# Software design
# Team project – Deliverable 2

**Team number**: 20

**Team members:**

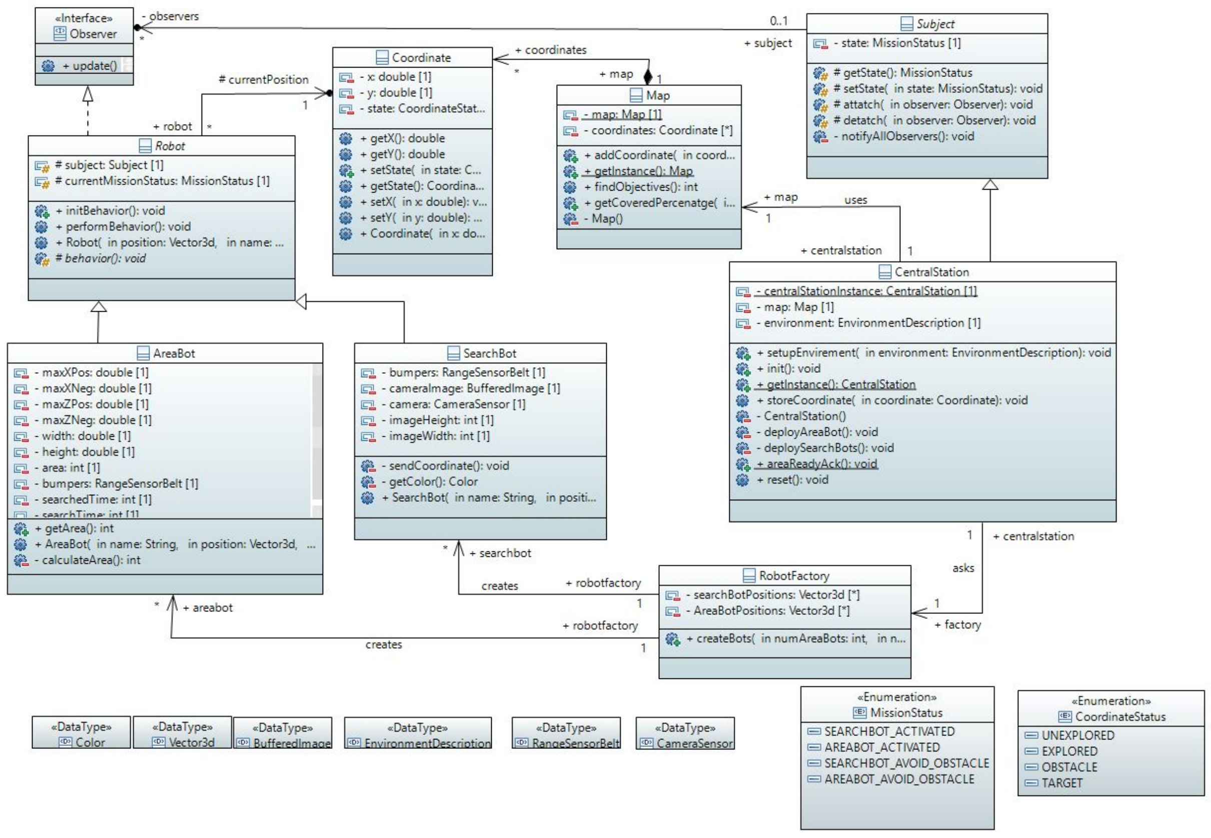
| **Name** | **Student Nr.** | **E-mail** |
|---|---|---|
| Ronan Hochart | 2614393 | rhochart@gmail.com |
| Mateo Vakili | 2619616 | mateovakili@gmail.com |
| Nick Kozanidis | 2616924 | nickkozanidis@gmail.com |
| Louk Onkenhout | 2619109 | loukonkenhout@hotmail.com |

This document has a maximum length of 10 pages.

