ICC343 Advanced Programming - 2021-2

Programming Lab. 10

Rabbits and Carrots Simulation

PhD. Carlos Cares

Previously you must read this:

The goal of this lab is to extend the previous C#.NET solution about Game of Life to an Environment where there are rabbits and carrots also represented as a cell. Rabbits and Carrots will have their own reproduction rules but, additionally, rabbits can eat carrots.

This learning activity will follow a “Tutorial style” in order to not impact you too much with this new technology. Both cases, game of life and rabbits and carrots, represent the base of cellular automaton simulation solutions which are, of course, advanced programming techniques.

The way of presenting your solution is pasting screenshots in the slack channel. Please paste the screenshots, do not attach files because it requires too many additional steps. You must be very explicit about the problem which you are answering. Most of the problems require that you show both: code and outputs. ALL screenshots MUST show part of the background of your desktop including the date and time.

GENERAL MAP OF PROBLEMS AND SOLUTIONS.

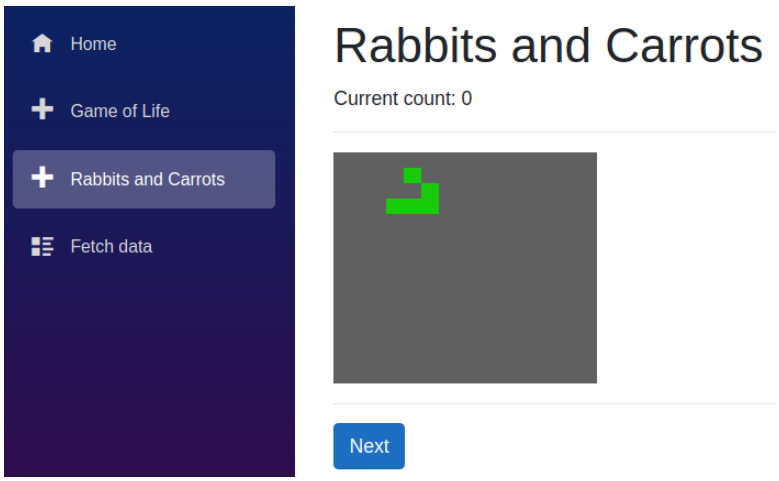
N°	%	Problem
1	10%	You must show the previous Game of Life Working, BUT with a different pattern.
2	15%	You must add a menu option for the New Environment of Rabbits and Carrots
3	25%	You must change the class Environment. The focus is very different. Now some spaces can be null cells (no objects in positions). The first versions of Rabbit and Carrots will be created. But there are no different colors.
4	10%	You must change the displaying system in order for rabbits to appear as white cells and carrots as orange cells.
5	15%	You must implement the `neighbors` method for delivering the objects surrounding a specific BioUnit.
6	25%	You must implement the Rabbit and Carrots reproduction rules
Total	**100%**	

Problem 1. 10%. Review in Wikipedia the Conway's Game of Life (https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life) and show (configure) ONE "still life" pattern and ONE "oscillator pattern" in the same Environment. Required:

TP10-P1-A Screenshot of the fragment of the code generating the patterns
TP10-P1-B 2 Screenshots of the output in your browser, "as is" and after the first click.
TP10-P1-C The complete source code of your solution. If you will answer the next problem then you can skip this request.

There is no guide for this problem.

Problem 2. 15%. Add a menu option for the Environment “Rabbits and Carrots”. You can keep the solution of Game of Life in this new option but having the new title, like this:

