

Problem-based Case 01

Rabbits and Carrots Simulation

October 14 - 2020

Advanced Programming

PhD. Carlos Cares



Previously you must read this:

The goal of this Problem-based case is to extend the previous solution about Game of Life to an Environment where there are rabbits and carrots. Rabbits and Carrots will have their own reproduction rules but, additionally, rabbits will eat carrots.

This learning activity will follow a “Tutorial style” because I got the idea that some of the students seem to have still basic knowledge about operating systems and also some problems for recognizing common errors and warnings messages from .NET Core environment. I apologize for the rest of you, but I expect that this activity results in fun learning. Anyway, both cases represent the base of cellular automaton simulation solutions which are, of course, advanced programming techniques.

“Tutorial style” means that you can have interaction, however, **I will change the methodology** rescuing best practices from the previous tutorial. In a similar way to a tutorial interaction, you can get interaction points.

The way of presenting your solution is pasting screenshots in the Microsoft team “**personal chat**”. 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 and date and time.

The way of presenting official questions (problems, errors, warnings) for getting interaction points is in the common MS-team chat. Of course, answers must be in the same chat. Please, be explicit saying “thank you to” who really helps you to get a solution.

Please, during the session DO NOT LET QUESTIONS IN THE PERSONAL CHAT and DO NOT LET YOUR ANSWERS IN THE COMMON CHAT.

Please, during the session DO NOT SPEND TIME talking about your score of previous activities. If you have a doubt we will solve after the session.

Also, in an explicit way, I will extend the scoring system. Under a rigorous interpretation of the current scoring system, you can not get points by works after the corresponding session. Anyway, due to the already explained situation of some students, I checked and assessed, like during the session, several works. It gives me a good opportunity to interact and also show me your interest in the corresponding results. But, it is not fair for students who finishing on time and under pressure. **Therefore the following are extended rules for THIS problem-based case.**

ER-1. Works delivered in the next 24 hours to the end of the session will be **reduced 20%** (for example a problem that has 15% will have $15 \cdot (1 - 0.2) = 12\%$).

ER.2 Works delivered between 25 and 72 hours after the session will be **reduced 33.3%**

ER.2 Works delivered after 72 hours but BEFORE next Monday (i.e. until the Sunday) will be **reduced 50%**. All other delivered will not be considered.

GENERAL MAP OF PROBLEMS AND SOLUTIONS.

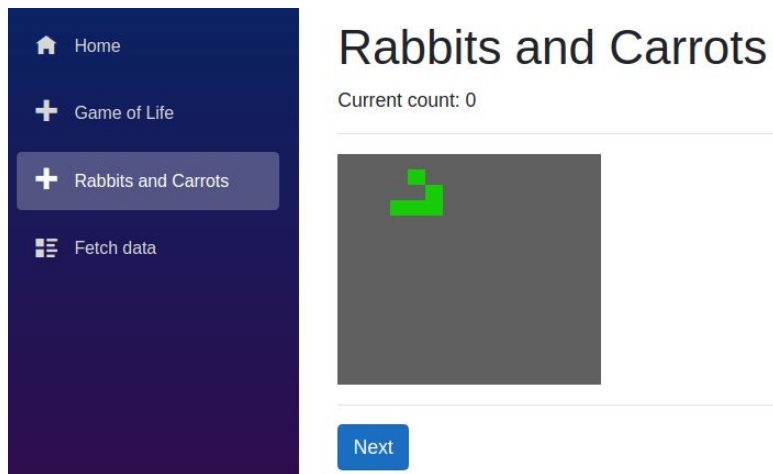
N°	%	Problem
1	10%	You must show the previous Game of Life Working, BUT with a different pattern.
2	15%	You must add a menu option for the New Environment of Rabbits and Carrots
3	25%	You must change the class Environment. The focus is very different. Now some spaces can be null cells (no objects in positions). The first versions of Rabbit and Carrots will be created. But there are not 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 <code>neighbors</code> 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 ONE "still life" pattern and ONE "oscillator pattern" in the same Environment. Required:

- 1) Screenshot of the fragment of the code generating the patterns
- 2) 2 Screenshot of the output, "as is" and after the first click.

