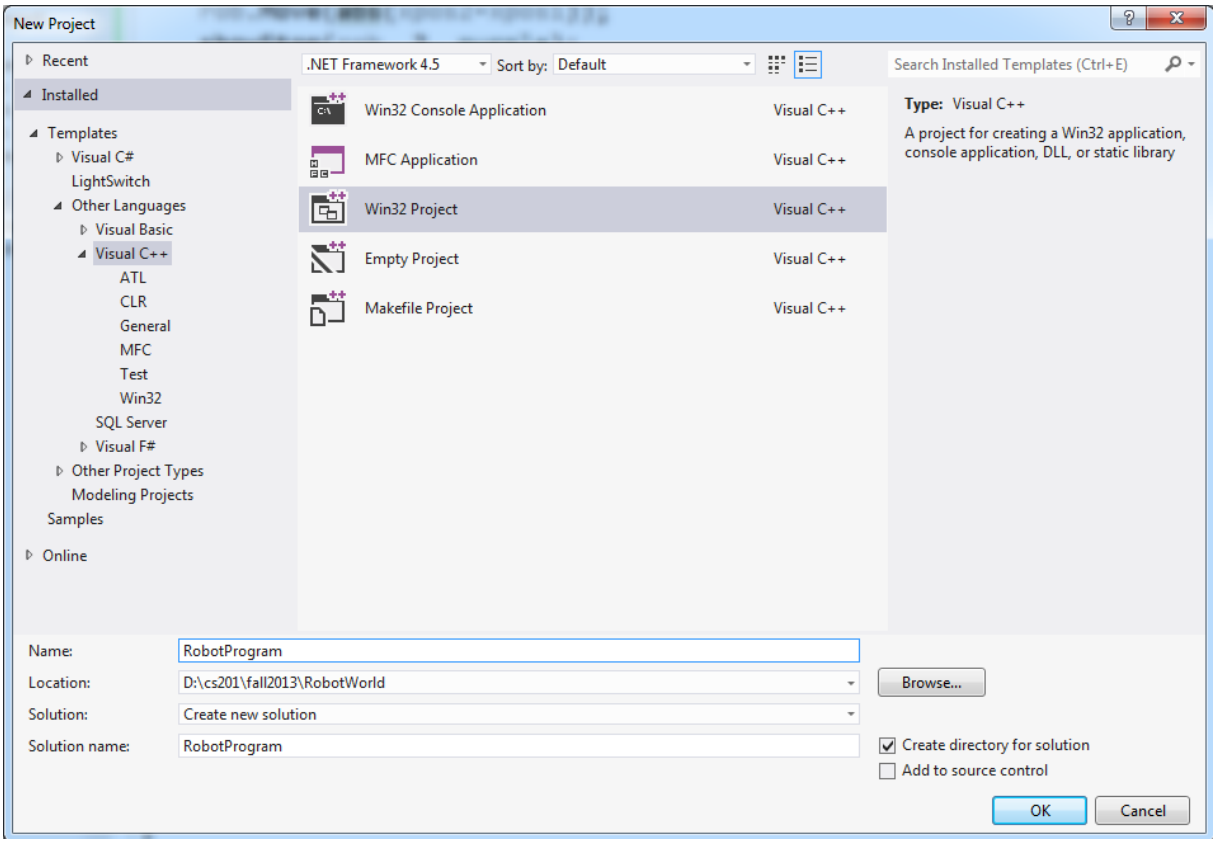
The Robot World

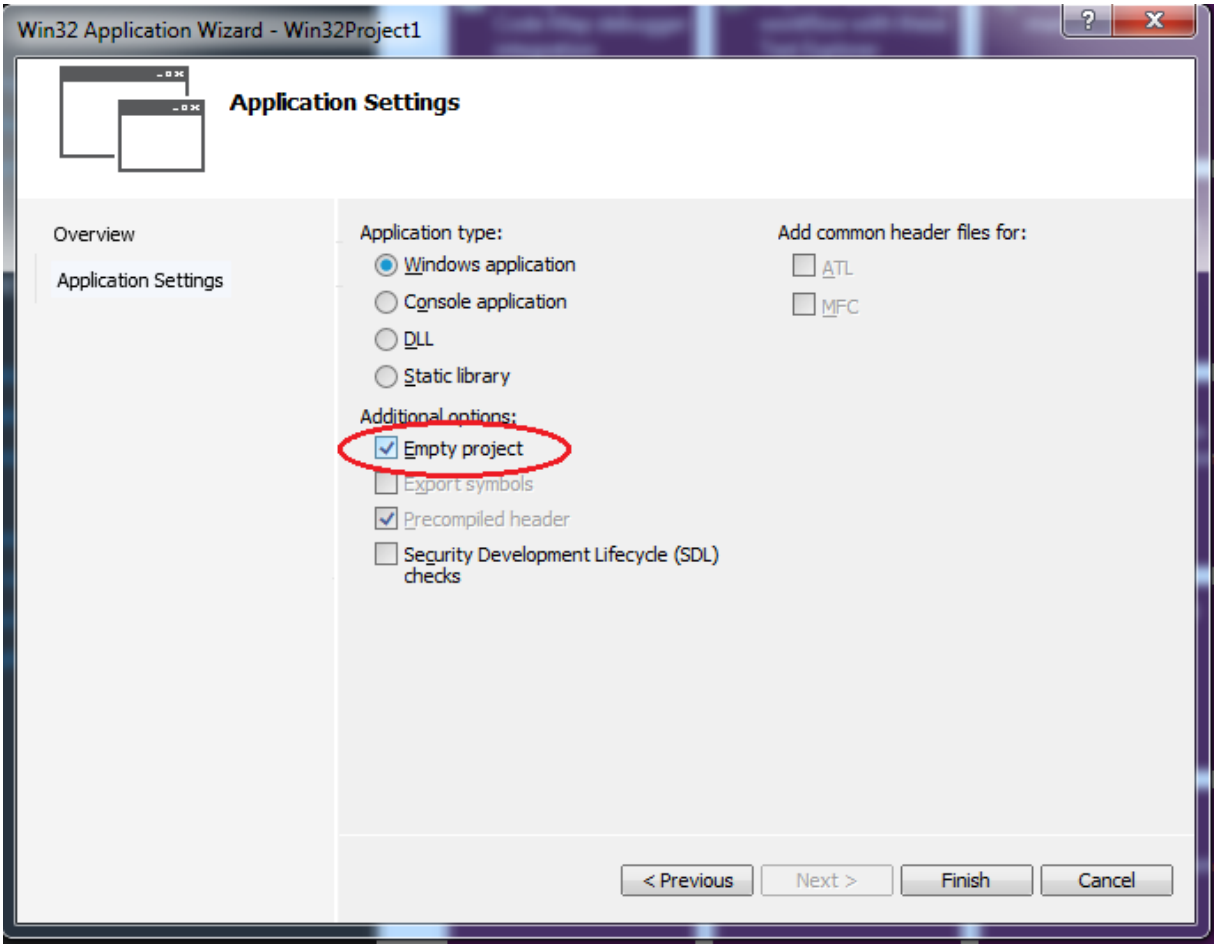
To use the Robot World, you have to add the files named Robots.cpp and MiniFW.cpp into your project, besides the cpp file that your code will appear. These files have to be copied to your project directory, with the corresponding header files Robots.h and MiniFW.h.
The minimal robot program (that does nothing other than creating the robot world window) is this:

```
#include "Robots.h"
int main ()
{
      return 0;
}
```