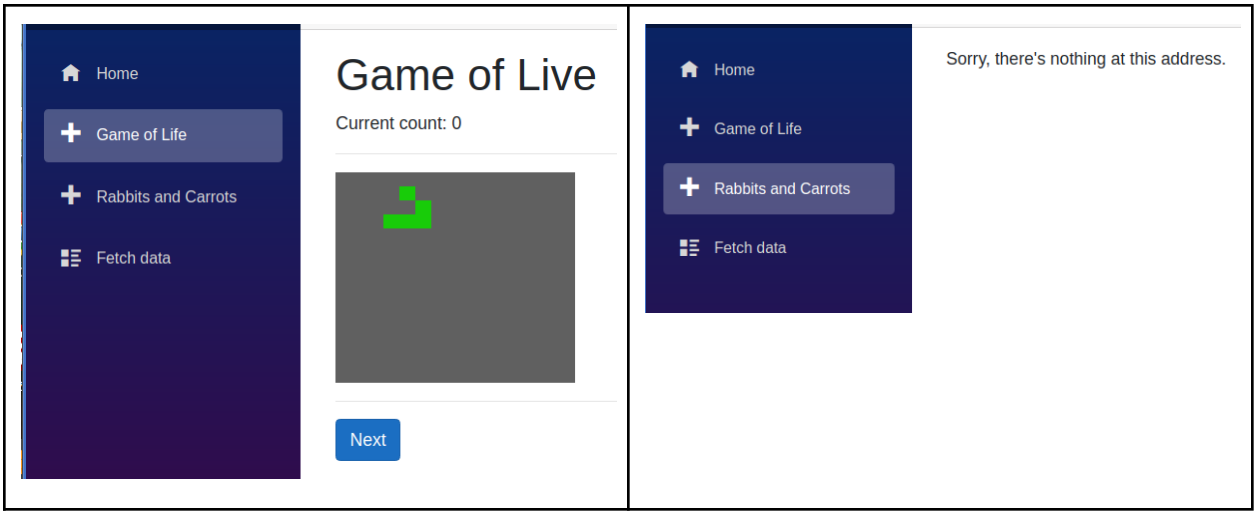


In NavMenu.razor copy and paste the option of “Game of Life”, change the href attribute and the name of the option.

NavMenu.razor

```
20          <li class="nav-item px-3">
21              <NavLink class="nav-link" href="rabbits_and_carrots">
22                  <span class="oi oi-plus" aria-hidden="true"></span> Rabbits and Carrots
23              </NavLink>
24          </li>
```

A temporary output should be as follows

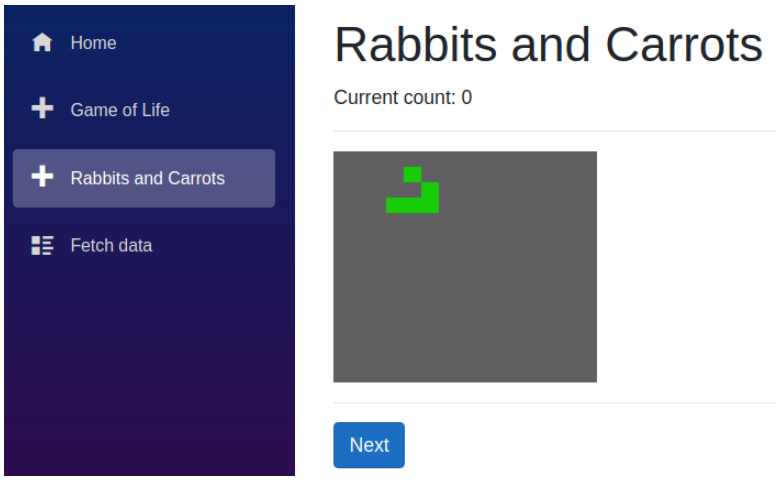


Copy Counter.razor as Rabbits_and_carrots.razor and change the @page line

Rabbit_and_carrots-razor

```
Pages > Rabbits_and_carrots.razor
  1   @page "/rabbits_and_carrots"
  2   @using System
```

Depending on your chosen Game of Life pattern the output can be different, but it should look like this:

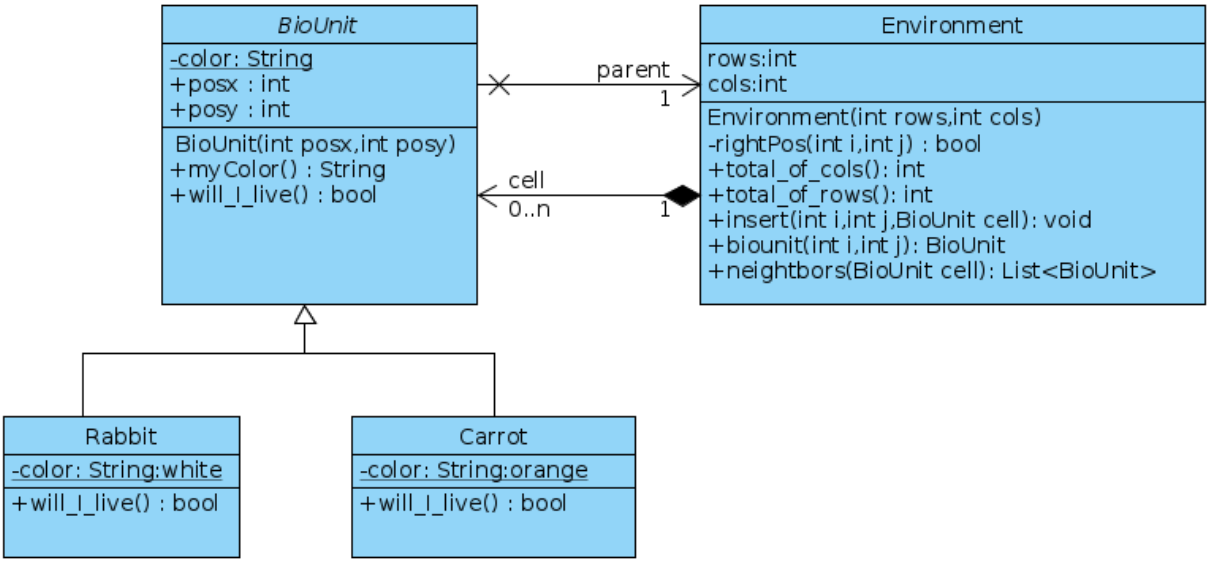


Required:

TP10-P2-A	Screenshot of 5 lines of the file Rabbit_and_carrots.razor
TP10-P2-B	Screenshot of your output ON the option Rabbits and Carrots
TP10-P2-C	The complete source code of your solution. If you will answer the next problem then you can skip this request.