## Table of Contents

## 1. Class diagram

**Author(s)**: <Ronan Hochart, Mateo Vakili, Louk Onkenhout>



«Interface» Observer
+ update()
Subject
- state: MissionStatus [1]
# getState(): MissionStatus
# setState( in state: MissionStatus): void
# attatch( in observer: Observer): void
# detatch( in observer: Observer): void
- notifyAllObservers(): void
Coordinate
- x: double [1]
- y: double [1]
- state: CoordinateStat...
+ getX(): double
+ getY(): double
+ setState( in state: C...
+ getState(): Coordina...
+ setX( in x: double): v...
+ setY( in y: double): ...
+ Coordinate( in x: do...
Map
- map: Map [1]
- coordinates: Coordinate [*]
+ addCoordinate( in coord...
+ getInstance(): Map
+ findObjectives(): int
+ getCoveredPercenatge( i...
- Map()
Robot
# subject: Subject [1]
# currentMissionStatus: MissionStatus [1]
+ initBehavior(): void
+ performBehavior(): void
+ Robot( in position: Vector3d, in name: ...
# behavior(): void
CentralStation
- centralStationInstance: CentralStation [1]
- map: Map [1]
- environment: EnvironmentDescription [1]
+ setupEnvirement( in environment: EnvironmentDescription): void
+ init(): void
+ getInstance(): CentralStation
+ storeCoordinate( in coordinate: Coordinate): void
- CentralStation()
- deployAreaBot(): void
- deploySearchBots(): void
+ areaReadyAck(): void
+ reset(): void
AreaBot
- maxXPos: double [1]
- maxXNeg: double [1]
- maxZPos: double [1]
- maxZNeg: double [1]
- width: double [1]
- height: double [1]
- area: int [1]
- bumpers: RangeSensorBelt [1]
- searchedTime: int [1]
- searchTime: int [1]
+ getArea(): int
+ AreaBot( in name: String, in position: Vector3d, ...
- calculateArea(): int
SearchBot
- bumpers: RangeSensorBelt [1]
- cameraImage: BufferedImage [1]
- camera: CameraSensor [1]
- imageHeight: int [1]
- imageWidth: int [1]
- sendCoordinate(): void
- getColor(): Color
+ SearchBot( in name: String, in positi...
RobotFactory
- searchBotPositions: Vector3d [*]
- AreaBotPositions: Vector3d [*]
+ createBots( in numAreaBots: int, in n...
«DataType» Color
«DataType» Vector3d
«DataType» BufferedImage
«DataType» EnvironmentDescription
«DataType» RangeSensorBelt
«DataType» CameraSensor
«Enumeration» MissionStatus
SEARCHBOT_ACTIVATED
AREABOT_ACTIVATED
SEARCHBOT_AVOID_OBSTACLE
AREABOT_AVOID_OBSTACLE
«Enumeration» CoordinateStatus
UNEXPLORED
EXPLORED
OBSTACLE
TARGET
- observers
0..1
+ subject
+ coordinates
+ map
# currentPosition
+ robot
+ map
uses
+ centralstation
+ searchbot
creates
+ robotfactory
+ areabot
asks
+ factory

## Class Subject:

The Subject class is setup to manage a list of observers and notify the observers of any state changes that happen to the central station. It accomplishes this task by attaching or detaching observers.

Attributes:

- List<Observer> observers = new ArrayList<Observer>(): This is the list where all the attached observers will be recorded and stored for management. This is private.
- MissionStatus state: This will be used to store the *MissionStatus* which will be updated frequently to notify all observers. This will be used to manage which bot is active when and what it is currently doing.

Associations:

- Subject is associated with Observer. This association allows Subject to keep a list of all the observers its working with and update their states periodically.
- Subject has a generalization to CentralStation and its detailed in the CentralStation class.

Operations:

- void notifyAllObservers(): This function cycles through every observer in the list and updates their state. This is private.

- void detatch(Observer observer): This function removes an observer from the list. This is protected.
- void attatch(Observer observer): This function adds an observer from the list. This is protected.
- void setState(MissionStatus state): This function would update the state variable and then call the notifyAllObservers() function to update all observers to the new state. This is protected.
- missionStatus getState(): Returns the state stored in the state variable. This is protected

## Class CentralStation:

The CentralStation class is setup as a singleton meaning only one instance of the class will ever be created during the programs operation. The role of the class is to manage the bots both in terms of their mission objectives and when each bot is activated, gather, and process the data produced by each type of bot for further use and set up the whole environment where the robots will be active in the first place.

