UNIVERSITÀ
DEGLI STUDI
FIRENZE

Scuola di Scienze Matematiche, Fisiche e Naturali
Second Level Degree in Computer Science
Resilient and Secure Cyber Physical Systems
"Cyber-Physical System of Systems"

REPORT OF KILOBOT PROJECT
"WITCH SAYS COLORS"

MATTEO MAURO 7008727

Academic Year 2018-2019

Matteo Mauro 7008727:
Report of Kilobot Project
"Witch says colors", Second Level Degree in Computer Science
Resilient and Secure Cyber Physical Systems
"Cyber-Physical System of Systems" , © Academic Year 2018-2019

INDICE

1

REQUIREMENTS: OVERVIEW OF BLOCKLY4SOS MODEL

ID	Description
	Architecture
1.1	The SoS shall be composed of 5 kilobots and a controller located about 50 cm above the playground
1.2	Every kilobot has its own ID which shall be unique, from 0 to 4
1.3	The kilobots shall operate in a space of 2m x 2m
1.4	The surface shall be a playground smooth and reflecting
1.5	Every ID is associated with a color: 0:=RED, 1:=GREEN, 2:=BLUE, 3:=YELLOW, 4:=PURPLE
1.6	The SoS target shall be simulating the game "Witch says colors" according to the rules described
1.7	At the initial state, the kilobots shall be arranged in a topology representing a connected network: they must be enough close to send each other information, but more than 45 mm (60-70mm is recommended)
1.8	The role of witch is always played by kilobot 0
1.9	Kilobots 1,2,3 and 4 shall be able to be selected either as runner or catcher
1.10	The round of the game always terminates globally after 2 minutes
1.11	Each kilobot shall be able to communicate through infrared
1.12	There shall be only one runner at the time
1.13	When a kilobot terminates locally, it stops moving
1.14	A kilobot shall terminate the catching for 2 reasons: for catching the runner or for the timeout
1.15	Termination of catching means that a catcher stops to move and to estimate the distances from the runner
1.16	The witch shall not be the runner
1.17	A kilobot shall change randomly a direction, the change happens randomly within 8 seconds since the last change

	Communication
2.1	Each kilobot shall be able to communicate with other kilobots through RUMIs
	Dependability
3.1	A person shall prevent kilobot to move far away from playground boundaries
3.2	A kilobot shall prevent a collision with other kilobots
3.3	A kilobot shall stop moving if the distance from another one is too close
	Dinamicity
4.1	A kilobot shall detect the presence of other kilobots in order to avoid collisions
4.2	A catcher shall detect the distance from the runner and try to follow its direction
4.3	A kilobot shall perform random movements in order to find the runner
	Time
5.1	There shall be no mechanism of synchronization
5.2	A kilobot shall use its own local clock to coordinate its activities

1.1 HIGH LEVEL DESCRIPTION OF THE GAME "WITCH SAYS COLORS"

The game is simulated using 5 kilobots, each of them associated with a unique ID from 0 to 4. Every ID is associated with a color as expressed in the requirement 1.5.
The initial configuration of the SoS shall expect the kilobots to be placed in a connected topology, that is they have to create an ad-hoc network in which every bot is reachable from any other by communication through messages. The kilobots are logically classified in one **witch**, one **runner** and three **catchers**. Every role behaves in a different way:

1. Witch: is the kilobot with ID equals to 0, its purpose is to choose a random number from 1 to 4 and to send it to neighbours (CHO-

SEN_COLOR message); it cannot choose its color, but it will take part to the catching (i.e. it's a catcher);

2. Runner: is the kilobot which receives the message containing its own ID, it periodically signals its position through a message (IM_HERE message) and randomly moves around the playground (both directions and delays are calculated randomly, according to requirement 1.17); it can be recognized by the blinking (1 second colored, 1 second white);

3. Catcher: its main objective is to capture the runner, detecting and estimating the distance from it; in order to catch it successfully, a catcher must estimate twice in a row a distance less than 55 mm: in this case it stops moving and starts to transmit a message (GOTCHA message) in order to notify every other catchers and runner around that the game is terminated.

The termination of the round is local for a kilobot when:

1. it catches the runner and starts to transmit the GOTCHA message;

2. it receives a GOTCHA message, so it stops and forwards the same;

3. timeout of 120 seconds is reached (it doesn't transmit anything).

