# CS201 – Spring 2017-2018 - Sabancı University

# Homework #5: A Robot Game

# Due April 24, Tuesday, 22:00

## Motivation

The aim of this homework is to practice on member functions and classes using the already known Robot class. Of course, you will need to use the topics that we have seen before such as if statements, functions and loops as required. For this homework, you should submit all of your files to SUCourse (your main cpp file, Robots_Modified.h, Robots_Modified.cpp, minifw.h, minifw.cpp, randgen.h, randgen.cpp and any other files in your solution) **without zipping them**. Name of your main cpp file should be in the expected format ("sucourseusername_lastname_name_hwnumber.cpp", check the submission procedure of the homework).

## Description

In this homework, you will write a game program, implementing an interactive game using the Robot class.

**Robot class:** In the zip package that you obtained this document, you can find the modified versions of Robot class files (Robots_Modified.cpp, Robots_Modified.h, MiniFW_Modified.cpp and MiniFW_Modified.h). You have to use these files in your project and program. Besides writing the program for the game in a cpp file (your main cpp file), you will also make some further modifications to the Robot class, so you are going to update Robots_Modified.cpp, Robots_Modified.h files. These modifications consist of adding some more member functions. The game will be implemented in another cpp (your main cpp) file that you will write from scratch, not in robot class files. Robot class files are to be updated only to add new member functions.

**Game:** In the game that you will develop, there is a player robot (manually controlled by the user of your program), and four monster robots present. The world is an area of 10-by-15, which has outside borders and some inner walls. There are two different kinds of movements for monster robots. 2 monster robots move in the area with a vertical route, and the other two have a clockwise route. The details of monster movements will be explained later in this document. The aim of the player is starting from (0, 4), to reach the target cell (14, 4) with gathering 50 things from the area. Please see the following Game Rules section for more details.

Please read the following explanations very carefully, because they are crucial for developing your program and modifying the header and source files of the robot class.

## IMPORTANT!

If your code does not compile, you will get **zero**. Please be careful about this and double check your code before submission.

# Game Rules and Details

## Environment

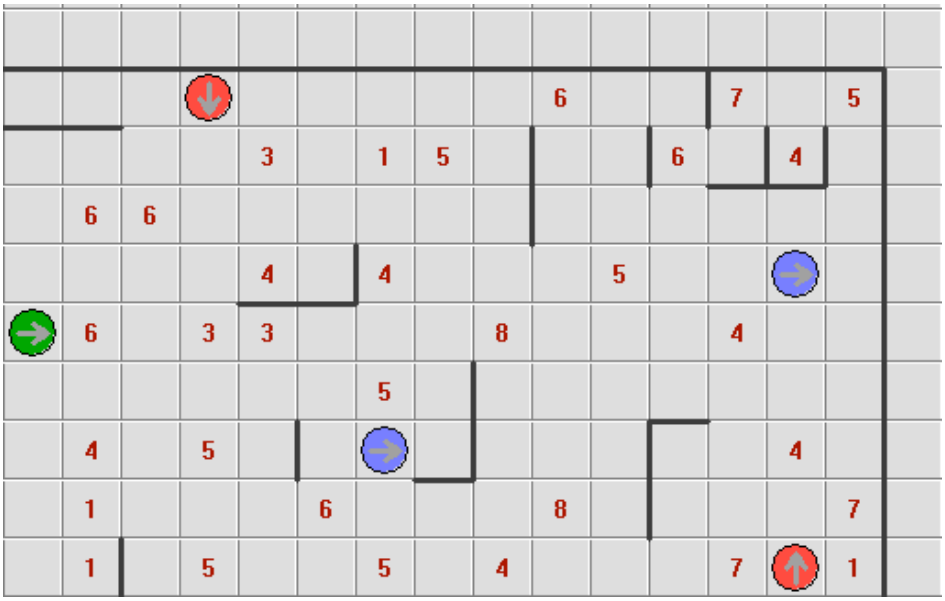
Your program will use an environment that consists of a rectangular area of size 10x15. The corner points of the environment are (0, 0), (0, 9), (14, 0), and (14, 9). The boundaries of this environment are all walls (south and west are world boundaries; north and east are manually set walls). Moreover, there might be horizontal and vertical walls in the environment as well.

You will use an environment file while running your program, this file will contain all the walls (borders and inner walls) and things placed in the area.