Attributes:

- CentralStation centralStationInstance: This will be used as a reference to the only CentralStation instance making it a singleton
- Map map: This will be used as a reference to the only Map instance. This instance will be used to store all the coordinate data for future use
- EnvironmentDescription environment: The simbad environment will be created and then stored into this object. It will be used to generate the environment with the bots spawned in therefore it is the current *EnvironmentDescription* object being used.

Associations:

- CentralStation is associated with RobotFactory. This association allows CentralStation to get a list of created bots from the RobotFactory object (the number and the type of the bots is specified by the central station).
- CentralStation is associated with Map. This association allows CentralStation to use the Map so that it can log and use the coordinates gathered from the bots on their mission.
- CentralStation has a generalization to Subject inheriting its properties. As mentioned The Central Station  manages the bots both in terms of their mission objectives and when each bot is activated therefore it should be able to  notify all observers and change their state.

Operations:

- CentralStation(): This is the constructor for the class. It is private as the Central Station is a Singleton .
- void deploySearchBot(): This function enables the SearchBots by setting the mission status to areabot_activated. It is private.
- void deployAreaBot(): This function enables the AreaBots by setting the mission status to searchbot_activated. It is private.
- void storeCoordinate( Coordinate coordinate): Using the map object and an input coordinate it will send that coordinate to the map class for storage. This is public so it can be called by SearchBot which sends the coordinates to Central Station.
- CentralStation getInstance(): Returns the instance of CentralStation. This is public

- void init(): Initializes all the tools needed to run the simulation. This includes the RobotFactory where it will designate how many bots is needed. Moreover, the Central Station will spawn the produced bots  and lastly sets the status of the bots at the start of the simulation. This is public.
- void setupEnvironment(EnvironmentDescription environment): Stores the environment for simbad from the environment class. The created environment is stored in the environment object inside the class for future use. This is public.
- void reset(): Resets the Mission Status. When the Reset button is selected in the simbad simulator this function will be called. this is public.
- static void areReadyAck(): Stores the area of the environment. This function is called by the AreaBot when it’s search time has reached the required limit to report its calculations.

## Class Map:

The Map class is setup as a singleton meaning only one instance of the class will ever be in use during the programs operation. The role of the class is to manage and store the coordinates for other classes to use. Therefore all functions in the class are related to storage or extracting stored data with some processing.

Attributes:

- List<Coordinate> coordinates: A list of coordinates. All coordinates reported from SearchBot will be stored in this list using the addCoordinate function.
- Map map: This will be used as a reference to the only Map instance making it a singleton. This instance will be used to store all the coordinate data for future use

Associations:

- Map is associated with CentralStation. This association is detailed in the CentralStation class.
- Map is associated with Coordinate. This association allows Map to use the Coordinate class as a datatype and gives it access to its public functions used to extract specific data from coordinates.

Operations:

- Map(): This is the constructor for the class. It is private.
- int getCoveredPercentage(int area): This function simply computes the percentage of the area that has been explored. It will then return this value. It is Public.
- int findObjectives(): This function will sort through the list containing all the coordinates and determine using an algorithm how many mission objectives were found in the environment. It is public.
- Map getInstance(): Returns the instance of Map. This is public
- void addCoordinate(Coordinate coordinate): This function takes as input a coordinate and adds it to the list. The coordinate only gets added if no other entry has the same coordinate to avoid duplicates. It is public.

## Class Robot:

The Robot class is setup to organize and manage the two types of bots in the environment. It is an abstract class of type Observer and just serves to group the common functions shared among the two types of bots.

Attributes:

- Subject subject: This will store the subject instance for use in functions to manage the bots. This is protected.
- MissionStatus currentMissionStatus: This variable will store the *MissionStatus* that is currently active.