In every case, the kilobot starts to blink quickly to inform the users about the state of termination. Since the range of transmission is limited, a kilobot may walk away too far from the others and never be able to detect a message: to allow a termination of the game, every kilobot counts the time elapsed since the catch was started and periodically check the timeout of 120 seconds (for the runner it begins when it receives the CHOSEN_COLOR message, for the others is when they receive the IM_HERE message, in this way they all start with a precision of 1 second approximately).

In order to avoid a collision, every catcher transmits periodically an AVOID message: a catcher also listens for this kind of message and estimates the distance from an other kilobot. If the distance is less than 45 mm; then the bot stops moving, in order to prevent any movement (in this case they are almost touching each other). In the last case, the catcher continues to signal its position and to listen for the presence of the runner, so it's still operating.

1.2 BLOCKLY4SOS MODEL OVERVIEW

The SoS is represented as "Witch_Says_Colors" SoS block composed by 3 catchers, 1 witch and 1 runner. The emergent behavior is the simulation of the game (as described in the previous section), while a detrimental but expected behavior is the lack of sufficient range transmission and unreliability of the communications, that may lead to lost or corruption of messages as the impossibility to detect messages sent by others. The block is described in Figure 1.

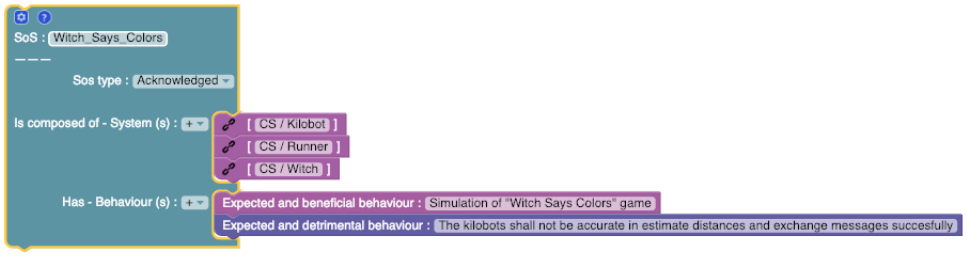

Figure 1: Witch_Says_Colors SoS block

The kilobots are classified in three different roles: Witch, Runner and Kilobot.

The Witch (Figure 2) provides the following services:

1. chosen_color: choose randomly a color and send it to neighbours through a CHOSEN_COLOR message;
2. forward_im_here: forward messages of type IM_HERE, these messages are forwarded for at most one second the first time that an IM_HERE message is received, in order to inform every other kilobot that the runner has started to move;
3. avoid_collision: periodically sends a message AVOID to signal its own presence on the playground;
4. gotcha: if the catcher gets closer enough to the runner, it stops moving and starts to send GOTCHA message to notify the successfully catching;
5. forward_gotcha: if a GOTCHA message is received, the kilobot stops to move and starts to forward this message until timeout is reached, in order to notify that the runner has been caught.