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:

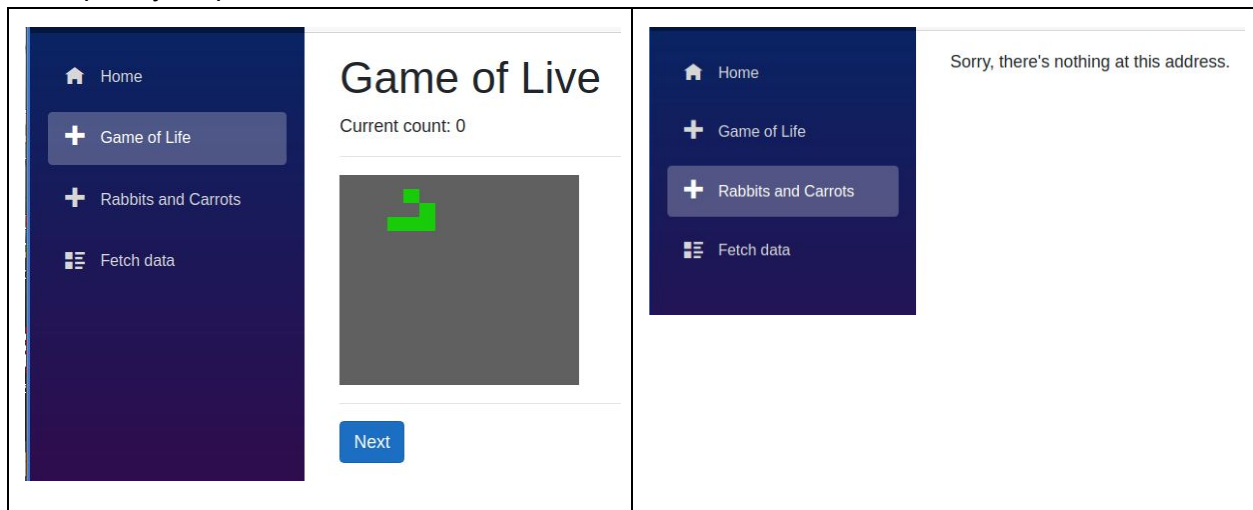


In NavMenu.razor copy and paste the option of "Game of Life", change the href attribute and the name of the option.

NavMenu.razor

```
19 </li>
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>
25 <li class="nav-item px-3">
```

A temporary output should be as follows

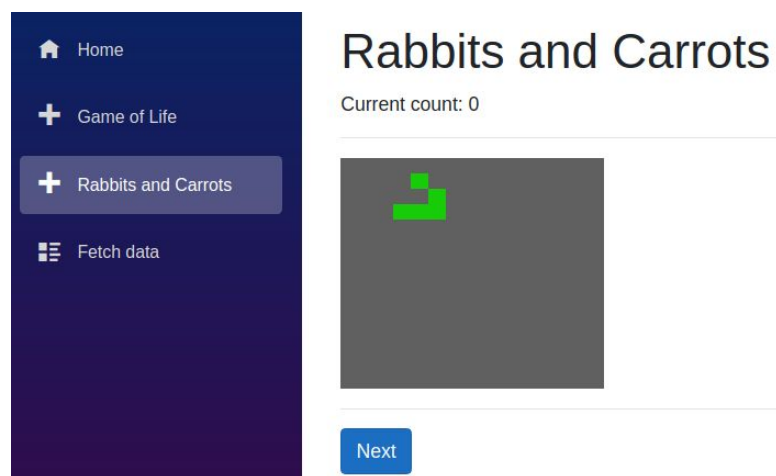


Copy Counter.razor as Rabbits_and_carrots.razor and change the @page line

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:

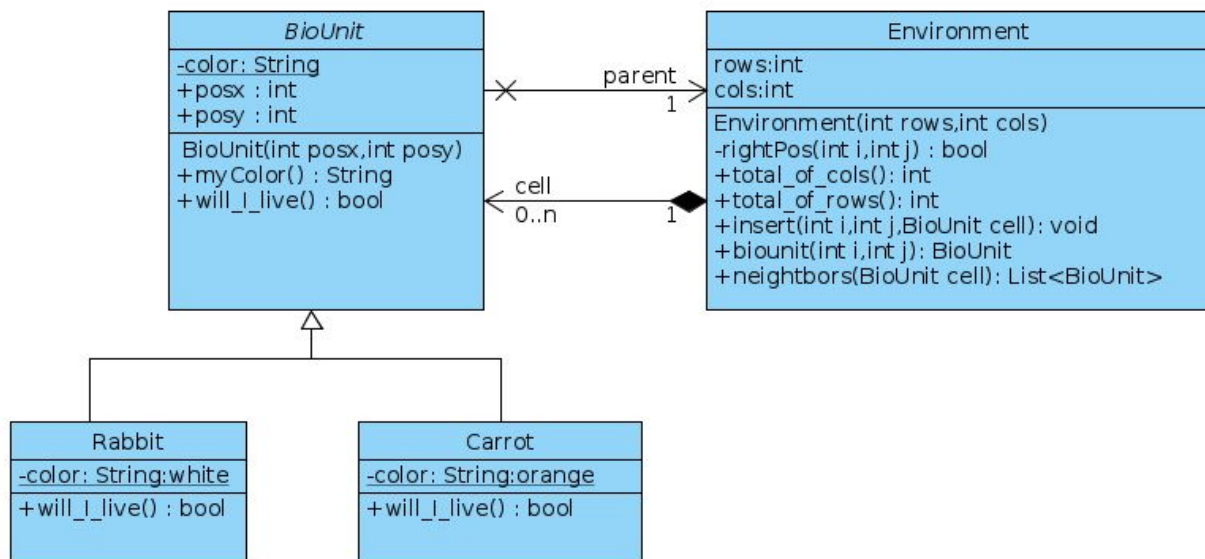


Required:

- (1) Screenshot of 5 lines of the file Rabbit_and_carrots.razor
- (2) Screenshot of your output ON the option Rabbits and Carrots

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.



This is the suggested version of Environment class

```
4 public class Environment
5 {
6     5 references
7     private int rows = 1;
8     5 references
9     private int cols = 1;
10    4 references
11    private BioUnit[,] cell;
12    2 references
13    public Environment(int rows_,int columns_) {
14        this.rows = rows_;
15        this.cols = columns_;
16        this.cell = new BioUnit[this.rows,this.cols];
17        for(var i=0; i<this.rows; i++)
18        for(var j=0; j<this.cols; j++)
19            this.cell[i,j] = null; // <--- said null
20    }
21    2 references
22    public int total_of_rows(){
23        return this.rows;
24    }
25    2 references
26    public int total_of_cols() {
27        return this.cols;
28    }
29    2 references
30    private bool rightPos(int i,int j){
31        return ((i>=0) && (i<this.rows) && (j>=0) && (j<this.cols));
32    }
33    8 references
34    public void insert(int i,int j,BioUnit been) {
35        if(this.rightPos(i,j)){
36            this.cell[i,j] = been;
37        }
38    }
39    2 references
40    public BioUnit biounit(int i,int j) {
41        if(this.rightPos(i,j)){
42            return this.cell[i,j];
43        }
44        return null;
45    }
46 }
```