Associations:

- Robot has a generalization to AreaBot and SearchBot. These two generalizations are detailed in the AreaBot and SearchBot class respectfully.
- Robot has an association with the Coordinate class. This allows the Robot class to store and track the currentPosition of one bot at once.

Operations:

- Robot(Vector3d position, String name, Subject subject): This is the constructor for the class. It sets up the variables for each bot when they are initialized. The subject is stored here which will be used by both Search Bot and AreaBot to get the current state and act accordingly. It is public.

- abstract void behavior(): This function will control the behaviors of each individual bot and it is called inside the performBehavior() function. It is abstract meaning it should be implemented by classes that inherits it. It is overridden in AreaBot and SearchBot as they have different uses and therefore different behaviors. The obstacle avoidance algorithms are also stored in these overridden functions. It is protected.
- void performBehavior(): This is the function that the simbad simulator will call. It simply calls the behavior() function. It is public.
- void initBehavior(): This function initializes the status of everybot. It calls the reset function from CentralStation to set everybot to the same initial state.

## Class RobotFactory:

The RobotFactory class is responsible for creating the bots that will be used in the simulation.  It abstracts the process of robot creation.

Attributes:

- Vector3d[] searchBotPositions: An array initialized and storing potential positions for where SearchBots will spawn. This is private.
- Vector3d[] areaBotPositions: An array initialized and storing potential positions for where AreaBots will spawn. This is private.

Associations:

- RobotFactory is associated with CentralStation. This association is detailed in the CentralStation class.

- RobotFactory is associated with both SearchBot and Areabot in the same manner. These associations allows RobotFactory to create bots of those classes. Once the bots are created they no longer belong to the RobotFactory class.

Operations:

- List<Robot> createBot(int numAreaBots, int numSearchBots, EnvironmentDescription environment, Subject subject): This function is used to create all the bots needed for the program to run. The function calls other functions in searchbot/areabot which do the actual creating.

## Class AreaBot:

The AreaBot class is responsible for the operation of the 2nd type of bot available in the environment. This bot focuses on finding the area of the environment which is used in the operation of SearchBot. It uses a simple algorithm to avoid obstacles and uses bumpers to identify when it collides with an object. Its task is complete when the CentralStation calls one of its functions to return the area it calculated from its data.

Attributes:

- RangeSensorBelt bumpers: A simbad required object that when initialized will add bumpers to the bot allows for collision detection.
- double maxXPos: one of the four variables that will hold the data gathered from the robots coordinate system. This variable will be used in the calculation of the height. It is private.
- double maxZPos: one of the four variables that will hold the data gathered from the robots coordinate system. This variable will be used in the calculation of the width. It is private.
- double maxXNeg: one of the four variables that will hold the data gathered from the robots coordinate system. This variable will be used in the calculation of the width. It is private.
- double maxZNeg: one of the four variables that will hold the data gathered from the robots coordinate system. This variable will be used in the calculation of the height. It is private.
- double width: A variable holding the calculated width of the environment, one of the two variables needed in the calculation of the area. It is private.
- double height: A variable holding the calculated height of the environment, one of the two variables needed in the calculation of the area. It is private.
- int area: A variable that will hold the calculated area of the environment. It is private.
- int searchTime: a variable initialized with the max search time the bot will run for.
- int searchedTime: a variable initialized to 0. It will hold the time the bots has searched for.

Associations:

- AreaBot is associated with RobotFactory. This association is detailed in the RobotFactory class.
- AreaBot is a generalization of the abstract Robot. This allows it to inherit all properties from Robot.

Operations:

- int calculateArea(): This function calculates the area of the environment. It rounds the area up and returns an int.
- AreaBot(String name, Vector3d position, Subject subject): The constructor of the class. Is called from RobotFactory and is used to spawn AreaBots into the environment
- int getArea(): Returns the area of the environment after its been calculated. Returns an int.

## Class SearchBot:

The RobotFactory class is responsible for creating the bots that will be used in the simulation.  It abstracts the process of robot creation.