Problem 3. 25%. In this problem, we will change the class Environment. Now the array can not contain any object, therefore, a dead cell will be represented as null. It will imply that the Game of Life solution will not work anymore. It is not part of this problem to upgrade the Game of Life solution, therefore, just put comments (/* */) to the lines having problems in the part of "Game of Life".

The following diagram shows (more or less) what it is pretended. Mainly the environment will contain objects from the class BioUnit in SOME of its cells (spaces). In this problem, only the cell-association is implemented. Also the first versions of the classes Rabbit and Carrot as subclasses of BioUnit.


BioUnit
-color: String
+posx : int
+posy : int
BioUnit(int posx,int posy)
+myColor() : String
+will_I_live() : bool
parent
1
cell
0..n
1
Environment
rows:int
cols:int
Environment(int rows,int cols)
-rightPos(int i,int j) : bool
+total_of_cols(): int
+total_of_rows(): int
+insert(int i,int j,BioUnit cell): void
+biounit(int i,int j): BioUnit
+neightbors(BioUnit cell): List<BioUnit>
Rabbit
-color: String:white
+will_I_live() : bool
Carrot
-color: String:orange
+will_I_live() : bool

This is the suggested version of Environment class

```
4       public class Environment
5       {
            5 references
6           private int rows = 1;
            5 references
7           private int cols = 1;
            4 references
8           private BioUnit[,] cell;
            2 references
9           public Environment(int rows_,int columns_) {
10              this.rows = rows_;
11              this.cols = columns_;
12              this.cell = new BioUnit[this.rows,this.cols];
13              for(var i=0; i<this.rows; i++)
14              for(var j=0; j<this.cols; j++)
15                  this.cell[i,j] = null;  //  <---  said null
16          }
            2 references
17          public int total_of_rows(){
18              return this.rows;
19          }
            2 references
20          public int total_of_cols() {
21              return this.cols;
22          }
            2 references
23          private bool rightPos(int i,int j){
24              return ((i>=0) && (i<this.rows) && (j>=0) && (j<this.cols));
25          }
            8 references
26          public void insert(int i,int j,BioUnit been) {
27              if(this.rightPos(i,j)){
28                  this.cell[i,j] = been;
29              }
30          }
            2 references
31          public BioUnit biounit(int i,int j) {
32              if(this.rightPos(i,j)){
33                  return this.cell[i,j];
34              }
35              return null;
36          }
```

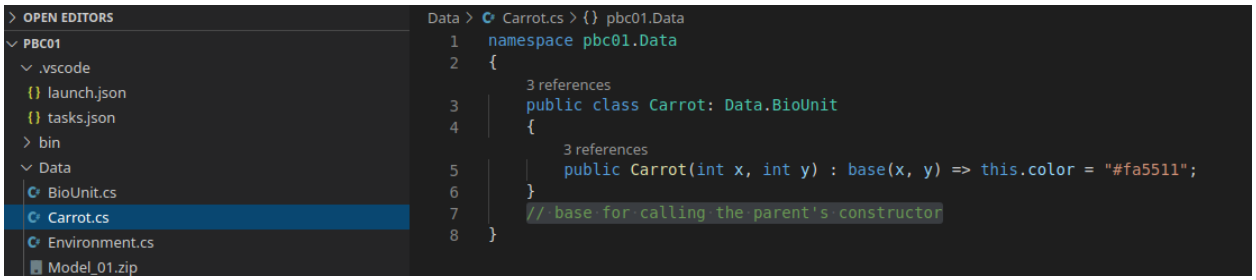
The previous Conway's step method is commented.

```
37
38  /*        public void nextConwayStep() {
39              int n;
40              bool[,]  aux = new bool[this.rows,this.cols];
41              for(var i=0; i<this.rows; i++)
42              for(var j=0; j<this.cols; j++) {
43                  n = this.aliveNeighbors(i,j);
44                  if(n==3)
45                      aux[i,j] = true;
46                  else if (n==2 && this.is_alive(i,j))
47                      aux[i,j] = true;
48                  else
49                      aux[i,j] = false;
50              }
51              for(var i=0; i<this.rows; i++)
52              for(var j=0; j<this.cols; j++) {
53                  if(aux[i,j])
54                      this.live(i,j);
55                  else
56                      this.die(i,j);
57              }
58          } */
59      }
60  }
```

