**CS201 – Fall 2019 - Sabancı University**

**Homework #6 – A Pacman Game**

**Due: December 18, 2019, Wednesday, 22:00**

# **Brief Description**

In this homework, you will write a program to implement an interactive game using the Robot class. In the google drive 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, 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 to the robot class as will be explained later in this document. The game will be implemented in another 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.

We will be manually grading your homework without GradeChecker. To be able to build your program, you should submit **ALL** of your files (Robots\_Modified.cpp, Robots\_Modified.h, MiniFW\_Modified.cpp, MiniFW\_Modified.h, strutils.cpp and strutils.h) to SUCourse **without zipping** them. **The name of your main source (cpp) file should be in the expected format:** "sucourseusername\_lastname\_name\_hwnumber.cpp" (all lowercase letters). Please check the submission procedures of the homework in this document.

**To get help you may ask questions to the list of TAs/LAs:** [**cs201gchelp@lists.sabanciuniv.edu**](mailto:cs201gchelp@lists.sabanciuniv.edu)

In the game that you will develop, there is a *player* robot (manually controlled by the user of your program), and three *monster* robots (moves automatically by your program). The aim of the player robot is to win the game by collecting more things than monster robots collect. If one of the monsters collect more or equal amount of things, then that monster wins the game.

The environment of the game is an 8-by-8 area (see details below). The monsters and the player robot collect "things" available in the environment as they move. You (i.e. the player robot) have 3 lives. The game finishes either all the things in the environment are collected by the player and monster robots, or the player robot uses up all of its three lives. If all of the lives of player robot finish, then you lose the game. If the game finishes due to the fact that all the things in the environment are collected, then the winner is determined by the number of things collected by the player and monster robots. Please see the details in upcoming sections.

The robots (the player and the monsters) may die during the game. The rules of the game about dying and resurrection will be explained later in this document. Moreover, the mechanisms and rules about moving the player and the monsters will be explained later.

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

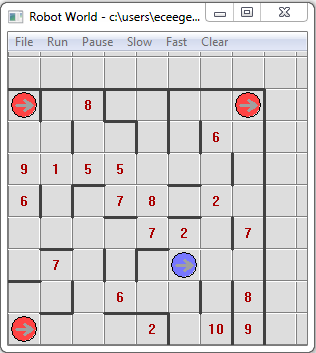
**Game Rules and Details**

###### **Environment**

Environment of the game is an 8-by-8 area. The corner points of the environment are (0,0), (7,0), (0,7), and (7,7). 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.

The things on the robot world within the boundaries must be planted randomly by your program. At least 20 cells must be populated with things between 1 and 10. You may choose to populate more cells with things, but **none** of them should have more than 10 things.

The player robot will start in the (5, 2) cell; monster robots will start in (0, 0), (0, 7) and (7, 7) cells.



**The Player Robot**

It is actually an ordinary robot (an object from Robot class). As shown in the figure above, the player robot is to be created at position (5, 2) and facing east. Initially its bag content is 0. Its color must be set to blue or green. Its name will be taken as input from the user.

The user of your program controls the player robot using up, down, right, left 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. 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. If there is a monster or a wall that blocks the movement, then the player robot, of course, dies.

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. In order to pick all things in a cell, you will develop and use a new member function. Moreover, please make sure that the player robot is alive before picking all the things in a cell, because if the player robot becomes dead after the movement (due to hitting a wall or a monster), picking all the things while dead may cause an infinite loop in your program.

The player robot may be killed by a monster if this monster hits into the player robot. The player robot may also die when it hits into a wall, world boundary or one of the monster robots.[[1]](#footnote-0) The player robot has a total of three (3) lives in this game. If the player is dead, then the life count must be decremented by one and a message that shows the remaining number of lives must be displayed. After that, the dead robot is resurrected at the same position it died. You are going to implement a member function for resurrection.

When the player robot hits one of the monsters, not only the player robot, but also the monster robot dies. There is a procedure to handle dead monsters. The details of handling dead monsters are explained below in Section "The Monster Robots".

**Flow of the Game**

Your program will begin with asking the names of the robots to the user. Name of the player robot should be asked first, and then the program should ask for monsters’ names. After getting the inputs, you may proceed to the general flow of the game.

The general flow of the game is that monsters and player robots are to be moved in an alternating manner until the game finishes. That means, in a loop do the following:

1. One of the monster robots will move randomly (see below for the detailed rules for monster movements). After the movement, if the player and/or monster robots become dead, you have to follow the dead robot procedures explained in the previous section (for player) and in the following section (for monsters).
2. If the previous monster move did not cause using up all the lives of the player robot, then you will check if one of the arrow keys is pressed to move the player. If one of these keys is pressed, then the player robot will move accordingly. After the movement, if the player robot and/or monsters become dead, you have to follow the dead robot procedures explained in the previous section (for player) and in the following section (for monster).

This loop will continue until the game ends. At the end of the game you will display an appropriate message that includes the name of the winner robot.