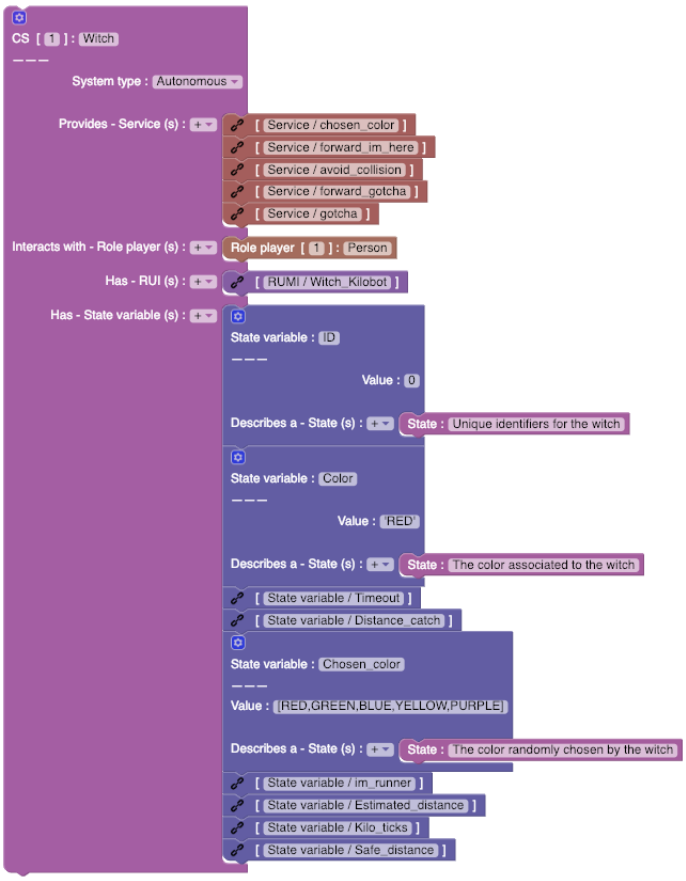

Figure 2: Witch block

The Runner (Figure 3) provides the following services:

1. im_here: periodically signals its position in order to inform the catchers about its presence;
2. forward_gotcha: already described in Witch section.

A generic Kilobot (Figure 4) provides the following services:

1. forward_chosen_color: in order to propagate the chosen color of the witch, every kilobot shall forward the message to its neighbours;
2. forward_im_here: already described in Witch section;
3. avoid_collision: already described in Witch section;
4. forward_gotcha: already described in Witch section;
5. gotcha: already described in Witch section;

Essentialy a Witch class is similar to the Kilobot class (in fact the Witch shall take part to the catching), except for the service forward_chosen_color (the Witch generates the chosen color, every other just forwards it), moreover the separation makes more clear the interactions in the sequence diagrams.