Attributes:

- RangeSensorBelt bumpers: A simbad required object that when initialized will add bumpers to the bot allows for collision detection. It is private.
- CameraSensor camera: A simbad required object that when initialized will add a camera to a bot, this enables our color detection. It is private.
- BuffedImage cameraimage: A variable that will store the generated image from the camera. This will be able to be processed at a later time to extract data. It is private.
- int imageWidth: This variable will hold the width . It is protected.
- int imageHeight: one of the four variables that will hold the data gathered from the robots coordinate system. This variable will be used in the calculation of the height. It is protected.

Associations:

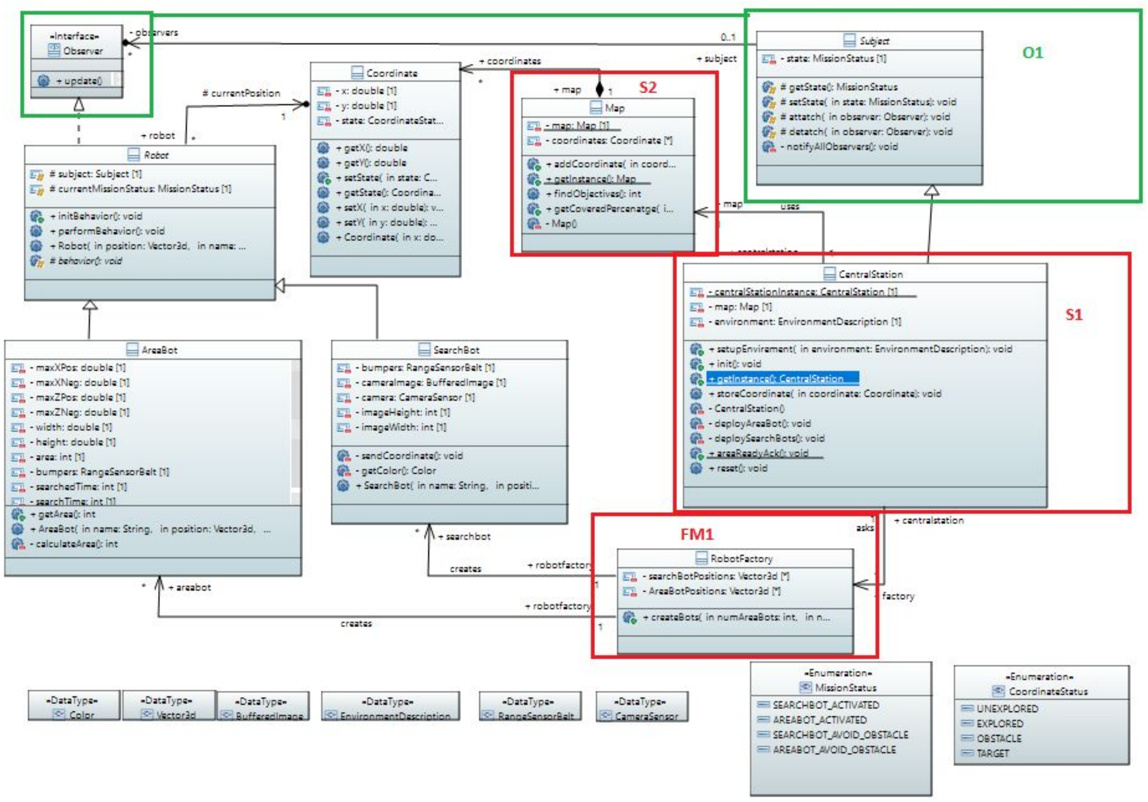
- SearchBot is associated with RobotFactory. This association is detailed in the RobotFactory class.
- SearchBot is a generalization of the abstract Robot. This allows it to inherit all properties from Robot.

Operations:

- SearchBot(String name, Vector3d position, Subject subject): The constructor of the class. Is called from RobotFactory and is used to spawn AreaBots into the environment. It is public.
- Color getColor(): This function takes a snapshot from the camera and processes it in order to get the color at the center of the image. This color is then used as the return value. This is private.
- void sendCoordinate(): This function sends the robots current position to the CentralStation class in order for it to get stored. This is private.