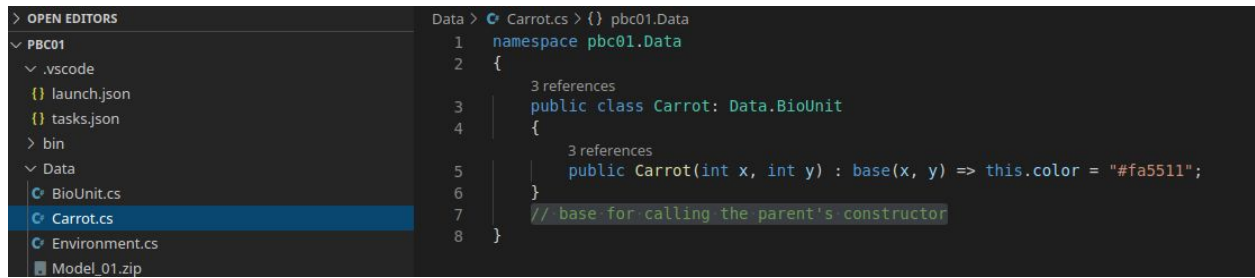
The previous Conway's step method is commented.

```
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  {
3      10 references
4      public class BioUnit
5      {
6          4 references
7          protected string color;
8          1 reference
9          public int posx;
10         1 reference
11         public int posy;
12         2 references
13         public BioUnit(int x,int y) {
14             this.posx = x;
15             this.posy = y;
16             this.color = "#444444";
17         }
18         0 references
19         public string myColor() => this.color;
20     }
21 }
```

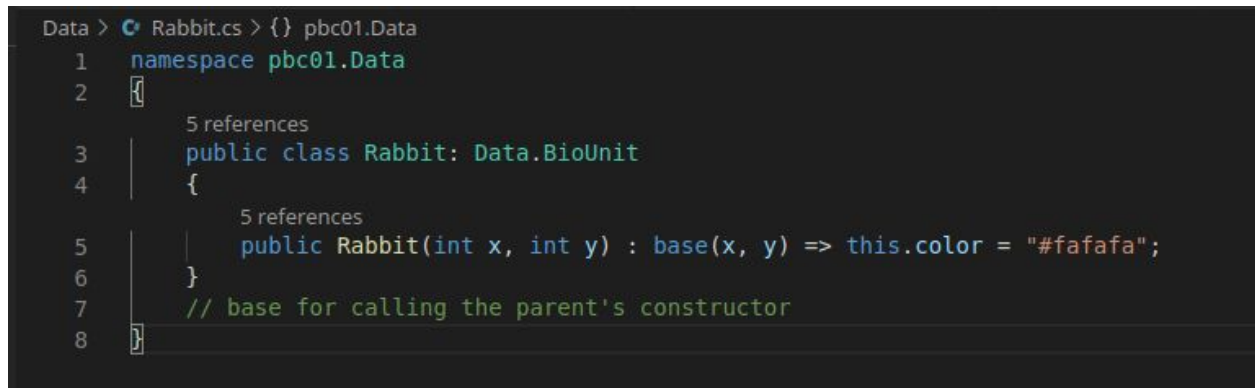
The class Carrot



The screenshot shows the Visual Studio Code editor with the file explorer on the left and the code editor on the right. The file explorer shows the project structure with folders like .vscode, tasks.json, bin, and Data. The Data folder is expanded, showing files like BioUnit.cs, Carrot.cs (selected), Environment.cs, and Model_01.zip. The code editor shows the Carrot.cs file with the following code:

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

The class Rabbit



The screenshot shows the Visual Studio Code editor with the file explorer on the left and the code editor on the right. The file explorer shows the project structure with folders like .vscode, tasks.json, bin, and Data. The Data folder is expanded, showing files like BioUnit.cs, Carrot.cs, Environment.cs, and Model_01.zip. The code editor shows the Rabbit.cs file with the following code:

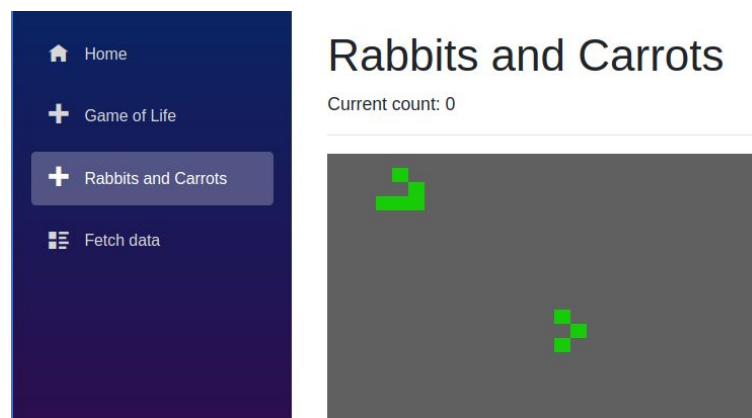
```
1 namespace pbc01.Data
2 {
3     5 references
4     public class Rabbit: Data.BioUnit
5     {
6         5 references
7         public Rabbit(int x, int y) : base(x, y) => this.color = "#fafafa";
8     }
9     // base for calling the parent's constructor
```