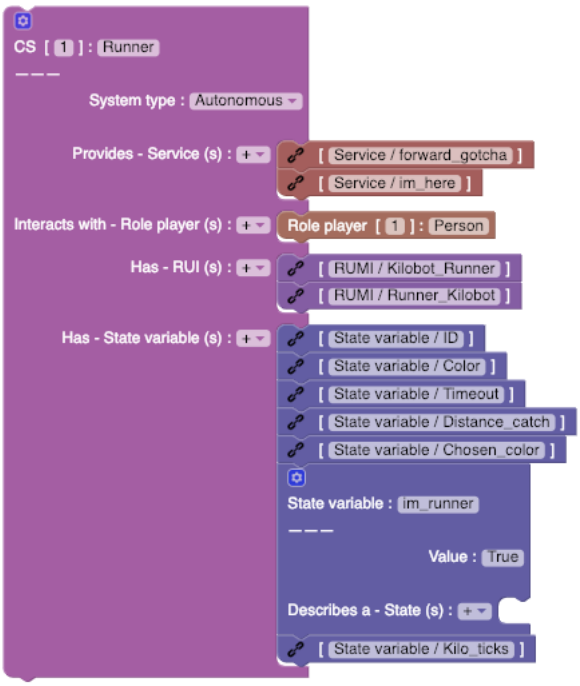

Figure 3: Runner block

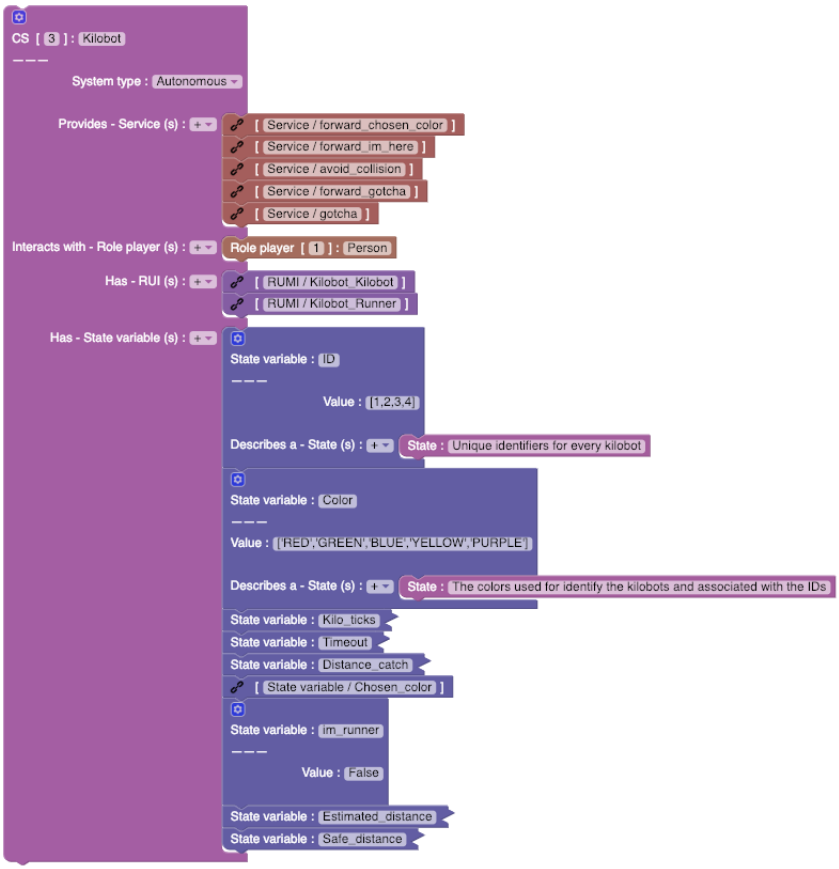

Figure 4: Kilobot block

The RUMIs (according to the example of "SmartGrid - Subset - with simulation" provided in the Blockly4SoS environment) have been grouped and classified regarding to the Source and Destination entities (Figure 5). This classification allows to identify immediately which entities are involved in the services provided by a RUMI:

1. RUMI Witch_Kilobot: the witch is expected to communicate the chosen color to the kilobots;

2. RUMI Kilobot_Kilobot: a kilobot is expected to temporarily forward the CHOSEN_COLOR and the IM_HERE messages at the beginning of the round, it shall also periodically send the AVOID messages and forward the GOTCHA messages to all other kilobots;

3. RUMI Runner_Kilobot: the runner is expected to send and IM_HERE message to signal its position to the kilobots;

4. RUMI Kilobot_Runner: a kilobot is expected to send a GOTCHA message whenever it estimates a sufficient distance of capture.