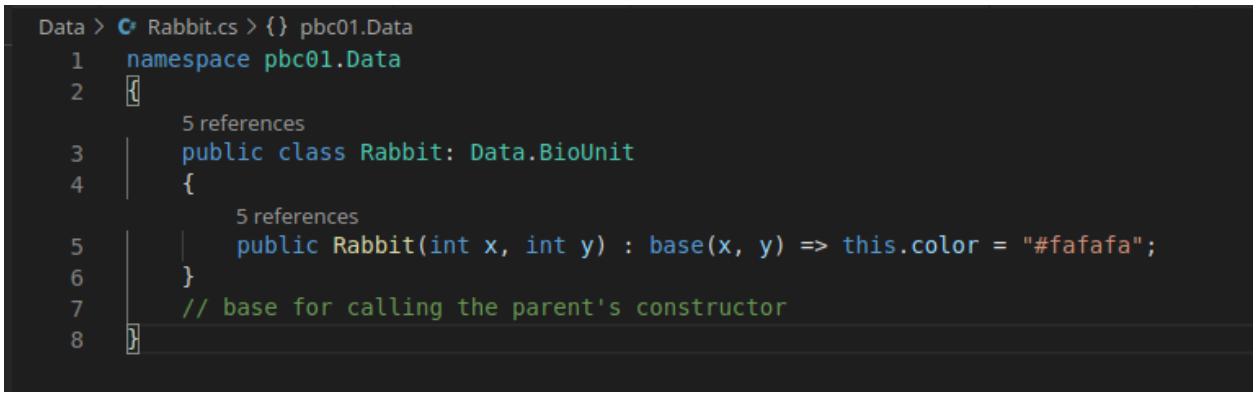
The class BioUnit

```
Data > BioUnit.cs > {} pbc01.Data > pbc01.Data.BioUnit
1   namespace pbc01.Data
2   {
        10 references
3       public class BioUnit
4       {
            4 references
5           protected string color;
            1 reference
6           public int posx;
            1 reference
7           public int posy;
            2 references
8           public BioUnit(int x,int y) {
9               this.posx = x;
10              this.posy = y;
11              this.color = "#444444";
12          }
            0 references
13          public string myColor() => this.color;
14
15      }
16  }
```

The class Carrot

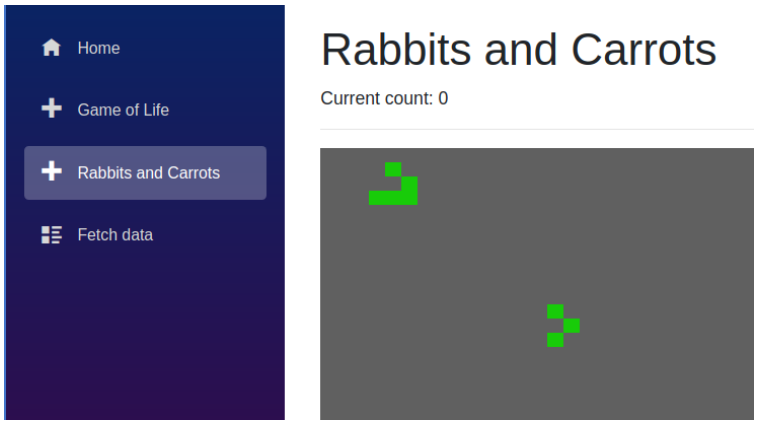


The class Rabbit



And a suggested version for the razor file Rabbits_and_carrots

The output MUST be:



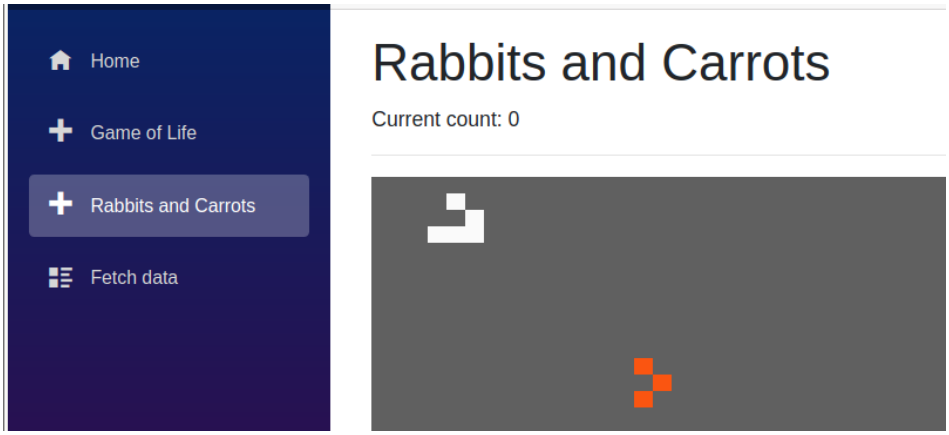
Required:

TP10-P3-A Screenshot of Carrot.cs
TP10-P3-B Screenshot of Rabbits_and_carrots.razor
TP10-P3-C Screenshot of the output
TP10-P3-D The complete source code of your solution. If you will answer the next problem then you can skip this request.