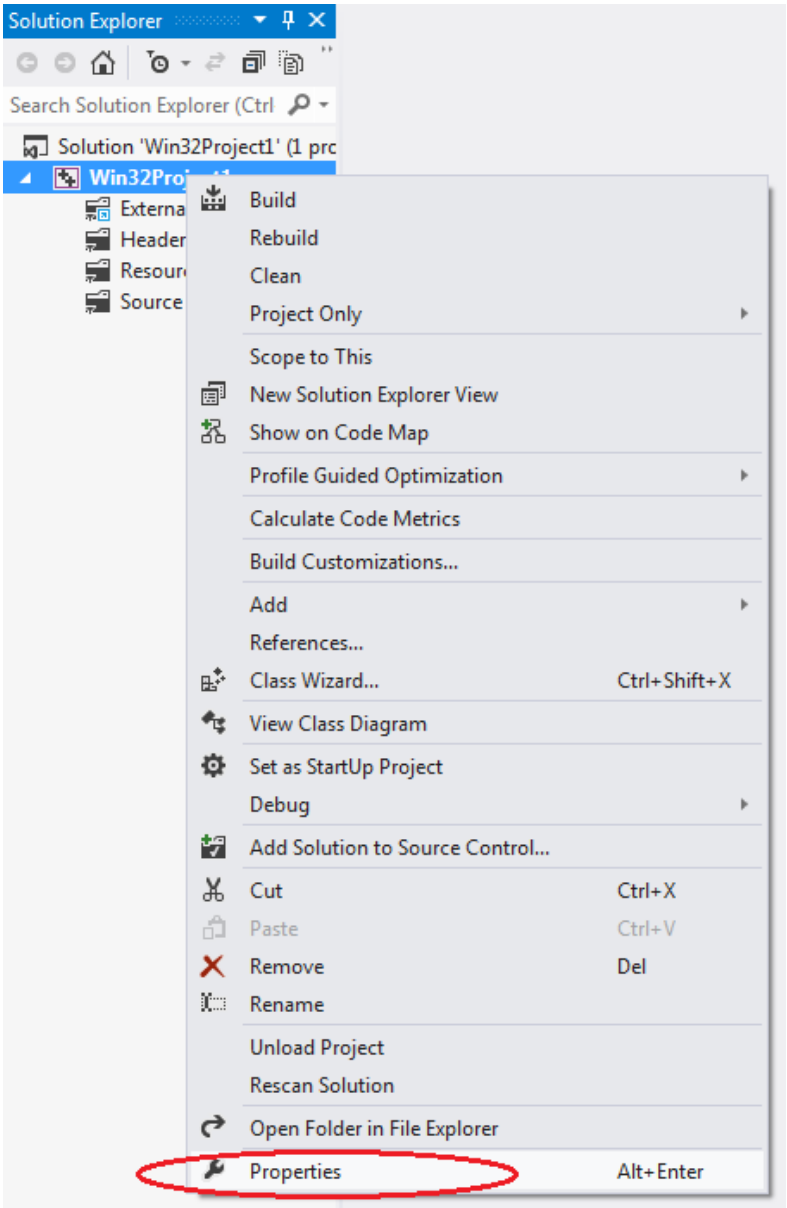
Robot applications are of family called Win32 applications. That is why when creating a project for robots, you have to select Win32 project as seen below (up to now we have used Win32 console applications for programs running at black console windows):



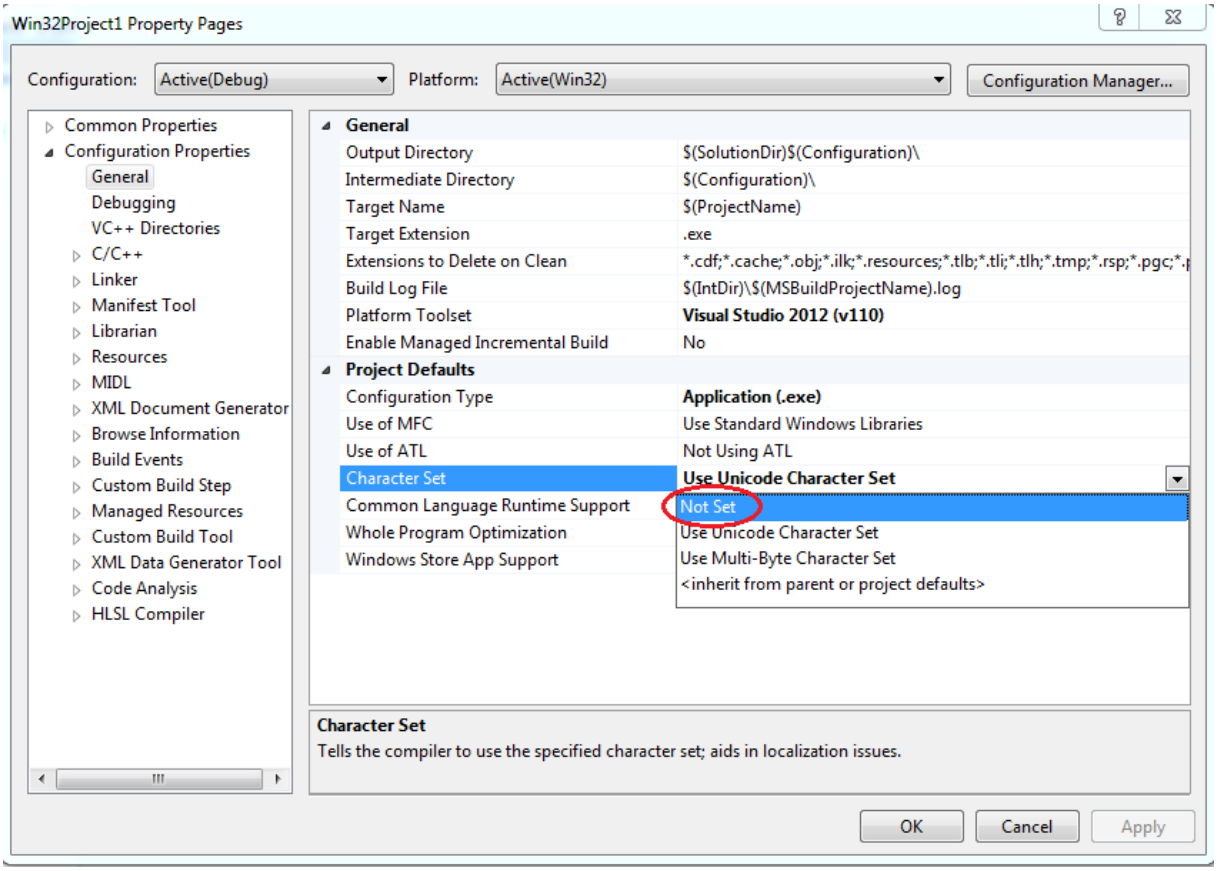
You will still create an Empty Project in the wizard shown below:



One last step would be to change a setting in the Properties of your Project:



In Configuration Properties\General settings, select "**Not Set**" for Character Set:



Windows and visual programming techniques are used here. The `main()` function in which you write your program will not actually be the main function that the execution will start. In win32 applications, the main function that the execution starts with is called `WinMain`. In our robot system, `WinMain` is defined in robots.cpp. It first does the necessary things to construct the robot universe, then, when you select the *Run* menu, it calls your function `main` to run your code.

It should be clear that here we tried to hide `WinMain` function from the programmer and tricked the programmer to believe that he/she writes his/her program in `main`. This is not a usual trick that we do all the time when writing windows applications. However, since CS201 is not a course on windows programming, we preferred not to change your programming abilities and tried to continue with `main` as the starting point of your code. The detailed windows programming is one of the topics of CS204 (advanced programming).

When you set up your project and run the resulting program you will see the (empty) robot world:

Vertical cell coordinates
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
Horizontal cell coordinates

The robot world is divided into cells. Cells are assigned coordinates starting from the bottomleft (southwest) corner. The world is infinite in the north and east directions, but bounded from west and south: There are no cells with negative coordinates. The world also may contain walls between cells, and certain unspecified "things" contained in the cells. A cell may contain any number of "things". If a cell contains some "things", you will see their number written at the center. If there is no number, which is the initial case, that means that cells has no "things".

Initially there are no walls at cell boundaries. However, it is possible to have walls. When exists, a wall between two cells will be shown with a thick line. See the following figure for cells with things and walls. You can edit the world using the mouse: Click on a boundary to create a wall. Click on a wall segment to erase it. Click at the center of a cell with the left mouse button to put one "thing" there, click with the right button to remove one "thing".

Double click with the right button to remove everything in the cell. This is a very primitive interface, indeed.

You may ask "how can I put 35000 things in a cell with this primitive interface?". Well, if you really want to put 35000 things in a cell you may: (a) Reverse engineer the .rw files and edit them with a text editor, or (b) in your main program use a robot to deposit them there. You can save the world as .rw files and use them later. Use the *File* menu to load or save the worlds. You can also do this world editing using a program which has a `main` that has only `return 0` in it.

This world is inhabited by robots. In your main program, you create a robot by declaring a variable of type **`Robot`** (which is defined in Robots.h. This is why we include this file.)

Here is a sample program and its result:

```
#include "Robots.h"
int Main ()
{
      Robot rob(0, 0);
      Robot bob(5, 5, north);
      Robor sam(8, 2, west, 99);
      return 0;
}
```

The constants `east, west, north, south` are defined in Robots.h. You have four parameters to specify when "constructing" a robot: Its horizontal and vertical coordinates, the direction it faces, and the initial number of "things" the robot carries. You may omit the last two parameters. If you omit the things parameter, the compiler will assume that it is zero. If you omit the direction parameter, the robot will face east, as you can see in the picture. You cannot see the "things" a robot carries. They keep them in their pockets.

