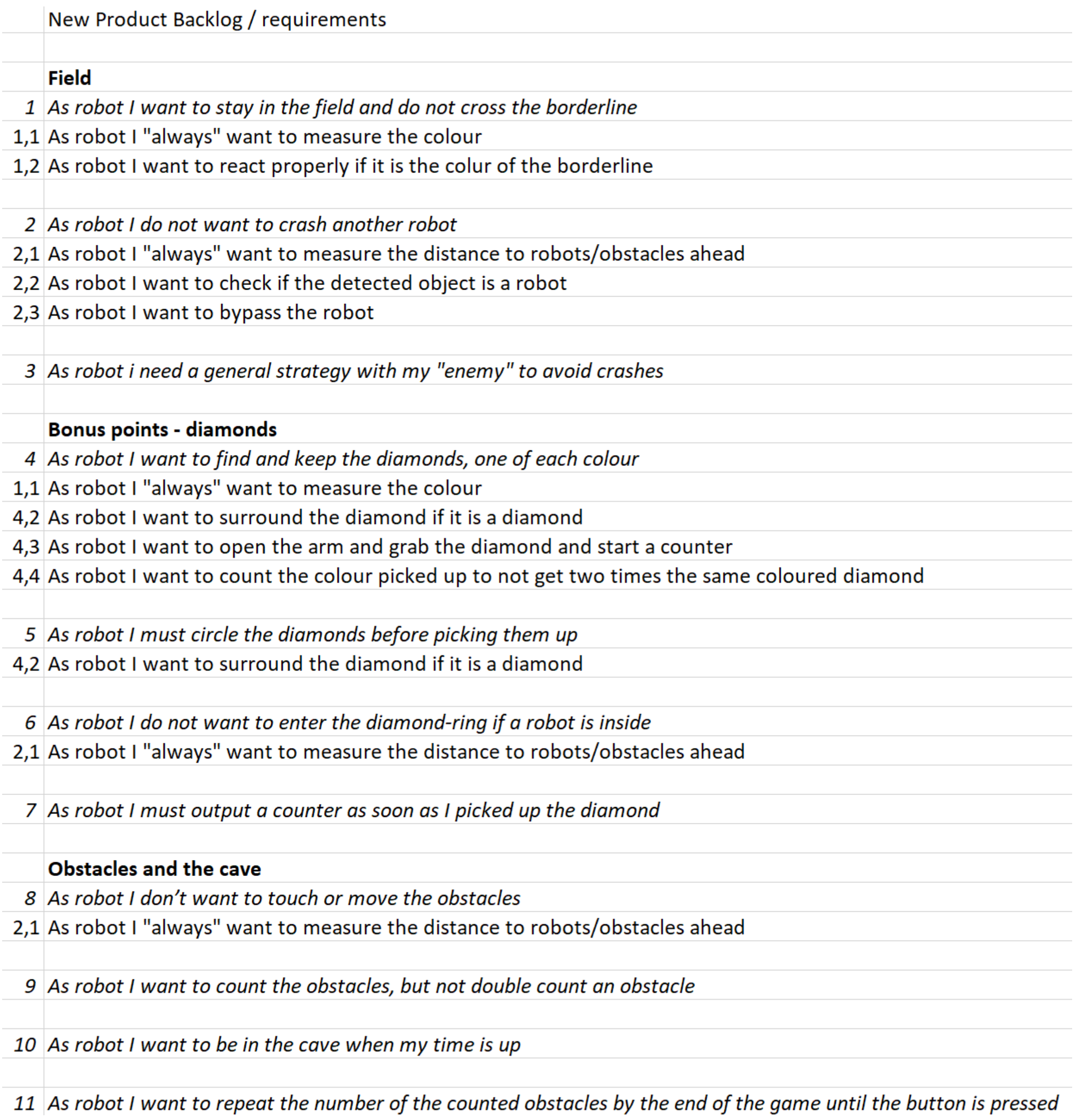


Figure 1: Burn-down chart for the 3 sprints



	Product Backlog / requirements	Priorization
	Field	
1	As robot I want to stay in the field and do not cross the borderline	high
2	As robot I do not want to crash another robot	high
3	As robot i need a general strategy with my "enemy" to avoid crashes	
	Bonus points - diamonds	
4	As robot I want to find and keep the diamonds, one of each colour	middle
5	As robot I must circle the diamonds before picking them up	middle
6	As robot I do not want to enter the diamond-ring if a robot is inside	high
7	As robot I must output a counter as soon as I picked up the diamond	middle
	Obstacles and the cave	
8	As robot I don’t want to touch or move the obstacles	high
9	As robot I want to count the obstacles, but not double count an obstacle	low
10	As robot I want to be in the cave when my time is up	high
11	As robot I want to repeat the number of the counted obstacles by the end	low