Problem 4. 10%. In this problem, you should make the changes for getting the right colors for rabbits (white) and carrots (orange). The solution is provided. But note that it has a different approach from “Game of Life”. Here I preferred to use directly a color output from the object and not from a CSS-style file. It would allow having other classes of objects in the future without adding styles.

```
34          @for(var j=0; j<@e.total_of_cols(); j++) {
35              @if(e.biounit(i,j)!=null) {
36                  <td class="cell" style="background-color:@e.biounit(i,j).myColor()"></td>
37              }
38              else
39              {
40                  <td class="cell dead"></td>
41              }
42          }
```

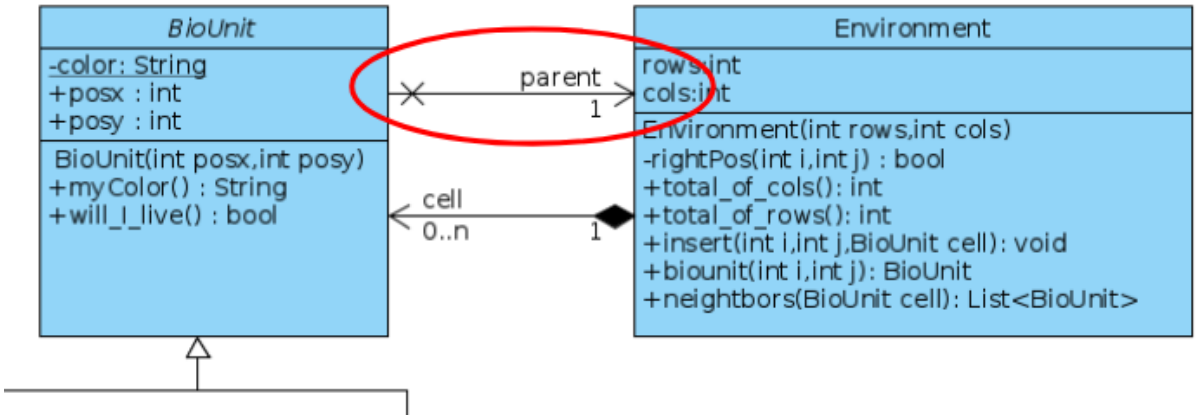
Output:



Required:

TP10-P4-A Screenshot of the code but starting at least 5 lines before.
TP10-P4-B Screenshot of the similar output
TP10-P1-C The complete source code of your solution. If you will answer the next problem then you can skip this request.

Problem 5. 15%. In this problem, the parent association must be implemented which requires changes in the constructors in order for the created objects to know their parent. The goal is checked by testing the implementation of the `neighbors`() method because BioUnit can ask for its neighbors, which implies navigating through the parent association. Please put attention to this implementation



Moreover, a good way of dealing with new features in code is writing which should run without problems. Therefore, I did put in the file Rabbits_and_carrots.razor the creation of rabbits and carrots in specific positions in the Environment `e`. Note that, now, the environment itself is a parameter, because this is the way that the `BioUnit` under creation "knows" its creator. Besides, I have written some lines in the terminal in order to see that the new `neighbors()` method works.

```
@{
    //only the first time (initial pattern)
    if(currentCount==0) {
        u = new Data.Rabbit(3,3,e);
        e.insert(3,3,u);
        u = new Data.Rabbit(3,4,e);
        e.insert(3,4,u);
        u = new Data.Rabbit(3,5,e);
        e.insert(3,5,u);
        u = new Data.Rabbit(2,5,e);
        e.insert(2,5,u);
        u = new Data.Rabbit(1,4,e);
        e.insert(1,4,u);

        u = new Data.Carrot(11,14,e);
        e.insert(11,14,u);
        u = new Data.Carrot(12,14,e);
        e.insert(12,15,u);
        u = new Data.Carrot(13,14,e);
        e.insert(13,14,u);
        u = new Data.Carrot(0,0,e);
        e.insert(0,0,u);

        int count1 = e.neighbors(2,4).Count();
        int count2 = e.neighbors(13,14).Count();
        int count3 = e.neighbors(1,1).Count();
        Console.WriteLine("Neightbors");
        Console.WriteLine(count1);
        Console.WriteLine(count2);
        Console.WriteLine(count3);
    }

}
```

Therefore, we need a new `BioUnit`'s constructor

```
9           public BioUnit(int x,int y,Environment e) {
10              this.posx = x;
11              this.posy = y;
12              this.color = "#444444";
13              this.parent = e;
14          }
```