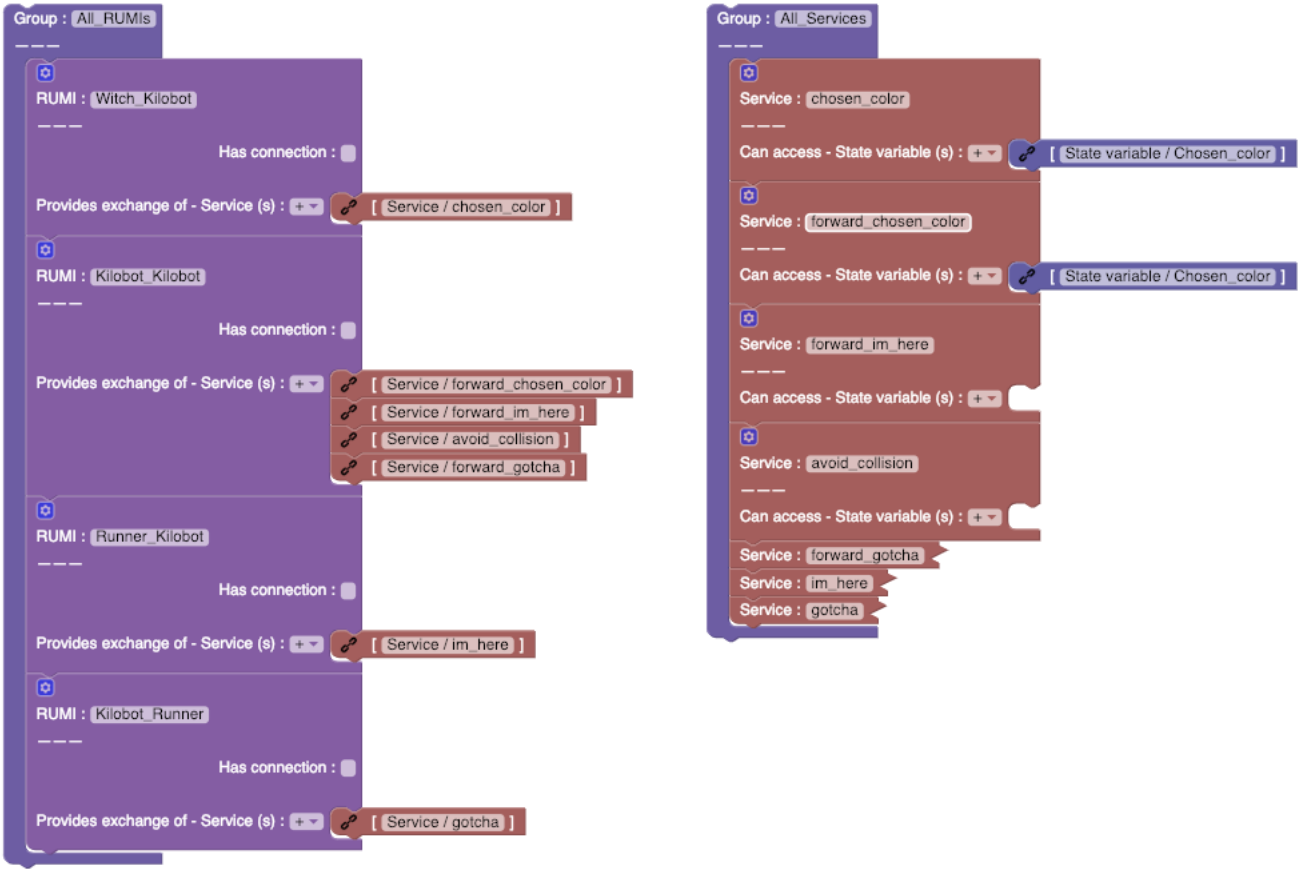

Figure 5: RUMIs block

2

SEQUENCE DIAGRAMS SIMULATIONS

In the following section, there are showed two possible interactions among the kilobots: a simulation of a successfully catching and one of a prevention of collisions. These two diagrams represent the expected emergent behaviors during the simulation of the game.