Figure 2: Former backlog



	New Product Backlog / requirements
	Field
1	*As robot I want to stay in the field and do not cross the borderline*
1,1	As robot I "always" want to measure the colour
1,2	As robot I want to react properly if it is the colur of the borderline
2	*As robot I do not want to crash another robot*
2,1	As robot I "always" want to measure the distance to robots/obstacles ahead
2,2	As robot I want to check if the detected object is a robot
2,3	As robot I want to bypass the robot
3	*As robot i need a general strategy with my "enemy" to avoid crashes*
	Bonus points - diamonds
4	*As robot I want to find and keep the diamonds, one of each colour*
1,1	As robot I "always" want to measure the colour
4,2	As robot I want to surround the diamond if it is a diamond
4,3	As robot I want to open the arm and grab the diamond and start a counter
4,4	As robot I want to count the colour picked up to not get two times the same coloured diamond
5	*As robot I must circle the diamonds before picking them up*
4,2	As robot I want to surround the diamond if it is a diamond
6	*As robot I do not want to enter the diamond-ring if a robot is inside*
2,1	As robot I "always" want to measure the distance to robots/obstacles ahead
7	*As robot I must output a counter as soon as I picked up the diamond*
	Obstacles and the cave
8	*As robot I don't want to touch or move the obstacles*
2,1	As robot I "always" want to measure the distance to robots/obstacles ahead
9	*As robot I want to count the obstacles, but not double count an obstacle*
10	*As robot I want to be in the cave when my time is up*
11	*As robot I want to repeat the number of the counted obstacles by the end of the game until the button is pressed*

Figure 3: Updated backlog

B Activity diagrams

All diagrams were made by us in January 2020.