And also in its derived classes (`Carrot` and `Rabbit` classes)

```
Data > Carrot.cs > {} pbc01.Data
1   namespace pbc01.Data
2   {
        3 references
3       public class Carrot: Data.BioUnit
4       {
            3 references
5           public Carrot(int x, int y,Data.Environment e) : base(x, y,e) => this.color = "#fa5511";
6       }
7       // base for calling the parent's constructor
8   }
```

```
Data > Rabbit.cs > {} pbc01.Data
1   namespace pbc01.Data
2   {
        5 references
3       public class Rabbit: Data.BioUnit
4       {
            5 references
5           public Rabbit(int x, int y,Environment e) : base(x, y, e) => this.color = "#fafafa";
6       }
7       // base for calling the parent's constructor
8   }
```

In the case of `Environment` class (Environment.cs file) now, it needs a List of "somethings". The class List is in the Package System.Collections.Generic, thus we need to include this package (line 2 in the next figure).

In Environment.cs

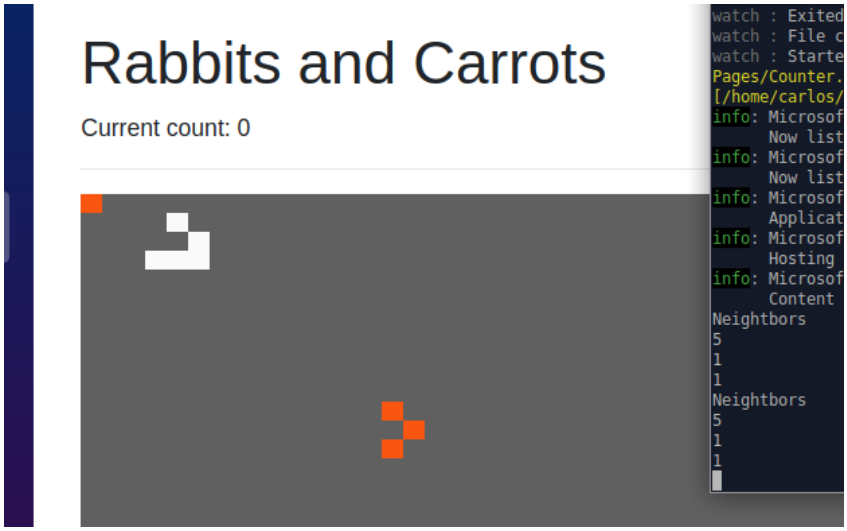
```
Data > Environment.cs > ...
1   using System;
2   using System.Collections.Generic;
3   namespace pbc01.Data
4   {
        6 references
5       public class Environment
```

Now, I show here an implementation of `neightbors()` method which returns a list of the existing biounits (wherever its class). Note that the first to check is in i-1,j-1

In Environment.cs

```
39      public List<BioUnit> neighbors(int i,int j) {
40          List<BioUnit> ans = new List<BioUnit>();
41          if(this.rightPos(i,j)){
42              if( this.rightPos(i-1,j-1) && this.cell[i-1,j-1]!=null) ans.Add(this.cell[i-1,j-1]);
43              if( this.rightPos(i-1,j) && this.cell[i-1,j]!=null) ans.Add(this.cell[i-1,j]);
44              if( this.rightPos(i-1,j+1) && this.cell[i-1,j+1]!=null ) ans.Add(this.cell[i-1,j+1]);
45              if( this.rightPos(i,j-1) && this.cell[i,j-1]!=null ) ans.Add(this.cell[i,j-1]);
46              if( this.rightPos(i,j+1) && this.cell[i,j+1]!=null) ans.Add(this.cell[i,j+1]);
47              if( this.rightPos(i+1,j-1) && this.cell[i+1,j-1]!=null ) ans.Add(this.cell[i+1,j-1]);
48              if( this.rightPos(i+1,j) && this.cell[i+1,j]!=null ) ans.Add(this.cell[i+1,j]);
49              if( this.rightPos(i+1,j+1) && this.cell[i+1,j+1]!=null ) ans.Add(this.cell[i+1,j+1]);
50          }
51          return ans;
52      }
```

And finally the output (including the terminal)



Required:

TP10-P5-A Change the file Rabbits_and_carrots.razor in order to the output shows 1,2 and 3 as neighbors of some biounits.
TP10-P5-B Screenshot of the changed lines in Rabbits_and_carrots.razor
TP10-P5-C Screenshot of the output considering the terminal with the corresponding output
TP10-P5-D The complete source code of your solution. If you will answer the next problem then you can skip this request.

Problem 6. 25%. In this problem, you must implement the following rules.

- Rabbits can live 6 cycles at maximum
- Rabbits can not live more than 3 cycles without eating (carrots).
- Carrots can live 3 cycles at maximum.