The player robot's and monster robots' starting positions are as the following:

- Player robot
  - Starting cell: (0, 4)
  - Direction: east
  - Color: green
- Monster1
  - Starting cell: (3, 8)
  - Direction: south
  - Color: red
- Monster2
  - Starting cell: (6, 2)
  - Direction: east
  - Color: blue
- Monster3
  - Starting cell: (13, 5)
  - Direction: east
  - Color: blue
- Monster4
  - Starting cell: (13, 0)
  - Direction: north
  - Color: red

Below you may see the start of the game:



## The Player Robot

The player robot is actually an ordinary robot (an object from Robot class). As described above, the player robot should be created at position (0, 4) and facing east. Initially, its bag content is 0. Its color must be set to green.

The player robot has 3 lives in the beginning. The game ends if the life count reaches 0.

The user will control the player robot using ***up***, ***down*** and ***right*** arrow keys on the keyboard. The control for the arrow keys can be done very easily using ***IsPressed*** function that you will see in recitations this week. Whenever one of the arrow keys is pressed, the player robot will move one step according to the pressed key. For example, if the user presses ↑ key, then the player robot will move one step (cell) towards north. Please note that, in this game, you will not use ***left*** arrow key, and the player robot can not go to west. This is intended to create a game feeling.

- If there is a wall that blocks the movement, then the player robot must die and if has any lives left should be resurrected and life count should be decreased by 1. The game ends if the life count reaches 0.
- If player robot bumps into a monster robot, then the player robot must die and if has any lives left should be resurrected and life count should be decreased by 1. The game ends if the life count reaches 0. You should also resurrect the monster robot.

Whenever the player robot moves into a cell that contains thing(s), it has to pick all the things in that cell after the movement.

**STOP action:** If the user presses SPACE key on the keyboard, it invokes a special action in the game. When SPACE key is pressed, if the player robot does not have at least 20 things in its bag, nothing happens. But if the player robot has equal to or more than 20 things in its bag, then monster robots are stopped, and 20 things are deduced from the player robot's bag. In other words, the user can pay 20 things to stop monsters. This action has no return, once the monsters are stopped moving, they will not continue to move in any case in that game. After invoking this, pressing SPACE key again will not do any more actions.

## The Monster Robots

As described above, there will be four monster robots in the game. Their starting positions are set. There are two different robot movements:

**Vertical movement:** Monster1 and Monster4 should move in a vertical manner. They will begin to move to the direction they are facing in the beginning. Whenever they are facing a wall, instead of bumping into the wall and dying, they should turn back and move to the opposite direction. They will continue this movement until the end of the game. (or until the user invokes STOP action)

**Clockwise movement:** Monster2 and Monster3 should move in a clockwise manner. They will begin to move to the direction they are facing in the beginning. Whenever they are facing a wall, instead of bumping into the wall and dying, they should turn right and move to that direction. They will continue this movement until the end of the game. (or until the user invokes STOP action)

***Hint:*** You may use ***FacingWall*** member function of the Robot class to check if a robot is facing a wall.

As described above, the program flow is a sequence of player and monster movements. At each iteration of this program flow, only one monster will have a chance to move. This monster robot will be selected by random number generation, which means every time randomly one of the four robots

will be selected to be moved. You can either do this selection in a function or directly in main. You will use the ***RandGen*** class to select a random robot.

Monster robots may only die by bumping into the player robot or another monster robot. As mentioned before, if the player has any more lives left both robots should be resurrected. If two monster robots crashed, again you should resurrect them both. Unlike the player robot, monster robots have unlimited lives. So, you do not need to keep track of how many times they died.

Please try the sample homework that is included with this document.

## End Conditions

- The player robot runs out of lives. The message should be:

  *"You run out of lives… You lost the Game!"*

- The player robot reaches the target cell with equal to or more than 50 things in its bag. The message should be:

  *"Congratulations, you win!"*

- The player robot reaches the target cell but with less than 50 things in its bag. The message should be:

  *"You reached the end but could not gather enough things… You lost the Game!"*

- The user invokes STOP action by pressing the SPACE key when one of the monster robots present in the target cell. The message should be:

  *"You blocked the target… You lost the Game!"*

## Use of Functions and Classes, etc. (READ THIS. VERY VERY IMPORTANT)

For this homework, you have to add some new member functions to the Robot class. The number of additional functions can be different up to your design, but you have to add at least the following 9 member functions:

1. **GetXCoordinate**: A member function that returns the X coordinate of the robot object.
2. **GetYCoordinate**: A member function that returns the Y coordinate of the robot object.
3. **TurnBack**: A member function that turns the robot object to the opposite direction it is facing.
4. **TurnFace**: A member function that takes a Direction type (this is an enumerated type defined in robot header file) parameter and turns the robot object towards this direction.
5. **GetBagCount**: A member function that returns the number of things in the robot's bag.
6. **SetBagCount**: A member function that sets the number of things in the robot's bag. This function will be useful to deduce 20 things from the player robot's bag when STOP action is invoked.
7. **IsAlive**: A member function to check if the robot object is alive.
8. **Resurrect**: A member function to resurrect the robot object. Hint: play with ***stalled*** private data member for the implementation.
9. **PickAllThings**: A member function that allows the robot object to pick all the things in the occupied cell.

