

CMPE 150 PROJECT #3

Project Deadline: May 18, 2009 at 23:59

In this project you will implement a simple maze game, in which an agent tries to get to the target. In your implementation you should use the Model-View-Controller design pattern. The expected structure of the program is described below. Make sure you read all of the text, so that you do not miss any required functionality.

Model View Controller (MVC) Design Pattern

You are expected to follow this design pattern for your project. In this section the MVC design pattern will be briefly described, more information and pointers to further resources are available through the Wikipedia entry: <http://en.wikipedia.org/wiki/Model-view-controller>

There are three components of the MVC, *model*, *view*, *controller*. Each of these components are designed and implemented incrementally.

First the model is designed and implemented. In this stage you must implement any data structures your program will use. For example you can define *struct* definitions for your project.

Second you should think about how you will present the model you designed to the user. For example you can implement a function called *view()*, in which you output the values of your data structures to the command line using the *printf* function.

Third you design and implement the controller of your program. This component defines how your model changes according to the input received from the user. For example you can implement a *control()* function, in which you make changes in your model's current state according to the input received from the command line.

Requirements of the Simple Maze Game

Model Requirements

Your program should be able to store several data *structures*, namely the following:

- An **agent** should be able to store its position in 2D coordinate frame and its name as a string.
- An **obstacle** should be able to store its position in 2D coordinate frame and its size. For the sake of simplicity you can assume square obstacles, so that a simple *integer* will be enough to store an obstacle's size.
- A **target** should be able to store its position in 2D coordinate frame.
- The **screen** should be able to store all the information regarding to the current state of the map. Using a 2D *integer* array, in which different states can be stored would be good idea.

In most computer vision applications the 2D coordinate frame is a little different than the common type of coordinate frames. The origin is taken to be the top left corner of the screen, such as in the figure below. Note that the positive x direction is towards **right** and positive y direction is **upwards**. *Make sure that your reference coordinate frame is consistent throughout your program.*

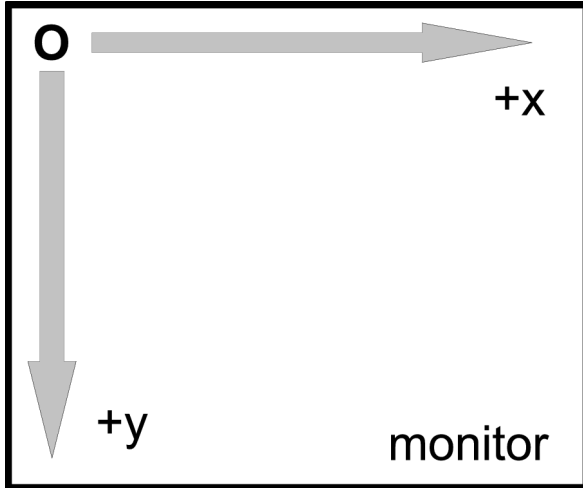


Figure on the left shows the common 2D coordinate frames used in computer vision applications. Note the directions of the x and y axes. **O** indicates the origin

View Requirements

Your viewing component should be able to display the current state of the *screen* data structure and do nothing else. For example you can use such an output method shown below:



Figure on the left shows an example view implementation. The *target* is shown with an x character, the *agent* is shown with an * character. *Obstacles* are drawn as squares.

Besides game elements, at the top of the screen name of the player, input at the beginning of the game, name of the game is also printed.

A border is also drawn to show the limits of the screen.

Controller Requirements

Your controller should be able to receive input from the user. According to this input the *agent* should move in the *screen*. An example is to use keys w,a,s,d to move the agent up, left, down and right.

The controller is also responsible of checking if the *agent* has reached the *target*, in which case the program should end with an appropriate message to let the user know that game has ended.

Technical Details

Main Function

In your main function you should only perform the following, initialize your model elements and run the controller and viewer functions. Observe the pseudo code of a main function below:

```
initialize model elements
while ( true )
    run_view_function
    run_control_function
    if_goal_reached
    break
```

Please follow the MVC design pattern, otherwise your life will not be happier.