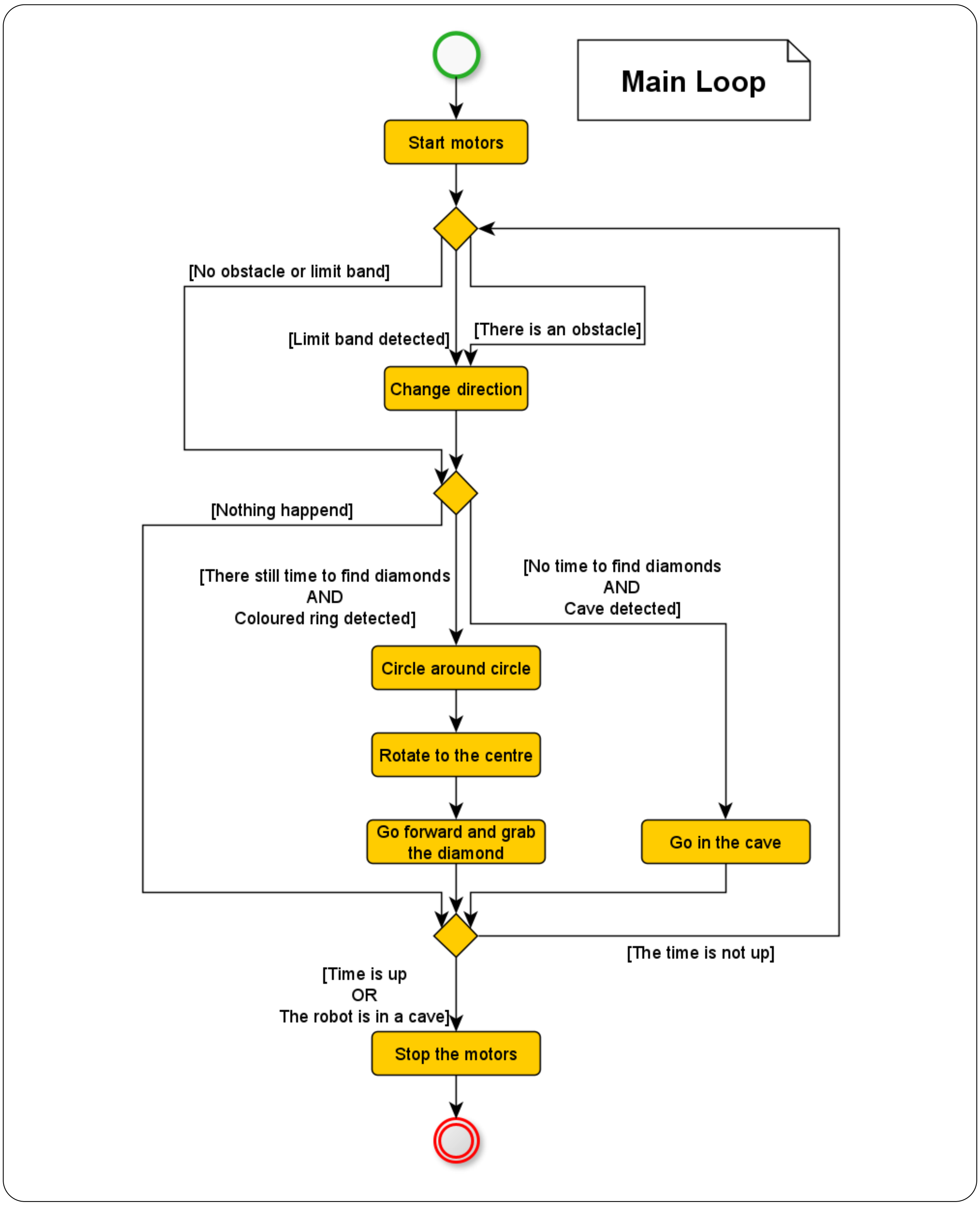


Figure 4: Activity diagram: Main loop

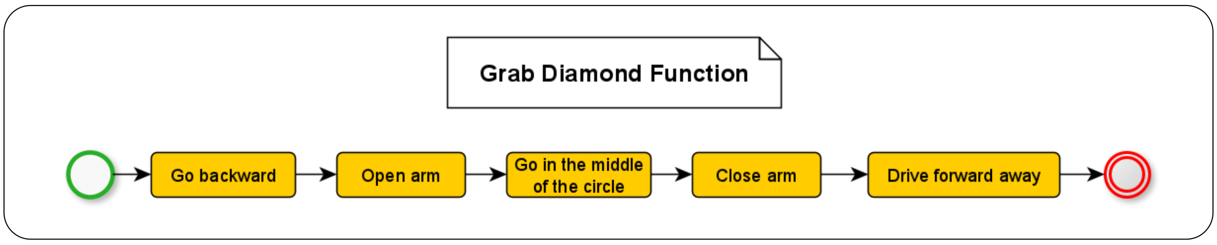


Figure 5: Activity diagram: Grab diamond function

.

