1.

You’ll write four methods in Game.cs. The game engine is written in Program.cs, which will call your methods as needed. As you complete this project, make sure that all of your methods are named exactly as specified so that they can be called correctly when the game is played.

You don’t need to edit Program.cs to get your game working.

In the game, you control the character with the keyboard. Up moves the character up, left moves the character left, etc. Close the game by pressing Q or CTRL + C

2.

Create an UpdatePosition() method. This method will be used to change the position of the character based on keyboard input. The method should:

have the public new static modifiers

return nothing

have three arguments: a string representing the key pressed, an int representing the change in the x coordinate, and an int representing the change in the y coordinate

have the last two parameters labeled with out

In the method, you should set the value of the two int arguments based on the string value. For example: if the string is "DownArrow", the x coordinate should change by 0 units and the y coordinate should be increased by 1 unit.

(You’ll notice that the y-axis is “flipped”. That’s because the origin, or (0,0) point, is in the top-left!)

3.

Create an UpdateCursor() method. It will allow the player icon to change with each keypress. Its input will be the key pressed and it will return a symbol that represents the player. The method should:

have the public new static modifiers

return a char

have one argument: a string representing the key pressed

In our example we used the following mapping:

"LeftArrow" : '<'

"RightArrow" : '>'

"UpArrow" : '^'

"DownArrow" : 'v'

4.

Create a KeepInBounds() method. Without this method, hitting the boundaries will break the game!

Every time a key is pressed, it will be called twice: once for x and once for y. The method will constrain a coordinate between 0 and its maximum value to keep the character from going off-screen.

The method should:

have the public new static modifiers

return an int

have two arguments: an int representing the coordinate, and an int representing the maximum value

Here’s how the method will be used. Imagine that the x-coordinate has a maximum of 10 (exclusive), so it should only take values from 0 through 9:

// newX will be 1

newX = KeepInBounds(1, 10);

// newX will be 0

newX = KeepInBounds(-1, 10);

// newX will be 9

newX = KeepInBounds(10,10);

// newX will be 9

newX = KeepInBounds(11,10);

5.

Create a DidScore() method. This will allow the score to increase if the character hits the @ object — let’s call it the “fruit”.

It should:

have the public new static modifiers

return a bool

have four int arguments: the x and y of the character, and the x and y of the “fruit”

The method will return true if they overlap and false otherwise. For example, if the character and “fruit” are both at (1,5), it should return true:

// scored will be true

scored = DidScore(1, 5, 1, 5);

If the character is instead at (1,4), it should return false:

// scored will be false

scored = DidScore(1, 4, 1, 5);

Extensions:

6.

Great work! If you’d like to extend your project on your own, you could consider the following:

Customize the character and controls! Maybe your character is an emoji, maybe you use W, A, S, D instead of the arrow keys.

Make the character “loop around” the bounds of the screen. If the character moves all the way right, it should reappear on the left. If the character moves all the way down, it should reappear on the top.