View Function

At the beginning of each run of the view function you will need to clean the screen to draw the current state of the screen. To do that you can use the following function call:

```
system("clear");
```

Controller Function

If you do not want to keep pressing enter each time after you give your input to move the agent, use the following function instead of the *scanf* function.

```
char command = getch();
```

This works with the online compiler. If you want to do this in a Linux console, see this post: <http://tinyurl.com/dd9u33>

Functional Requirements

- You must input user's name at the beginning and display it on the title of your game and at the last message of your program.
- Initial positions of the agent and the target must not be out of bounds of the screen.
- Initial positions of the agent and the target must not be inside of the obstacles.

Obstacles

- Centers of obstacles and size of obstacles must be randomly chosen at each run of your program. In other words when your program is run a second time it must start with obstacles at different positions and different sizes.
- You must make sure your obstacles are not drawn on to each other. **Hint:** trust the random process, try pick another rectangle if it does not fit with the other obstacles.

Agent

- Your agent must act sensibly.
- It must not pass through walls.
- It must not pass through obstacles.
- The rule says agents can move one step at a time, towards up, down, left or right.

GRADING:

- Indentation: 10 points. (check example code at the bottom)
- Meaningful variable naming: 10 points
i.e:
`int option;`
`char word[20];`
are good,
`int opt;`
`char wrd[20];`
are acceptable (but not recommended),
`int x;`
`char y[20];`
don't dare!!
- Comments: 10 points (do not write too much comments, but write necessary ones - check example code at the bottom)
- Correct data type (struct,array, array of struct) usage: 30
- Correct execution of the program: 40 points
- Late submission: 20% for each day. Also note that deadline is sharp thus 1 minute after the deadline means late submission. Please submit your project before last day.

Note that projects are mandatory for only cmpe (computer engineering) students (check grading section of web page for details). For non-cmpe students project grade will not effect your course grade. However, the best way of learning programming is writing code thus you are strongly advised to do projects. In addition, we have a very strong tendency to ask one or more questions in the exams directly related to projects.

Cheating: Please read carefully:

http://www.cmpe.boun.edu.tr/courses/cmpe150/fall2008/course_policy.php carefully.

HINTS ABOUT CODING:

- Start early (i.e. now)
- Save frequently, otherwise power failures or computer failures will result in disastrous work loss.
- Take multiple saves (e.g save your source code in another files, or sent yourself an e-mail with source code pasted or attached (do not send your source code to a friend, this is cheating!!))
- Write incrementally. (do not try to write the whole program first and then try to make it work, instead write a small working program which does a tiny job, then develop it)

SUBMISSION:

Submit your projects via online compiler. E-mail submissions will be disregarded.

For submitting your project via online compiler:

- Login to online compiler (<http://cmpe150-1.cmpe.boun.edu.tr>),
- Select the PROJECT UPLOAD tab from the top menu
- Upload your .c source file (if you are using visual studio, it may name your project with cpp extension. Submitting a project with cpp extension is acceptable but using special C++ commands which are not supported by C is not allowed (e.g. cin, cout, iostream etc.)). Do not send your .exe or your .obj file. If you are using online compiler, copy-paste your source to a text file (which has .txt extension) and upload that file. If you are using Visual Studio 2005, you can find your source files under My Documents (Belgelerim for Turkish windows) \Visual Studio 2005\Projects folder (search your file).
- Only your latest submission will be considered. However, your previous submissions will also be stored and listed. Whatever your file name is, it will be renamed as date_time_yourStudentNo.extension

EXAMPLE CODE:

Check the following code for variable namings, comments and indentation (note that anykey() function before return lines in main are for online compiler only, not for Visual Studio or other compilers, also note that the code below is not the solution to the project, just a sample program):

```
/*
cmpe150 fall2008 project 1
Description:   A program that checks whether the number
               is greater than 3 or not.
               It prints 1 on success, 0 on failure
Author:       Mehmet Caliskan
Student no:   2005123123
Section:     08
E-Mail:      mehmet.caliskan@boun.edu.tr
Compiler used: Online Compiler
*/

#include<stdio.h>

int main(){
    int number,result;
    // reading the number:
    scanf("%d",&number);
    // checking whether the number is greater than 3 or not:
    if(number > 3){
        result=1;
    }
    else{
        result=0;
    }
    // output the result:
    printf("%d\n",result);
    anykey();
    return 0;
}
```