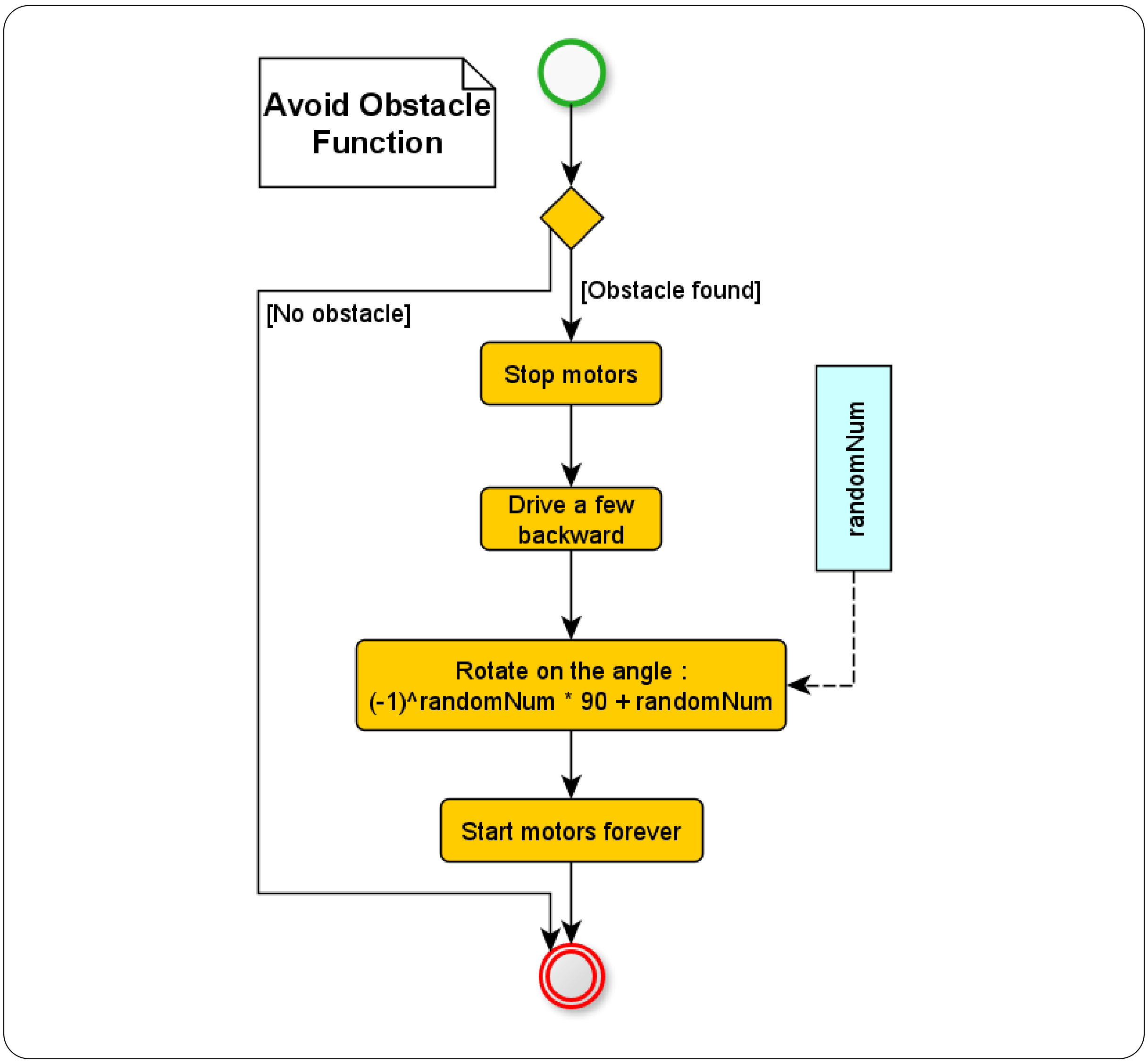


Figure 6: Activity diagram: Avoid obstacles functions

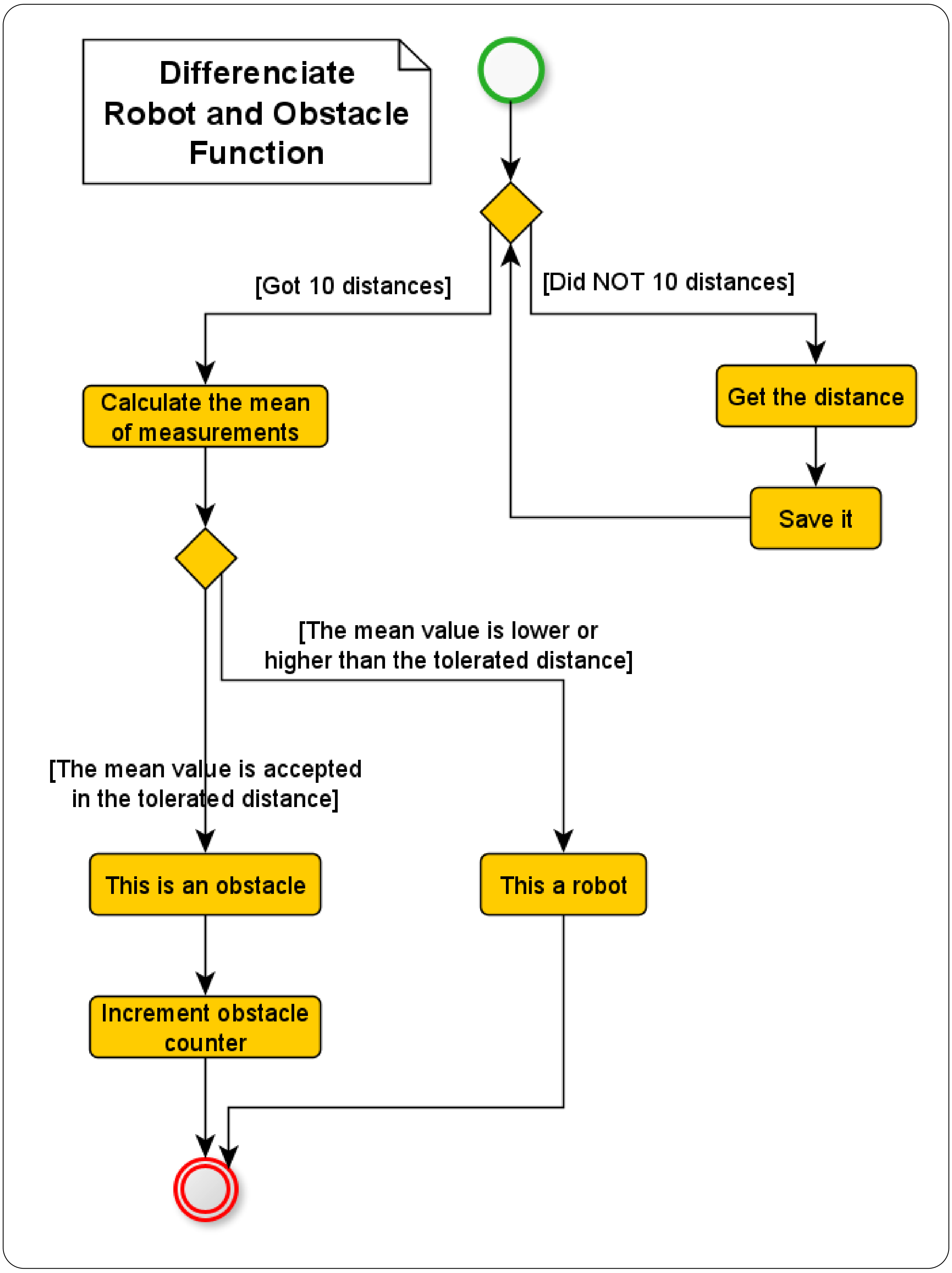

Differenciate Robot and Obstacle Function
[Got 10 distances]
[Did NOT 10 distances]
Get the distance
Save it
Calculate the mean of measurements
[The mean value is lower or higher than the tolerated distance]
[The mean value is accepted in the tolerated distance]
This is an obstacle
This a robot
Increment obstacle counter