As mentioned above, after the movement of each robot (monsters or player), you have to check all robots in the game to see whether they are dead or alive. The reason is that when a robot hits another robot, both die.

When robots are moving too fast you can insert **Sleep(100);** into your code as you need.

Sometimes the user of your program may not press an arrow key so that the player robot stands still. However, monsters should continue to move in the loop, even if player robot does not move.

**The Monster Robots**

There will be three monster robots in the game. These monsters are to be created as objects of Robot class. One of the monsters will be created at cell (0, 0) another one will be created at cell (0, 7); and the last one will be created at cell (7, 7). Monster robots must be red colored. Initially, there will be 0 things in the monsters’ bags. Their names will be taken as inputs from the user.

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. The one that attempts to move in one iteration of the program flow will not do so in the next iteration. In the next iteration the other monster will attempt to move. In this way, an alternating sequence is obtained.

The movements of the monsters are automatic and random. You are going to write a function to turn a robot object towards a random direction and move the robot towards that direction if there is no wall. After the random turn, the monster robot will attempt to move one step at the facing direction by calling the **Move()** member function if not blocked by a wall. After this move, the monster either moves to the facing cell without a problem, or it hits another robot and dies (yes, the monsters are robots, but not intelligent ones ☺).

If the monster is still alive after this move, then the monster picks all the things in the newly moved cell. **Caution:** Please make sure that the monster robot is alive before picking all the things, because otherwise picking all the things while the robot is dead may cause an infinite loop in your program.

If the monster becomes dead after this attempt of move, you should resurrect it. Moreover, it is possible to have another dead robot (the other monster or the player robot) if the monster hit it. If the other monster was killed in this way, it should be resurrected as well. If the player robot was dead due to being hit by a monster, you should decrement the life count by one and display a message before resurrecting it.

The abovementioned actions to be taken for the dead monster also apply when the monster dies due to being hit by the player robot.

Monsters are immortal such that they have infinite "number of lives".

**Ending the Game**

The game finishes if **all** the things in the environment are collected by the robots or the player robot runs out of lives.

* At the end of the game, if the player robot runs out of lives, then you lose the game.
* If there are still 1 or more lives at the end of the game, then the winner is determined by the number of things collected by the robots.
  + If the player robot has highest number of things in its bag, then you win the game.
  + If one of the monster robots has the highest number of things in its bag, then you lose the game and that specific monster robot wins.

When the game finishes, you will show a message indicating the win/loss status of the game. You should display the message using the given **name of the winner robot**.

**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 7 member functions:

1. **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. This function will be developed during the recitations. If you miss it, you have to write on your own.
2. **PickAllThings**: A member function to pick all of the things in the current cell of the robot object. Both the player and the monster robots will use this member function. Hint: You can make use of **PickThing()** member function for the implementation.
3. **IsAlive**: A member function to check if the robot object is alive or not. This function will be developed in recitations this week. If you miss it, you have to write on your own.
4. **Resurrect**: A member function to resurrect a robot object. Hint: play with **stalled** private data member for the implementation.
5. **SetName**: A member function to sets the name of the robot object. This function will get a string as a parameter and sets the **name private data member.**
6. **GetName**: A member function that returns the name of the robot object.
7. **GetBagCount**: A member function that returns the number of things in the robot’s bag. This function will be useful to determine the winner.

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

**SUCourseUserName\_YourLastname\_YourName\_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 some 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 (only in member function implementations, not in your program), if you change a visual characteristic of a robot, then you have to have the command:

**theRobotWindow->Redraw(this);**

at the end of the member function implementation.

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. Another OO programming strategy is to have member functions to 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 difficulty to control the flow of your programs. Thus we do not approve the use of them. You are also not encouraged to use “break” and “continue”. The use of “break” and “continue” prevent you from forming good readable loop conditions and hence prevent you from learning how to form good loops. Think cleverly in order not to use any of these instructions. If you don't know these commands, do not even try to learn them (we explained “break” in class).

# **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 Robot class. You will do the modifications we mentioned in the previous sections, you are going to add some member functions, you will further update 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' hw6 member functions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* END: CS201 students' hw6 member functions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

MiniFW\_Modified.h and MiniFW\_Modified.cpp files are not needed to be updated by you.

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**. 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.

**New Free Functions Provided**

We supply you (in Robots\_Modified.h and Robots\_Modified.cpp files) some very useful functions. These functions are free functions (not a member function of the Robot class). An example for using these functions will be given in recitations.

int GetThingCount(int x1, int y1, int x2, int y2);

This function returns the number of things available on the field specified by four parameter coordinates. See the figure below. This function will be necessary to check whether the game should end or not.

------------------------------------ (x2, y2)

| |

| Field |  
| |

| |

----------------------------------------

(x1, y1)

int GetCellCount (int x, int y);

This function returns the number of things on the given cell. This function will be necessary to collect all things on a cell by the robots.

void PutThings(int x, int y, int num);

This function puts things at a given cell. The number of things is also an input, so that the code can input all things at once. This function will be useful to randomly populate the world with things.