Needless to say, you have to utilize these member functions in your program, as needed.

**sucourseusername_lastname_name_hwnumber.cpp**, this file will include the game loop, robot creation and any other user defined functions that will be needed to implement the game.

In your program, please do not employ variables in order to keep track of a characteristic (private data member) of a robot object. This is against the philosophy behind OO (Object Oriented) programming. Whenever you need to access or change these private data members, use member functions. For example, do not employ a variable in the program to keep track of the position of a robot. Use member functions for this purpose.

In the member function implementations that change the visual characteristic of a robot, you need to use the command: `theRobotWindow->Redraw(this);` at the end of the member function implementation (do not use this anywhere else in the program).

If you need some other member functions, you are free to design and implement. However, do not forget the OO programming philosophy that proposes to have member functions for general class use. If a member function is too specific for a particular application, it should not be written as a member function, it might be a free function in the program. Again, according to OO programming principles, member functions should do a single job, rather than multiple jobs. Please remember this also while writing your member functions. All of these OO programming rules will be considered in the grading process.

Moreover, your program must be modular, and you should avoid code duplication. Thus, you have to show your ability to use functions in an appropriate way. This will affect your grade. In general, if your main function or any user-defined function is too long and if you do everything in main or in another user-defined function, you grade may be lowered.

**AND PLEASE DO NOT WRITE EVERYTHING IN MAIN AND THEN TRY TO SPLIT THE TASK INTO SOME FUNCTIONS JUST TO HAVE SOME FUNCTIONS OTHER THAN MAIN. THIS IS TOTALLY AGAINST THE IDEA OF FUNCTIONAL DESIGN AND NOTHING BUT A TRICK TO GET SOME POINTS. INSTEAD PLEASE DESIGN YOUR PROGRAM BY CONSIDERING NECESSARY FUNCTIONS AT THE BEGINNING.**

Try to use parametric and non-void functions wherever appropriate. Do NOT use any global variables (variables defined outside the functions) to avoid parameter use.

In this homework (and in the coming ones) you are not allowed to use instructions such as “exit” and “goto”. These cause difficulties to control the flow of your programs. Thus, we do not approve using them.

## Other Important Details (READ THIS. VERY IMPORTANT)

In the zip package, we provided you the necessary class files. Robots_Modified.h and Robots_Modified.cpp are original files for the Robot class. You will add the member functions mentioned before, and you will do the necessary modifications in the Robots_Modified.h, Robots_Modified.cpp files. In Robots_Modified.h file, you have to add the prototypes of the new member functions. Please add these prototypes to the end of the "public" section of the class definition. In the Robots_Modified.cpp file, you are going to write the implementations of the new member functions. You have to write these implementations between the following comment lines that are already provided in the file:

```
/*************** START: CS201 students' hw5 member functions ******************/




/*************** END: CS201 students' hw5 member functions ******************/
```

MiniFW_Modified.h and MiniFW_Modified.cpp files do not need to be updated.

For random number generation try to use "RandGen" class (an example will be given in recitations). In general, in order to use "RandGen" class, you have to include "randgen.h" in your program and add "randgen.cpp" to your project.

In order to check if an arrow key is pressed or not, you can use the function called **`IsPressed`**. This function takes a parameter, which is the name of the arrow key, and returns true if currently that key is pressed. The names of arrow keys are **`keyUpArrow`**, **`keyDownArrow`**, **`keyLeftArrow`**, and **`keyRightArrow`**, and the name of the space key is **`keySpace`**. This function is defined in minifw_modified.h, so you have to include this file at the beginning of your program. An example use of **`IsPressed`** will be explained in recitations.

# No abrupt program termination please!

You may want to stop the execution of the program at a specific place in the program. Although there are ways of doing this in C++, it is not a good programming practice to abruptly stop the execution in the middle of the program. Therefore, your program flow should continue until the end of the main function and finish there.