Figure 7: Activity diagram: Differentiate robot and obstacle

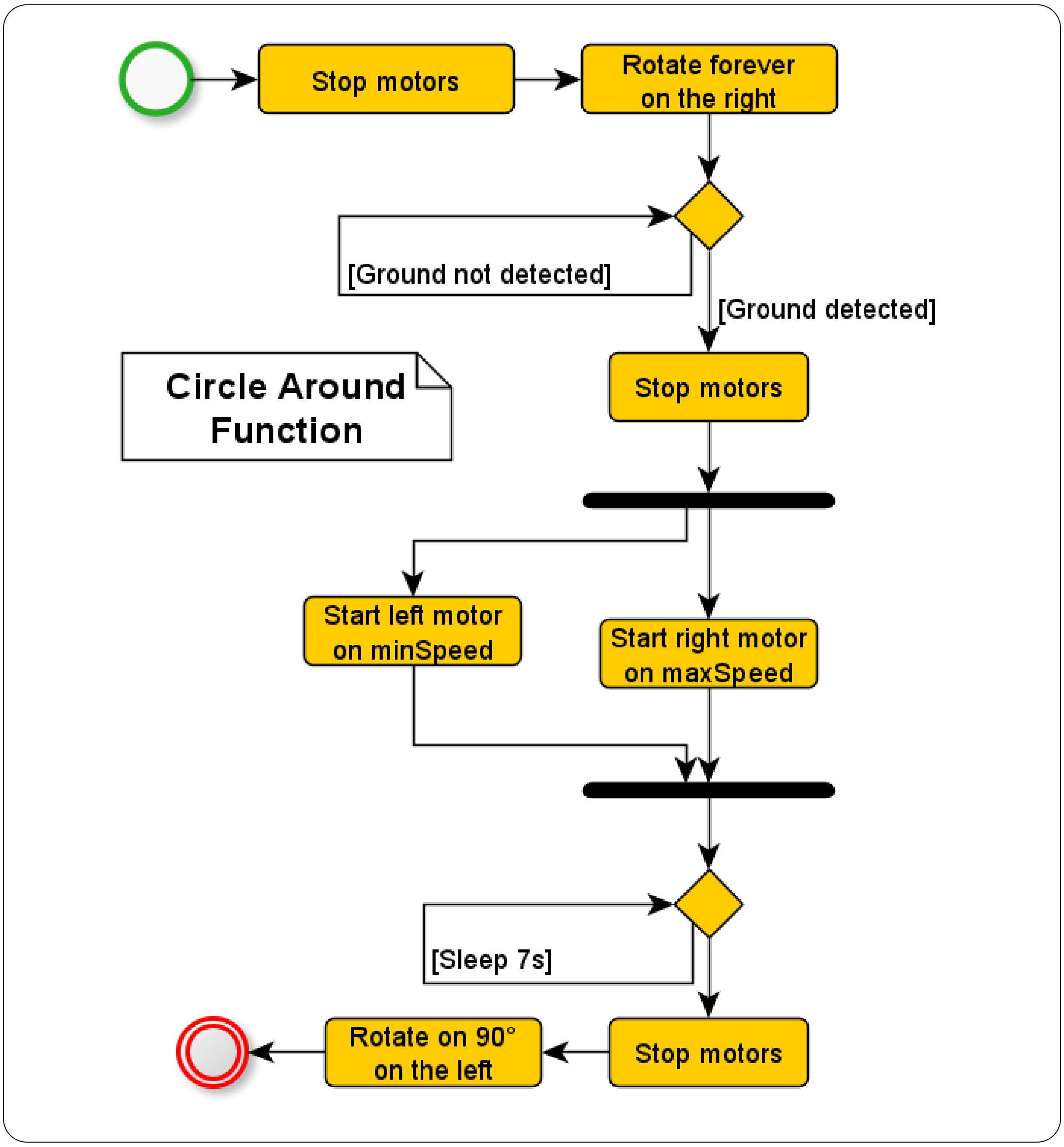

Stop motors
Rotate forever on the right
[Ground not detected]
[Ground detected]
Circle Around Function
Stop motors
Start left motor on minSpeed
Start right motor on maxSpeed
[Sleep 7s]
Rotate on 90° on the left
Stop motors