2.1 SUCCESSFULLY CATCHING SIMULATION

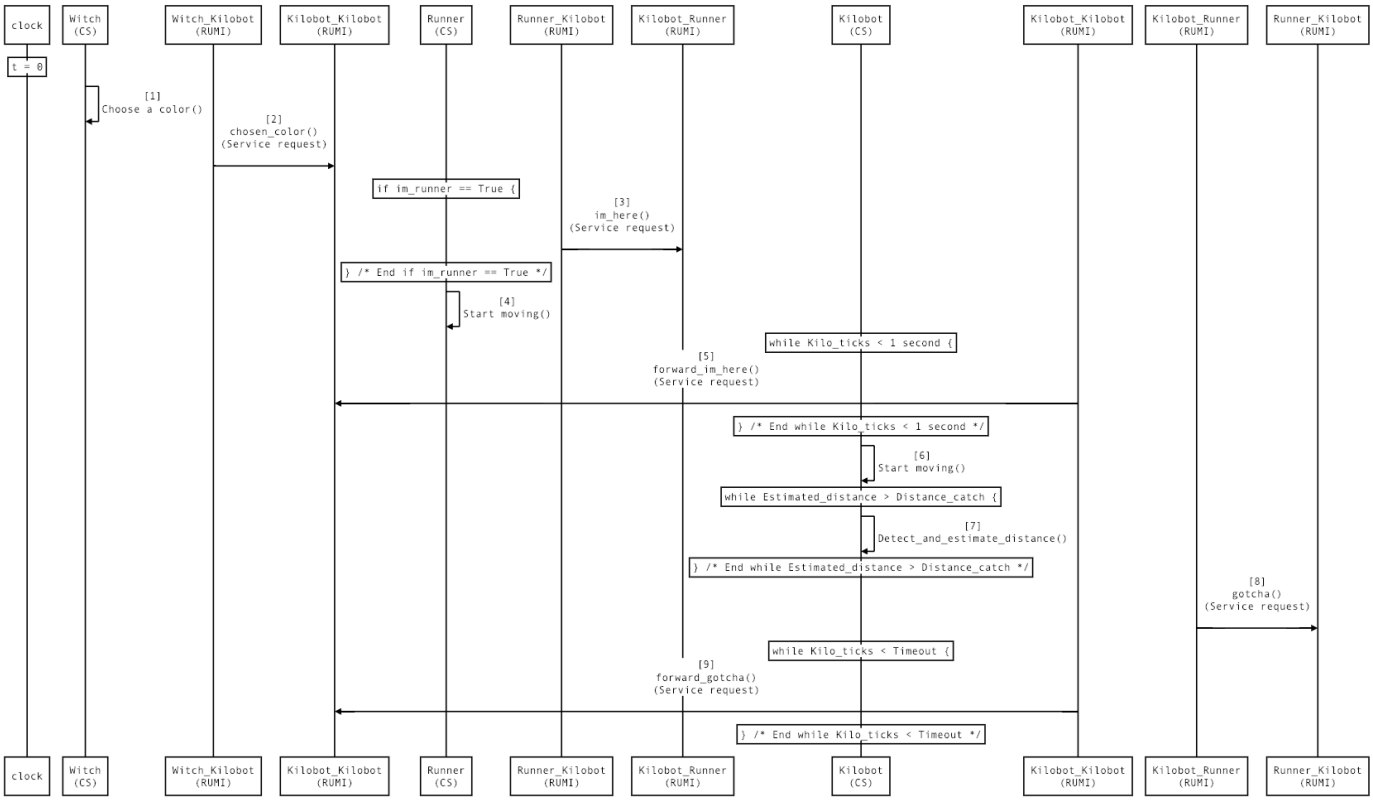

Figure 6: successfully catching sequence diagram

1. The witch chooses a random color;
2. Through the RUMI Witch_Kilobot, the chosen color is sent to all nearly kilobots;
3. If the kilobot identify itself as the runner, it starts to send messages IM_HERE through its RUMI Runner_Kilobot;
4. The runner starts to move and starts the timeout;
5. Every kilobot forwards the IM_HERE message for at most one second to all nearly kilobots;
6. Every catcher starts to move and starts the timeout;
7. During the catching, a kilobot detects any message and estimates the distances from the runner;
8. The catcher estimates a sufficient distance, so it sends a GOTCHA message;
9. Every kilobot (including the runner) that receives the GOTCHA message, starts to forward the same.