## General Rules and Guidelines about Homeworks

The following rules and guidelines will be applicable to all homeworks, unless otherwise noted.

### How to get help?

You may ask questions to TAs (Teaching Assistants) of CS201. Office hours of TAs are at the class website. Recitations will partially be dedicated to clarifying the issues related to homework, so it is to your benefit to attend recitations.

### What and Where to Submit

Please see the detailed instructions below/in the next page. The submission steps will get natural/easy for later homeworks.

### Grading and Objections

Grading:

- Late penalty is 10% off of the full grade and only one late day is allowed.
- **Having a correct program is necessary, but not sufficient to get the full grade. Comments, indentation, meaningful and understandable identifier names, informative introduction and prompts, and especially proper use of required functions, unnecessarily long program (which is bad) and unnecessary code duplications (which is also bad) will also affect your grade.**
- Please submit your own work only (even if it is not working). It is really easy to find out "similar" programs!
- For detailed rules and course policy on plagiarism, please check out http://myweb.sabanciuniv.edu/gulsend/su_current_courses/cs-201-spring-2008/plagiarism/ and keep in mind that

# Plagiarism will not be tolerated!

Grade announcements: Grades will be posted in SUCourse, and you will get an Announcement at the same time. You will find the grading policy and test cases in that announcement.

Grade objections: It is your right to object to your grade if you think there is a problem, but before making an objection please try the steps below and if you still think there is a problem, contact the TA that graded your homework from the email address provided in the announcement.

- Check the comment section in the homework tab to see the problem with your homework.
- Download the files you submitted to SUCourse and try to compile it.
- Check the test cases in the announcement and try them with your code.
- Compare your results with the given results in the announcement.

## What and where to submit (IMPORTANT)

Submissions guidelines are below. Students are expected to strictly follow these guidelines in order to have a smooth grading process. If you do not follow these guidelines, depending on the severity of the problem created during the grading process, 5 or more penalty points are to be deducted from the grade.

Add your name to the program: It is a good practice to write your name and last name somewhere in the beginning program (as a comment line of course). Do not use Turkish characters anywhere in your program (not even in the comments).

Name your submission file:

- Use only English alphabet lowercase letters, digits or underscore in the file names. Do not use blank, Turkish characters or any other special symbols or characters.
- Name your cpp file that contains your program as follows.

"**sucourseusername_lastname_name_hwnumber.cpp**"

- Your SUCourse user name is actually your SUNet user name which is used for checking sabanciuniv e-mails. Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SUCourse user name is cago, name is Çağlayan, and last name is Özbugsızkodyazaroğlu, then the file name must be:

  **cago_ozbugsizkodyazaroglu_caglayan_hw5.cpp**

- Do not add any other character or phrase to the file name.
- Make sure that this file is the latest version of your homework program.
- You have to submit all files that are used in your homework (cago_ozbugsizkodyazaroglu_caglayan_hw5.cpp, Robots_Modified.h, Robots_modified.cpp, MiniFW_Modified.h, MiniFW_Modified.cpp, randgen.h, randgen.cpp) **without zipping them**.

Submission:

- Submit via SUCourse ONLY! You will receive no credits if you submit by other means (e-mail, paper, etc.).
  1) Click on "Assignments" at CS201 SUCourse (not the CS201 web site).
  2) Click Homework 5 in the assignments list.
  3) Click on "Add Attachments" button.
  4) Click on "Browse" button and select all file used in homework.
  5) Now, you have to see your file(s) in the "Items to attach" list.
  6) Click on "Continue" button.
  7) Click on "Submit" button. We cannot see your homework if you do not perform this step even if you upload your file.

Resubmission:

- After submission, you will be able to take your homework back and resubmit. In order to resubmit, follow the following steps.
  1) Click on "Assignments" at CS201 SUCourse.
  2) Click Homework 5 in the assignments list.
  3) Click on "Re-submit" button.
  4) Click on "Add/remove Attachments" button
  5) Remove the existing files by clicking on "remove" link. This step is very important. If you do not delete the old files, we receive both files and the old one may be graded.
  6) Click on "Browse" button and select the new files that you want to resubmit.
  7) Now, you have to see your new files file in the "Items to attach" list.
  8) Click on "Continue" button.
  9) Click on "Submit" button. We cannot see your homework if you do not perform this step even if you upload your file.

**Successful submission is one of the requirements of the homework. If, for some reason, you cannot successfully submit your homework and we cannot grade it, your grade will be 0.**

***Good Luck!***
***Ece Egemen, İnanç Arın and Gülşen Demiröz***