Figure 8: Activity diagram: Rotate around a circle

C Challenge Program Source Code

```
# -*- coding: utf-8 -*-
"""
Creation: 2020 01 18 14:00
Authors: Guillaume COUZINET & Elias NEUMAIER
Context: Computer Sciences for Engineers -- TEC Reutlingen Hochschule
"""


import ev3dev.ev3 as ev3
import math
import logging
import time
from statistics import mean

logging.basicConfig(format='%(levelname)s:%(message)s', level=logging.DEBUG)
logging.info("Program : START")

'''
--------------------------------------------------------------------------------
------ CONSTANTS AND GLOBALS ----------------------------------------------------
--------------------------------------------------------------------------------
'''
# ------ CONSTANTS -------

# Time of the challenge (in second)
CHALLENGE_TIME = 300
TIME_TO_FIND_DIAMONDS = 180

# Set the colours of the elements
```

```
RINGS_COLOURS = [4, 5]
LIMIT_BAND_COLOUR = 3
CAVE_COLOUR = 6
GROUND_COLOUR = 1

# ------ Function : is_distance_ok
# DISTANCE_MIN is the minimum permissible distance (in millimetres)
DISTANCE_MIN = 150
TOLERANCE = 10


# ------- GLOBAL VARIABLES -------

# Motors definition
MotorRight = ev3.LargeMotor('outB')
MotorLeft = ev3.LargeMotor('outA')

# Distance sensor definition
UltraSonic = ev3.UltrasonicSensor('in2')

# Put the US sensor into distance mode.
#   <!> The sensor return values in millimetres <!>
UltraSonic.mode='US-DIST-CM'

# -- Gyroscope --
Gyro = ev3.GyroSensor('in4')
Gyro.mode = 'GYRO-ANG'


# Definition of colour sensor and setting the mode
ColSensor = ev3.ColorSensor('in1')
```

```
ColSensor.mode = 'COL—COLOR'
# inputs of the different colours in 'COL—COLOR'—mode
# 0: No color   1: Black   2: Blue   3: Green   4: Yellow   5: Red   6: White
# 7: Brown

# Medium motor (needed to open the arm) definition
ArmMotor = ev3.MediumMotor('outD')

# Button sensor (needed to start the challenge)
Touch = ev3.TouchSensor('in3')

# ObstacleCounter is the obstacle counter
ObstacleCounter = 0


'''
————————————————————————————————————————————————————————————————————————————————
——————— MATHS ——————————————————————————————————————————————————————————————————
————————————————————————————————————————————————————————————————————————————————
'''

def angleCalculation(distance, RADIUS=28):
    """
    Function calculating the angle of rotation with a distance
        and the radius of the wheels.

    distance —— the value of the distance to reach in millimeter
    RETURN —— angle of rotation in degres
    """
    return (distance * 360) / (2 * math.pi * RADIUS)
```

```
'''
--------------------------------------------------------------------------------
------- MOTORS -------------------------------------------------------------------
--------------------------------------------------------------------------------
'''

def start_motors_forever(speed):
    '''
    Function giving the start signal to the motors with a
        given operating speed.

    speed -- value between 100 and 900 indicating the rate of the maximum
        possible speed to reach.
    '''
    global MotorRight, MotorLeft
    MotorRight.run_forever(speed_sp=speed)
    MotorLeft.run_forever(speed_sp=speed)




def stop_motors(action='brake'):
    '''
    Function that sends the stop signal to the motors.

    action -- value indicating the stopping method to be used by the motors.
        Refer to the official EV3 documentation.
    '''
    global MotorRight, MotorLeft
    MotorRight.stop(stop_action=action)
    MotorLeft.stop(stop_action=action)
```

```
def rotate_motors_right():
    MotorLeft.run_forever(speed_sp=100)
    MotorRight.run_forever(speed_sp=-100)


def rotate_motors_left():
    MotorLeft.run_forever(speed_sp=-100)
    MotorRight.run_forever(speed_sp=100)


'''
-------------------------------------------------------------------------------
------- PATH ------------------------------------------------------------------
-------------------------------------------------------------------------------
'''


def driveForward(distance, speed=500):
    """
    Function to drive the robot straight forward for a given distance

    distance -- the value of the distance to reach in millimeter
    """
    global MotorRight, MotorLeft

    MotorRight.run_to_rel_pos(position_sp=angleCalculation(distance),
        speed_sp=speed)
    MotorLeft.run_to_rel_pos(position_sp=angleCalculation(distance),
        speed_sp=speed)
    logging.info('DriveForward : Start to drive')
    MotorRight.wait_while('running')
    MotorLeft.wait_while('running')
    logging.info('DriveForward : Stop')


def rotate(angle):
    """
    Function rotating clockwisely the robot to a given angle.
        We want use the smallest angle possible.
        That is why we use the modulo operator.
        Then we calculate the difference between the angle before moving and
        the angle during the movement.

    angle -- the angle (negative or positive) of rotation to reach in degres
    """
    global MotorRight, MotorLeft, Gyro

    # "angleTarget" is the variable geting the rest of the
    #   angle divided by 360 degrees
    angleTarget = 0
    # "angleOld" is the angle measured by the GyroscopSensor before moving.
    angleOld = Gyro.value()

    if (angle % 360 <= 180):
        # If the rest is smaller than 180 degrees, we turning on the right.