Under this consideration, the reproduction rules are the following.

1) A space occupied by a carrot, and there are no surrounding rabbits, will keep that carrot while it does not have more than 3 cycles. In this case, the carrot only adds a new life cycle.
2) A space occupied by a carrot that has 4 cycles of life (or more) will be empty in the next cycle, i.e. the carrot will die. Note that this carrot can not be consumed and the space can not be occupied by another entity because it is currently occupied by a non-consumable carrot.
3) A space occupied by a consumable carrot and having a rabbit neighbor in the next cycle will be an empty space. When several rabbits find the same carrot the carrot can be consumed only for one rabbit. They are genetically organized to respect a north-west priority (first i-1,j-1, then i,j-1, etc..). A rabbit can eat more than a carrot in the same cycle. The rabbit that eats will again start its “hungry” counter.
4) A space occupied by a rabbit will keep that rabbit when it does not have more than 6 cycles. In this case, the rabbit only adds a new life cycle.
5) A space occupied by a rabbit that has passed 4 cycles of life (or more) without eating will be empty in the next cycle, i.e. the rabbit will starve.
6) An empty space has the following priority to be occupied. (a) The space will have a rabbit or at least, two surrounding rabbits. (b) The space will have a carrot, by the density of seeds, or it has, at least, three surrounding carrots.

These rules imply that we have to enhance the model in order to support the number of life cycles and the amount of time without eating (in the case of rabbits).

In BioUnit.cs is added a virtual in the method will_I_live in order that it can be overloaded (in the subclasses).

```
17          public string myColor() => this.color;
            2 references
18          public virtual bool will_I_live() => true;
19      }
20  }
```

Therefore in classes Carrot and Rabbit you should specify that the method is a different method than the superclass. Therefore in the class `Carrot` the method should be:

```
            5 references
5           public Carrot(int x, int y,Data.Environment e) : base(x, y,e) {
6               this.color = "#fa5511";
7               this.living=0;
8               this.livingTop=3;
9           }
            2 references
10          public override bool will_I_live() {
11              this.living++;
12              if((this.living-1)>=this.livingTop) return false;
13              return true;
14          }
15      }
```

And in the class `Rabbit` the methods stay as follows:

```
            6 references
7           public Rabbit(int x, int y,Environment e) :base(x,y,e) {
8               this.color = "#fafafa";
9               this.living=0;
10              this.livingTop=6;
11          }
            2 references
12          public override bool will_I_live() {
13              this.hungry++;
14              this.living++;
15              if((this.living-1)>=this.livingTop) return false;
16              if((this.hungry-1)>=this.hungryTop) return false;
17              return true;
18          }
            1 reference
19          public void eat() {
20              this.hungry=0;
21          }
22      }
23  }
```

Most of the relevant changes are in the Environment class. New packages are required for changing List of objects (`List<Object>`) and also for using the method for getting the object's name of the class. Here follows:

```
Data > Environment.cs > {} pbc01.Data > pbc01.Data.Environment >
1   using System;
2   using System.Collections.Generic;
3   using System.ComponentModel;
4   namespace pbc01.Data
5   {
        6 references
6       public class Environment
7       {
            8 references
8           private int rows = 1;
            8 references
9           private int cols = 1;
```

Also, new methods are implemented in this class. Here we have the method for obtaining the number of surrounding neighbors belonging to a particular class.

```
            3 references
54          public int surrondingNeighborns(int i,int j,String specie) {
55              int ans=0;
56              List<BioUnit> surr = this.neighbors(i,j);
57              Console.WriteLine(" i  j  "+i.ToString()+" , "+j.ToString());
58              foreach (object unit in surr)
59              {
60                  if(this.specie(unit)==specie) ans++;
61              }
62              return ans;
63          }
64
```

To know the specific name of the class, which is needed because the obtained name from TypeDescriptor includes the packages, for example in this case is obtained project.Data.Carrot. Due to this the string is separated (using split) and the result is the last element of this string.

```
            4 references
65          public String specie(Object obj) {
66              String[] w;
67              if(obj==null) return "";
68              w = TypeDescriptor.GetClassName(obj).Split(".");
69              return w[w.Length-1];
70          }
71
```