**Demo** **Application**

Due to interactive feature of this homework, we are not able to provide sample outputs, but we provide an executable file that can be run to understand how your program should work.

We have already written the program for this homework and the corresponding executable file (demohw6.exe) is given to you in the same google drive folder as this homework document. Before pressing “run” button on the menu, either you have to open a sample robot world (files with .rw extension), or you have to create an environment specified in this homework.

# **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 clarify the issues related to homework, so it is to your benefit to attend recitations.

**Grading and Objections**

Careful about the semi-automatic grading: Your programs will be graded using a semi-automated system. Therefore, you should follow the guidelines about input and output order; moreover, you should also use same prompts as given in the Sample Runs. Otherwise semi-automated grading process will fail for your homework, and you may get a zero, or in the best scenario you will lose points.

Grading:

* **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 comment section of your announced homework grade or attend the specified objection hour in your grade announcement.

* Check the comment section in the homework tab to see the problem with your homework.
* Download all .cpp and .h 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 (PLEASE READ, IMPORTANT)?**

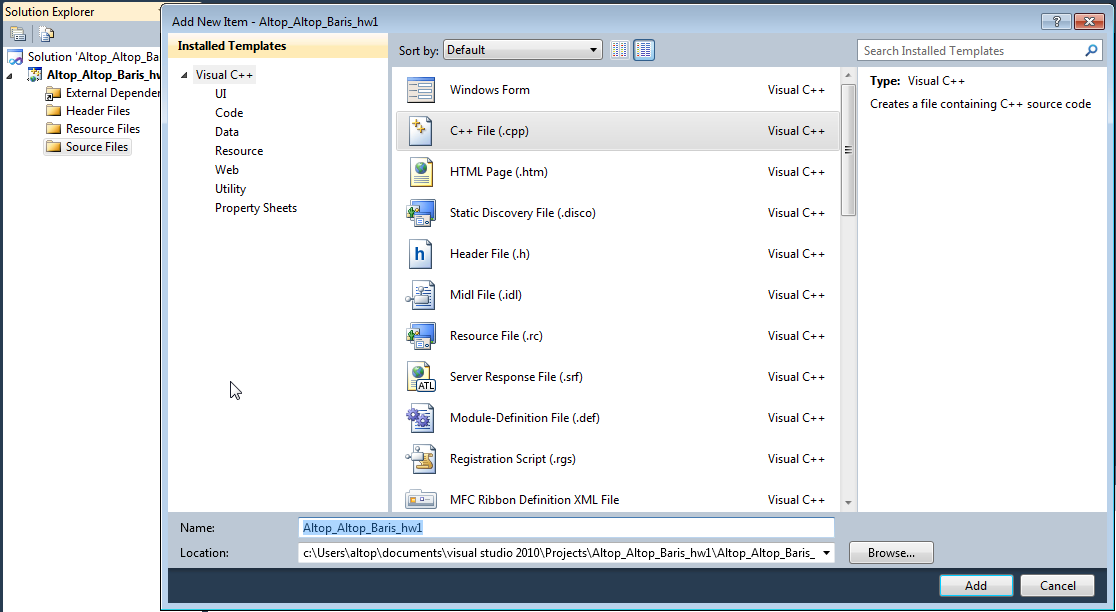
You should prepare (or at least test) your program using MS Visual Studio 2012 C++. We will use the standard C++ compiler and libraries of the abovementioned platform while testing your homework.

It'd be a good idea to write your name and last name in the program (as a comment line of course).

Submissions guidelines are below. Some parts of the grading process are automatic. 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.

* Name your cpp file that contains your program as follows.

“**SUCourseUserName\_YourLastname\_YourName\_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\_hw6.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.
* Check that your compressed file opens up correctly and it contains your **cpp** file. You will receive no credits if your cpp file does not contain the correct file.
* The naming convention of the cpp file is the same as the cpp file (except the extension of the file of course). The name of the cpp file should be as follows.

“**SUCourseUserName\_YourLastname\_YourName\_HWnumber.cpp**”

For example kipler\_kipleroglu\_zubeyir\_hw6.cpp is a valid name, but hw6\_hoz\_HasanOz.cpp, HasanOzHoz.cpp are NOT valid names.

* 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 6 in the assignments list.
3. Click on "Add Attachments" button.
4. Click on "Browse" button and select the cpp file that you generated.
5. Now, you have to see your cpp file 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.

* 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 6 in the assignments list.
3. Click on "Re-submit" button.
4. Click on "Add/remove Attachments" button
5. Remove the existing cpp file by clicking on "remove" link. This step is very important. If you do not delete the old cpp file, we receive both files and the old one may be graded.
6. Click on "Browse" button and select the new cpp file that you want to resubmit.
7. Now, you have to see your new cpp 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!***

***Hanefi Mercan and Gülşen Demiröz***

1. Actually, you do not need to write a special code to kill the normal robots. If a robot moves when blocked by a wall or world boundary, it automatically dies. Moreover, if a robot hits into another robot, both die automatically. [↑](#footnote-ref-0)