        angleTarget = angle % 360
        rotate_motors_right()
    else:
        # If the rest is bigger than 180 degrees, we turning on the left.
        # Then we calculate the right angle.
        angleTarget = 360 − (angle % 360)
        rotate_motors_left()

    # FLAG boolean
    angleReached = False
    logging.info('Rotation : Rotate until the target angle is reached')
    while not angleReached:
        if abs(angleOld − Gyro.value()) >= angleTarget:
            angleReached = True
            stop_motors()
            logging.info('Rotation : Stop')



'''
––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
––––––– FUNCTIONS ––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
'''

def is_distance_ok():
    '''
    Function that compares the distance between the robot and the nearest
        object facing it. If this object is below the minimum distance value,
        the stop signal is sent to the motors.

    RETURN −− Boolean indicating if the robot is too close of an obstacle.
    '''
    distanceOK = True

    distance = UltraSonic.value()
    if distance <= DISTANCE_MIN:
        distanceOK = False

    return distanceOK



def circle_found(ringColour):
    '''
    Function setting up the robot in position before do the circle.

    ringColor -- is the color of the detected circle.
    '''
    logging.info('CAR : START')

    logging.info('CAR : Rotate to find the ground')
    rotate_motors_right()
    reachGroundColor = False

    while not reachGroundColor:
        if ColSensor.value() == GROUND_COLOUR:
            logging.info('CAR : Ground found !')
            reachGroundColor = True
            stop_motors()

    driveAround(ringColour)
    logging.info('CAR : END')



def driveAround(ringColour):
    '''
    Function that determines the rotational speed of the right
        motor by dichotomy.

    ringColor -- is the color of the detected circle.
    '''
    logging.info('Circle : START')
    global MotorRight, MotorLeft

    minSpeed = 300
    maxSpeed = 500

    logging.info('Circle : Start motors')
    MotorLeft.run_forever(speed_sp=minSpeed)
    MotorRight.run_forever(speed_sp=maxSpeed)

    logging.info('Circle : Sleep for 7s during the drive')
    time.sleep(7)
    logging.info('Circle : Stop motors')
    stop_motors()
    logging.info('Circle : END')


def open_grab_close():
    logging.info('OGC : START')
    logging.info('OGC : Going into the circle')
    # Go back
    driveForward(-150, speed=500)

    # Open the arm
    logging.info('OGC : Open arm')
    ArmMotor.run_timed(time_sp=3500, speed_sp=-900)
    ArmMotor.wait_while('running')

    # Drive in the middle of the circle
    logging.info('OGC : Grab the diamond')
    driveForward(300, speed=500)

    # Close the arm
    logging.info('OGC : Close arm')
    ArmMotor.run_timed(time_sp=3500, speed_sp=900)
    ArmMotor.wait_while('running')

    # Drive away forward
    logging.info('OGC : Going away')
    driveForward(250, speed=500)
    logging.info('OGC : END')


'''
--------------------------------------------------------------------------------
------- MAIN LOOP --------------------------------------------------------------
--------------------------------------------------------------------------------
'''

logging.info('Program : Robot ready, waiting...')
while Touch.value() == 0:
    time.sleep(0.01)
    logging.info('Program : Button pressed !')

startTime = time.time()
timeToDiamonds = True
inTheCave = False
```

```
loopCounter = 0

logging.info('Program : Starting Main Loop !')

while ( time.time() - startTime ) <= CHALLENGE_TIME:

    if loopCounter > 90:
        loopCounter = 0
    else:
        loopCounter = loopCounter + 1

    if (time.time() - startTime) > TIME_TO_FIND_DIAMONDS:
        logging.info('Main : No time for diamonds')
        timeToDiamonds = False

    if not inTheCave:
        start_motors_forever()

    # Check the distance to be sure that there is no obstacle
    #   or robot ahead and bypass if there is something
    if not is_distance_ok():
        logging.info('Main : Obstacle detected')
        stop_motors()
        driveForward(distance=-200)
        rotate( (90 + loopCounter) * (math.pow(-1, loopCounter)))

    # Check the colour, if its the colour of the bands turn around
    if ColSensor.value() == LIMIT_BAND_COLOUR:
        logging.info('Main : Limit band detected')
        stop_motors()
        driveForward(distance=-200)
```

```
            rotate( (90 + loopCounter) * (math.pow(-1, loopCounter)))

        # Check the colour, if its one of the diamonds, circle the diamond
        #   and collect it
        if ColSensor.value() in RINGS_COLOURS and timeToDiamonds:
            logging.info('Main : Circle detected')
            colour = ColSensor.value()
            stop_motors()
            circle_found(colour)
            rotate(-90)
            open_grab_close()

        # Check the colour is a cave and if the time is up to go in a cave
        if ColSensor.value() == CAVE_COLOUR and not timeToDiamonds:
            logging.info('Main : Cave detected')
            stop_motors()
            driveForward(150)
            inTheCave = True

logging.info('Program : Stop motor, time is up, challenge ended')
stop_motors()

logging.info('Program : END')
```