## 2. Applied design patterns

**Author(s)**: <Louk Onkenhout>



Design Pattern Descriptions:

| | |
|---|---|
| **ID** | S1 |
| **Design pattern** | Singleton |
| **Problem** | There must be always one and only one Central Station present, the reason being that the responsibility of updating mission parameters as well as general communication is covered by the Central Station. If there are multiple instances of the Central Station, communication becomes too complicated. |
| **Solution** | The Central Station class is a singleton, meaning any other object with relevant authority to access an object of this class should have the same reference to it. In the singleton design pattern, this is done by not directly instantiating from a class, but requesting the active instance (if there is one, otherwise it is generated) through a simple, well-defined function. |
| **Intended use** | Before the mission is started, when the environment has been generated, any bots that need a reference to the Central Station can get one by calling the function "getInstance", which will provide a reference to the only active Central Station. |
| **Constraints** | |
| **Additional remarks** | - |

| | |
|---|---|
| **ID** | S2 |
| **Design pattern** | Singleton |
| **Problem** | There must be always one and only one Map present, all the recorded coordinates should be stored in the same location, this makes maintaining it the most simple. |
| **Solution** | The Map class is also a singleton, the same reasoning applies as with the Central station class, where any object that requires a reference (in this case only the Central Station will be communicating with a Map object) can get one to the active Map instance by calling a similar simple, well-defined function. |
| **Intended use** | When the Central Station is first instantiated, it will call the "getInstance" method of the class Map to acquire the sole reference to that object (if there is any, otherwise it will be generated first). |
| **Constraints** | There must be an active Central Station instance in order for the Map singleton to exist. |
| **Additional remarks** | - |

| | |
|---|---|
| **ID** | O1 |
| **Design pattern** | Observer |
| **Problem** | There must be a way for the bots to efficiently communicate with the Central Station, this is essential so that the Map can be updated quickly and progress tracked appropriately. |
| **Solution** | An Observer-Subject pattern is applied to two of the classes. There will be one and only one Subject (the Central Station) and multiple Observers (all active bots in the environment).<br>The Central Station (subject) maintains a list with all the bots (observers), any time a state change occurs, the Central Station will send the updated state to all bots that require it to operate correctly. This works by having the Central Station force all bots to update, this update will simply request any relevant state information from the Central Station. |
| **Intended use** | Whenever a change of state occurs, e.g. when the AreaBots are done finding the Area, the Central Station notices this and changes the state, and requires all bots to update, alter their behaviour based on that. |
| **Constraints** | |
| **Additional remarks** | - |

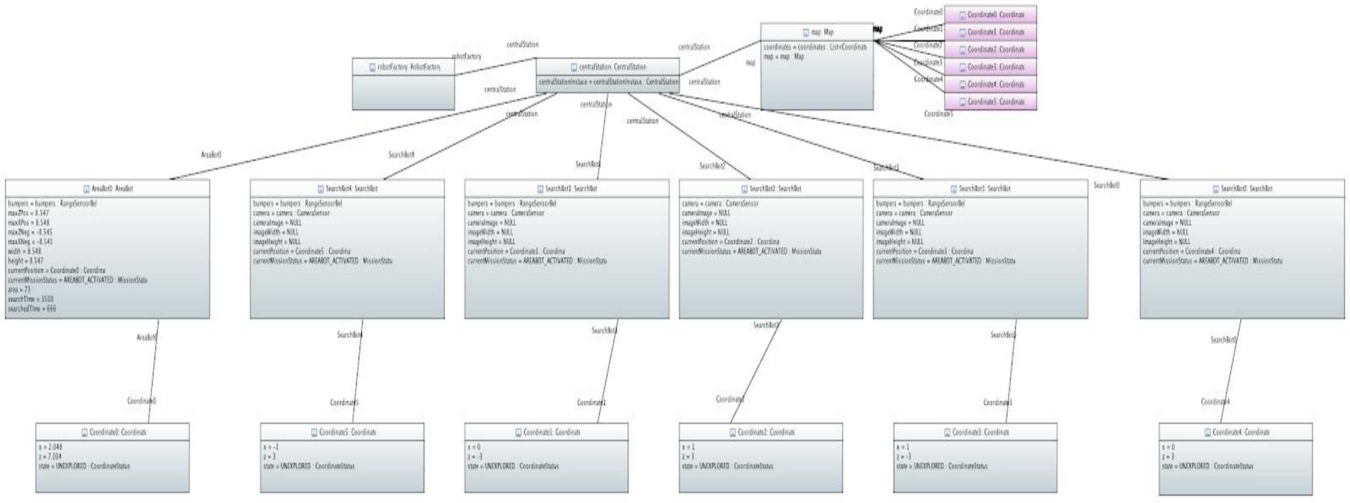
| | |
|---|---|
| **ID** | FM1 |
| **Design pattern** | Factory Method |
| **Problem** | The design must allow for some customizability in terms of object creation at run-time. Avoiding the 'new' operator will abstract away from the specifics of very closely related classes. |
| **Solution** | An intermediary Factory class is introduced to hide the 'new' operator away in. This way the Factory class can be the class that actually generates objects that are closely related (e.g. the bots). This allows for different setups at run-time. |