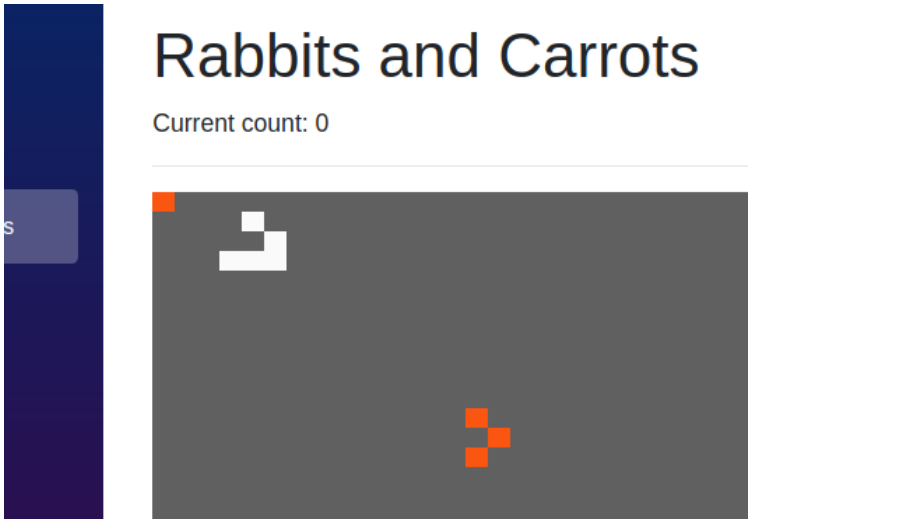
To obtain the first rabbit in the list of neighbors the method firstRabbit was implemented. Here is used a cast for saying that the object unit should be returned as a member of the class Rabbit.

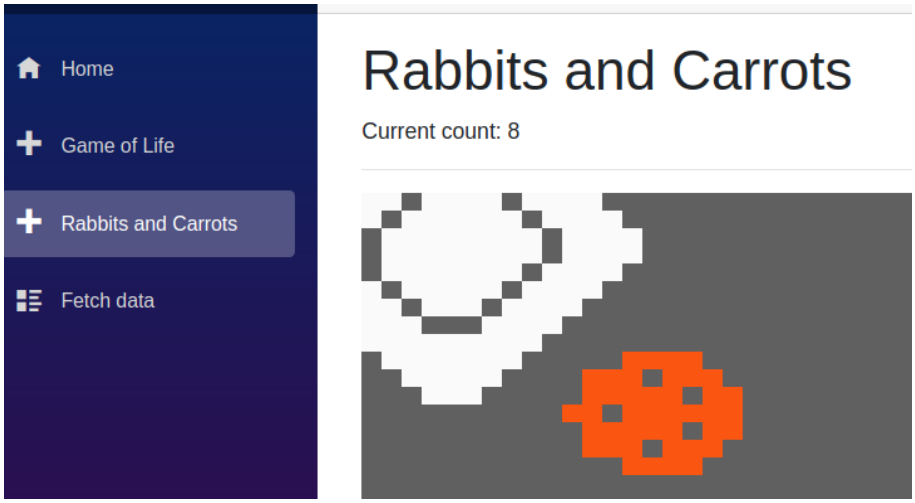
```
1 reference
72      public Rabbit firstRabbit(int i,int j) {
73          List<BioUnit> neis = this.neighbors(i,j);
74          foreach(object unit in neis) {
75              if(this.specie(unit)=="Rabbit")
76                  return (Rabbit) unit;
77          }
78          return null;
79      }
```

And finally, the next implementation step of the simulation.

```
1 reference
81      public void next_Rabbit_Carrot_Step() {
82          BioUnit[,]  aux = new BioUnit[this.rows,this.cols];
83          for(var i=0; i<this.rows; i++)
84          for(var j=0; j<this.cols; j++) {
85              aux[i,j] = null; //rule 2, rule 5
86              //rule 1
87              if(this.specie(this.cell[i,j])=="Carrot") {
88                  if(this.surrondingNeighborns(i,j,"Rabbit")==0) {
89                      if(this.cell[i,j].will_I_live())
90                          aux[i,j] = this.cell[i,j];
91                  }
92                  else { //rule 3
93                      this.firstRabbit(i,j).eat();
94                  }
95              }
96              // rule 4
97              else if (this.specie(this.cell[i,j])=="Rabbit") {
98                  if(this.cell[i,j].will_I_live()) {
99                      aux[i,j] = this.cell[i,j];
100                 }
101             }
102             else { //rule 6
103                 if(this.cell[i,j]==null) {
104                     if(this.surrondingNeighborns(i,j,"Rabbit")>=2)
105                         aux[i,j] = new Rabbit(i,j,this);
106                     else if (this.surrondingNeighborns(i,j,"Carrot")>=3)
107                         aux[i,j] = new Carrot(i,j,this);
108                 }
109             }
110         }
111         for(var i=0; i<this.rows; i++)
112         for(var j=0; j<this.cols; j++) {
113             this.cell[i,j] = aux[i,j];
114         }
115     }
```

The outputs should be as follows:





Required:

TP10-P6-A The complete code of the firstRabbit method
TP10-P6-B The complete code of the next_Rabbit_Carrot_step
TP10-P6-C The function that calls next_Rabbot_Carrot_step in the razor file
TP10-P6-D The output (screenshot) with no clicks
TP10-P6-E The output (screenshot) after 8 clicks
TP10-P6-F The complete source code of your solution.