2.2 AVOID COLLISIONS SIMULATION

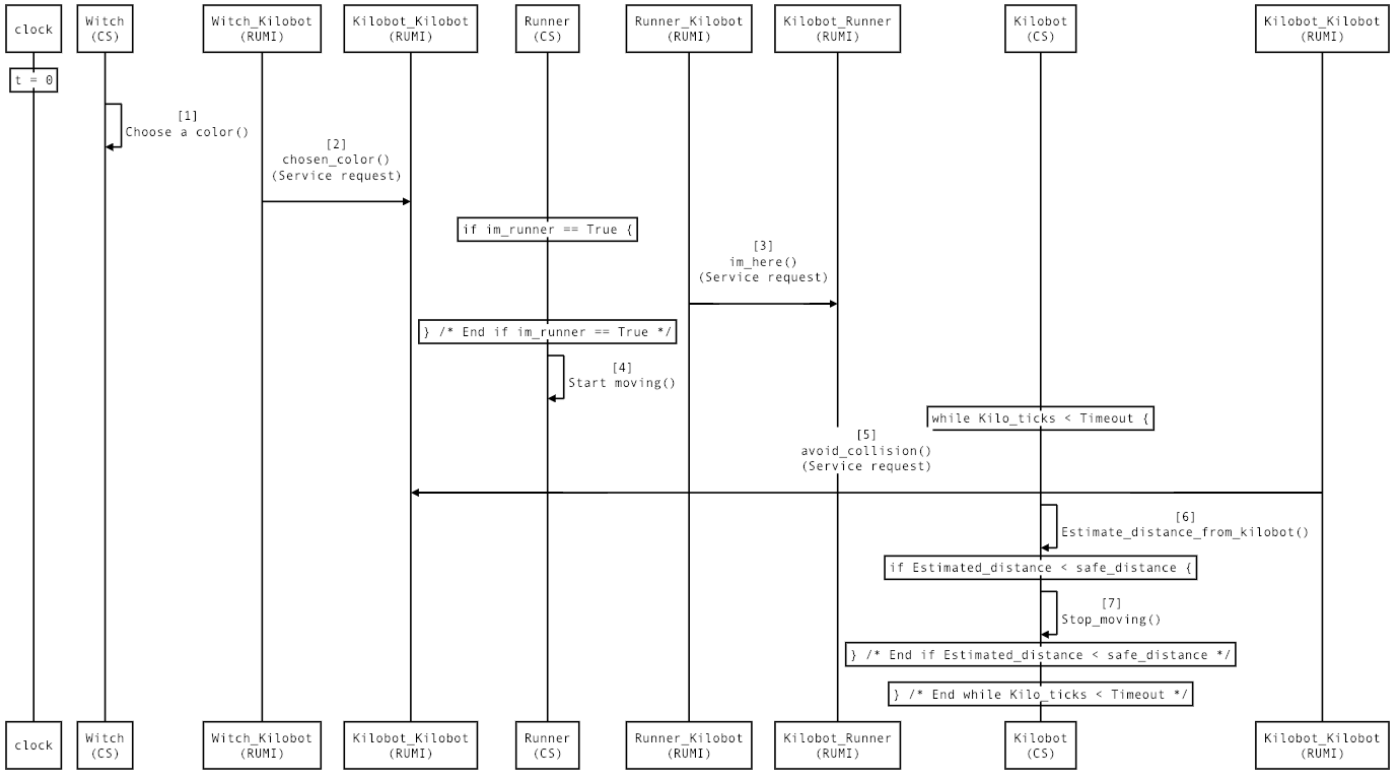

Figure 7: avoid collision sequence diagram

Point 1,2 and 3 are identical to the previous diagram.

4. while the timeout is not over, a kilobot periodically sends an AVOID message to signal its presence;

5. for every AVOID message received, a kilobot estimates the distance from the sender;

6. if the distance is less than 45 mm (specified in safe_distance), then the kilobot stops to move, because in that state the kilobots are basically touching each other and any movement can be dangerous.

3

IMPLEMENTATION OF THE SOS

3.1 loop() function explanation

The function loop() is separated in 4 different activities performed periodically by every kilobot:

1. check the timeout: the kilobot checks if the timeout has been reached since the starting of the timer, in the affirmative case the kilobot stops to send or receive anything and call the game_over() function;
2. forward IM_HERE message: whenever a kilobot receives for the first time the message IM_HERE, it means that the runner has just started to run away; in order to activate every other catcher, for a brief interval of time (1 second) every kilobot has to forward the message IM_HERE (this is set on the message_rx() function), then in the loop() the message is changed in AVOID when the time is elapsed;
3. blink section: the runner blinks for 0.5 second of white and 0.5 second colored, the witch blinks 1 second white and 2 second red, finally whenever the timeout is triggered every kilobot blinks approximately for 0.25 second white and 0.25 colored.
4. change direction: periodically change randomly the direction of the kilobot, the interval of time before another change is randomly chosen but always less than 8 seconds.

Essentialy, the loop() is used to perform all the timed activities.

3.2 capture estimation

In order to simulate a successfully catching more than a slight touch, a catcher must estimate twice in a row a distance less than 55 mm from

the runner: this is performed in the function check_capture() using two support variables updated whenever a message IM_HERE is received. The function check_capture() returns 1 (true) if the distances are both less than 55, otherwise 0 (false). There is also an attempt to change direction, in order to get closer to the runner, when the final distance estimated is greater than the second-last: in that case it means that the catcher is running away from the target (the direction is changed randomly and the operation is not repeated before 3 seconds).

3.3 HOW TO RUN THE SIMULATION WITH KILOMBO

Two initial configurations are provided: one with the kilobots placed in a ring and one in sequence.

Figure 8: sequence placement

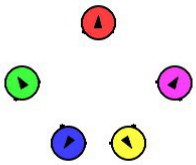

Figure 9: ring placement

In order to run the simulation (the following was performed on Ubuntu 18.04):

1. open the terminal;
2. navigate to project folder;
3. call `make` to compile;
4. execute `./witch_says -b [bot_pos_list.json | bot_pos_ring.json]` (choose one between list or ring configuration).

The main files that must be present on the Kilombo project folder are: witch_says.c, witch_says.h, utility.c, utility.h, Makefile, bot_pos_list.json, bot_pos_ring.json.

witch_says.c contains all the relevant code, including the loop() and main() functions.
witch_says.h contains the data structures and definitions of all the variables used by the kilobots.
utility.c contains the method set_motion(), useful for setting a smooth movement also for the real kilobots.

ATTENTION: the code for the real kilobots is implemented in the file "main_for_kilobots.c", the .hex file is already present (compiled with the following editor -> https://www.kilobotics.com/editor).