| | |
|---|---|
| **Intended use** | The Factory class in our design, will be the RobotFactory, when the bots are first created, a RobotFactory object should be created. This object has the createBots operation, which allows to create both types of bots (specified in the parameters: type and number of bots) in bulk and this simply returns all the newly created bots in a list. |
| **Constraints** | Only the RobotFactory class should be used in order to create bots, this is important to maintain the abstraction that this class provides. |
| **Additional remarks** | - |

*Maximum amount of pages: 2*

## 3. Object diagram

**Author(s)**:<Nick Kozanidis>



robotFactory: RobotFactory
centralStation: CentralStation
centralStationInstace = centralStationInstace : CentralStation
map: Map
coordinates = coordinates : List<Coordinate
map = map : Map
Coordinate0: Coordinate
Coordinate1: Coordinate
Coordinate2: Coordinate
Coordinate3: Coordinate
Coordinate4: Coordinate
Coordinate5: Coordinate
robotFactory
centralStation
map
AreaBot0
SearchBot0
SearchBot1
SearchBot2
SearchBot3
SearchBot4
AreaBot0: AreaBot
bumpers = bumpers : RangeSensorRel
maxZPos = 8.547
maxXPos = 8.548
maxZNeg = -8.545
maxXNeg = -8.545
width = 8.548
height = 8.547
currentPosition = Coordinate0 : Coordina
currentMissionStatus = AREABOT_ACTIVATED : MissionStatu
area = 73
searchTime = 3500
searchedTime = 666
SearchBot4: SearchBot
SearchBot1: SearchBot
SearchBot2: SearchBot
SearchBot3: SearchBot
SearchBot0: SearchBot
bumpers = bumpers : RangeSensorRel
camera = camera : CameraSensor
cameraImage = NULL
imageWidth = NULL
imageHeight = NULL
currentPosition = Coordinate5 : Coordina
currentPosition = Coordinate1 : Coordina
currentPosition = Coordinate2 : Coordina
currentPosition = Coordinate3 : Coordina
currentPosition = Coordinate4 : Coordina
currentMissionStatus = AREABOT_ACTIVATED : MissionStatu
x = 2.048
z = 7.004
x = -1
z = 3
x = 0
z = -3
x = 1
z = 3
x = 1
z = -3
x = 0
z = 3
state = UNEXPLORED : CoordinateStatus

This object diagram is a representation of a snapshot moment in a typical system execution.

The centralStation, robotFactory and map objects are instantiated. Once RobotFactory creates the AreaBot and SearchBots at their respective positions, robot deployment is passed on to the Central Station.

In the depicted phase, the AreaBot0 is deployed, the MissionStatus is set to AREABOT_ACTIVATED and it traverses the environment while avoiding obstacles, calculating the area for a set amount of searchTime. The SearchBots 0 through 4 await the calculation to finish at their original position, before being deployed by the Central Station to look for color objectives by taking images.

Since the Search Bots have not been deployed yet, their camera images are empty. The Central Station, using the Map object, adds objects of the class Coordinate that correspond to the x and z positions of the robots as well as the CoordinateStatus, to a List.