Follow the link to get the online project:

https://github.com/GuillaumeCou/ReHo1920_ComputerScience

D Unused or Deleted Functions

Differentiation Robot or Obstacle

```
def is_obstacle(t_obs=5, q_obs=10):
    '''
    Function that, after an observation period, indicates the nature of the
        obstacle in front of the robot.

    t_obs -- time for do several observations. (seconds)
    q_obs -- quantity of observations to do. (integer)
    RETURN -- The returned value is a Boolean indicating whether the obstacle
        encountered is a simple obstacle or not.
    '''
    distanceList = []
    isObstacle = False
    i = 0

    # Observation period
    while i < q_obs:
        distanceList.append(ufc.ultraSonic.value())
        i = i+1
        ev3.Sound.beep()
        sleep(t_obs/q_obs)

    if mean(distanceList) <= (DISTANCE_MIN + TOLERANCE):
        isObstacle = True

    return isObstacle
```

Drive around ring : speed modulation

```python
def speed_modulation_rotation(ringColour):
    '''
    Function that determines the rotational speed of the right
        motor by dichotomy.

    ringColor -- is the color of the detected circle.
    '''
    global MotorRight, MotorLeft, ColSensor

    lastSpeedMR = CIRCLE_MAX_SPEED
    maxSpeed = lastSpeedMR
    minSpeed = SPEED_DRIVING

    startTime = time.time()
    lastColour = ColSensor.value()
    MotorLeft.run_forever(SPEED_DRIVING)

    while (time.time() - startTime) <= TIME_TO_CIRCLE:
        if ColSensor.value() == GROUND_COLOUR and GROUND_COLOUR != lastColour:
            minSpeed = lastSpeedMR
            lastSpeedMR = (maxSpeed + lastSpeedMR)/2

        if ColSensor.value() == ringColour and ringColour != lastColour:
            maxSpeed = lastSpeedMR
            lastSpeedMR = (minSpeed + lastSpeedMR)/2

        MotorRight.run_forever(speed_sp=lastSpeedMR)
```

Bypass Obstacle

```
def bypass_robot_and_obstacle():

    logging.info('Start to bypass the obstacle')

    # Turn left
    rotate(angle=BYPASS_ANGLE, speed=SPEED_ROTATION)
    driveForward(distance=BYPASS_DISTANCE)

    # Turn right
    rotate(angle=(-1)*BYPASS_ANGLE, speed=SPEED_ROTATION)
    driveForward(distance=BYPASS_DISTANCE)

    # Turn right
    rotate(angle=(-1)*BYPASS_ANGLE, speed=SPEED_ROTATION)
    driveForward(distance=BYPASS_DISTANCE)

    # Turn left
    rotate(angle=BYPASS_ANGLE, speed=SPEED_ROTATION)

    logging.info('End to bypass the obstacle')
```