You can move robots in the world, and make them perform jobs using their member functions. One such `Robot` member function is **`Move`**. When you call `Move` with no parameters, the robot moves to the next cell in the direction it is facing.

```
#include "Robots.h"
int Main ()
{
      Robot rob(0, 0);
      rob.Move();
      return 0;
}
```

If a robot tries to move when it is blocked, that is, when it is next to a wall, or the western or the southern boundary, or another robot, and is facing the obstruction, it *crashes*, and becomes inoperational. You can check this condition before moving by using the member function **`Blocked`**.

Here is a more formal description of all member functions of `Robot`.

`Robot` member functions

Move

`void Move ();`

Moves the robot one cell in the direction it is facing.
If the robot is blocked by a wall, world boundary, or another robot, it crashes.

`void Move (int numcells);`

Moves the robot `numcells` cells in the direction it is facing.
During this journey, if the robot is blocked by a wall, world boundary, or another robot, it crashes.

Blocked

`bool Blocked ();`

Returns `true` if the robot is blocked by a wall, world boundary, or another robot.
Returns `false` otherwise.

TurnRight

`void TurnRight ();`

Changes the direction of the robot (relative to its original direction). For example, if the robot
is facing north, it will be facing east after `TurnRight` is called.

PickThing

`bool PickThing ();`

Get one "thing" from the current cell and put it in the "bag".
If there is at least one thing in the cell occupied by the robot, one is transferred to the robot's bag. In this case the function returns `true`. If the cell is empty, nothing happens, and the function returns `false`.

PutThing

`bool PutThing ();`

Get one "thing" from the bag and put it in the current cell.
If there is at least one thing in the robot's bag, one is transferred to the cell occupied by the robot. In this case the function returns `true`. If the bag is empty, nothing happens, and the function returns `false`.

SetColor

`void SetColor (Color color);`

Changes the color of the robot.
`Color` is an enumerated type defined in Robots.h. It can take the values `white, yellow, red, blue, green, purple, pink, orange`.

FacingEast

`bool FacingEast ();`

Returns `true` if the robot is facing east.
Returns `false` if it is facing any other direction.

FacingWall

`bool FacingWall ();`

Returns `true` if the robot is blocked by a wall or the world boundary (but not when it is
blocked by another robot.)
Returns `false` otherwise.

CellEmpty

```
bool CellEmpty ();
```

Returns `true` if the current cell contains nothing.

Returns `false` if the current cell contains one or more things.

BagEmpty

```
bool BagEmpty ();
```

Returns `true` if the robot's bag contains nothing.

Returns `false` if the robot's bag contains one or more things.

Input and Output in Robot Applications

In robot applications, we cannot use `cin` and `cout`. Since it is a windows application, input and output operations should be performed by small windows. Thus, we have designed two primitive window-based input and output functions for you. These are namely ShowMessage and GetInput functions. The details of those functions are given below. **Please notice that these are not regular I/O routines that we will use in CS201; we just need them when we write robot programs.** Those two functions are NOT member functions of robot class. They are user-defined utility (free) functions.

In a regular robot application, you do need to have `#include <string>` and `using namespace std;` lines, but if you are planning to use these two I/O functions, you have to have these two lines at the beginning of your program together with #include "robot"

```
void ShowMessage(string message)
void ShowMessage(int num)
```

It takes a parameter and displays the value of it inside a message box that has an OK button to be closed. The parameter can be either a string or an integer. You may not have more than one parameter.

```
void GetInput(string prompt, string inp)
void GetInput(string prompt, int inp)
```

GetInput function requires two parameters. First parameter is a string that prompts the user for the input. The other parameter is a variable, which can be either a string variable or an integer variable. The function simply prints the prompt inside an input box and assigns the user input to the input variable. Input box has two bottoms, "OK" and "Cancel". "OK" button is used to assign the input to the variable, and "Cancel" button assigns an empty string to the variable if the type of variable is string, or similarly 0 is assigned if the type of the variable is integer. If the user enters a non-integer value where integer is expected, then the value of the integer becomes 0. GetInput function may not be used to input several variables; in such a case, you have to call it several times.