*Maximum amount of pages: 1*

## 4. Code generation remarks

**Author(s)**: <Ronan Hochart>

The code generation took a few class diagrams to get right. 3 versions of the class diagram were used throughout the process. The first diagram was a very rough outline of the functionality we wanted and after generating and looking at the code certain functions and variables and associations were seen as missing and modified/added for the second version. The second version was therefore almost complete with all the necessary elements being generated. However there were still small issues like lsits being generated as arrays or variables and functions having the wrong return types. These errors were fixed in the third version. This is the version shown in the paper currently.

A lot of the core code has been implemented in the generated code functions to make sure everything links and flows well together. Some functions are still incomplete such as the obstacle avoidance as it still carries some issues that need to be resolved. These will be resolved at a later date. Almost all variables inside each class were set to private causing us to create a lot more functions to allow the retrieval and modification of those variables as compared to our deliverable 1 code.

As most of the core code is written and allows for the simulation to role albiet not well the next step is to optimize the code and allow the designated missions to be completed in a timely manner. The structure of the generated code will not change as it is sufficient for our use case no matter what random element gets added for the next step of the process.

*Maximum amount of pages: max 1*

## 5. Implementation remarks

**Author(s)**: <Mateo Vakili>

The implementation mainly improves on the structure of the system and most importantly the communication between the Agents and the Central Station.

Central Station extends the Subject class to improve on the communication process with the Robots. The Central Station uses its super class Subject to set a mission status. This will trigger the notifyAllObservers() function which will notify all objects that implement the Observer interface. The Observer interface is implemented by Robot and is the super class of both SearchBot and AreaBot, meaning that they will update their mission status whenever notifyAllObservers is called. There are two different commands in which the CentralStation can give as mission states, namely, AREABOT_ACTIVATED and SEARCHBOT_ACTIVATED. In the beginning the Central Station will set the mission status on AREABOT_ACTIVATED when the AreaBot finishes its mission it notifies the CentralStation. This the triggers a change of mission state to SEARCHBOT_ACTIVATED by the Central Station. There are two more mission states that are only used internally by the Robots to avoid obstacles, SEARCHBOT_AVOID_OBSTACLE and AREABOT_AVOID_OBSTACLE, but when the process is done they will switch back to either AREABOT_ACTIVATED or SEARCHBOT_ACTIVATED based on the type of the bot.

For the objective of the system the Coordinate class is introduced in this implementation. The Coordinate class can be used to represent the coordinates in the environment in terms of their state which could be UNEXPLORED, EXPLORED, OBSTACLE, TARGET. As the name implies Target would be the coordinates around the mission targets ( boxes in certain color ). When the SearchBot is activated it will constantly send its coordinates to the central station marking their state as EXPLORED. This will then be stored in the Map class's coordinate list by the central station.

For the SearchBots to be able to identify mission targets both camera and belt sensors are needed. The getColor() function will use the camera to get the color of the object in front of the Robot.  This functionality will be used in the future implementation for deliverable 3. using the latter feature whenever SearchBots find a mission target or an obstacle the will set the state of the explored coordinate accordingly.

Moreover, the CentralStation is not responsible for creating the robots. For this purpose the RobotFactory class is implemented to do the work and return a list containing the bots to the CentralStation.  The Central Station will then use its environment variable ( which is assigned to the current environment being used in main.java ) to spawn the created Robots. It is worth mentioning that the RobotFactory will let the CentralStation decide on how many SearchBots and AreaBots it needs but their is a limit imposed which is 2 for AreaBot and 10 for Search Bots.

**Future Implementations:**

Map has a function called findObjectives() this function will find the number of mission targets based on the states of the coordinates and their distance from each other. This will not work for now as the SearchBots implementation is not complete. The SearchBot will have to use its getColor function when ever it hits an obstacle so that it can identify weather the object it collided with is a target or an obstacle.

Lastely, The implementation of both SearchBot and AreaBot for avoiding obstacles haven't changed and will be done in the future implementation for deliverable 3.