And a suggested version for the razor file Rabbits_and_carrots

```
Pages > Rabbits_and_carrots.razor
1  @page "/rabbits_and_carrots"
2  @using System
3
4  <h1>Rabbits and Carrots</h1>
5  <p>Current count: @currentCount</p>
6  <hr>
7  @{
8      //only the first time (initial pattern)
9      if(currentCount==0) {
10         u = new Data.Rabbit(3,3);
11         e.insert(3,3,u);
12         u = new Data.Rabbit(3,4);
13         e.insert(3,4,u);
14         u = new Data.Rabbit(3,5);
15         e.insert(3,5,u);
16         u = new Data.Rabbit(2,5);
17         e.insert(2,5,u);
18         u = new Data.Rabbit(1,4);
19         e.insert(1,4,u);
20
21
22         u = new Data.Carrot(11,14);
23         e.insert(11,14,u);
24         u = new Data.Carrot(12,14);
25         e.insert(12,15,u);
26         u = new Data.Carrot(13,14);
27         e.insert(13,14,u);
28     }
29
30 }
```

```
31 <table class="environment">
32     @for(var i=0; i<@e.total_of_rows() ; i++) {
33         <tr>
34             @for(var j=0; j<@e.total_of_cols(); j++) {
35                 @if(e.biounit(i,j)!=null) {
36                     <td class="cell alive"></td>
37                 }
38                 else
39                 {
40                     <td class="cell dead"></td>
41                 }
42             }
43         </tr>
44     }
45 </table>
46 <hr>
47 <button class="btn btn-primary"
48     @onclick="IncrementCount">Next</button>
49
50 @code {
51     3 references
52     private int currentCount = 0;
53     11 references
54     private Data.Environment e = new Data.Environment(40,40);
55     16 references
56     private Data.BioUnit u;
57     1 reference
58     private void IncrementCount()
59     {
60         currentCount++;
61     }
62 }
```

The output MUST be:



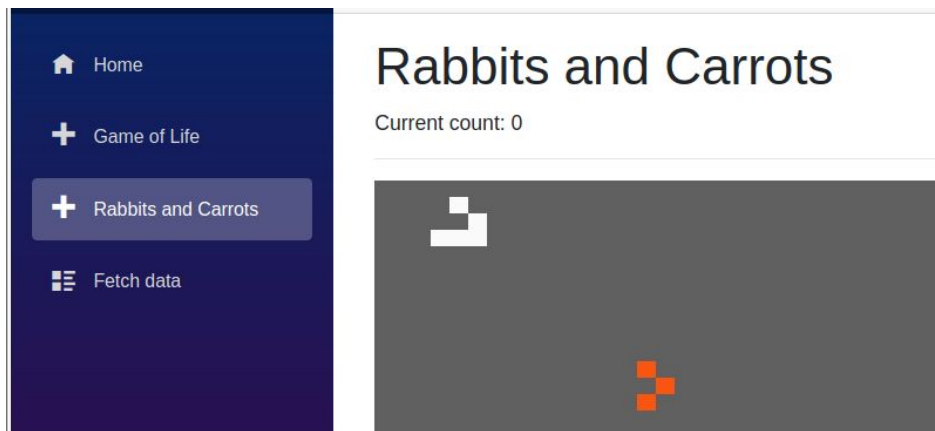
Required:

- 1) Screenshot of Carrot.cs
- 2) Screenshot of Rabbits_and_carrots.razor
- 3) Screenshot of the output

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     }
```

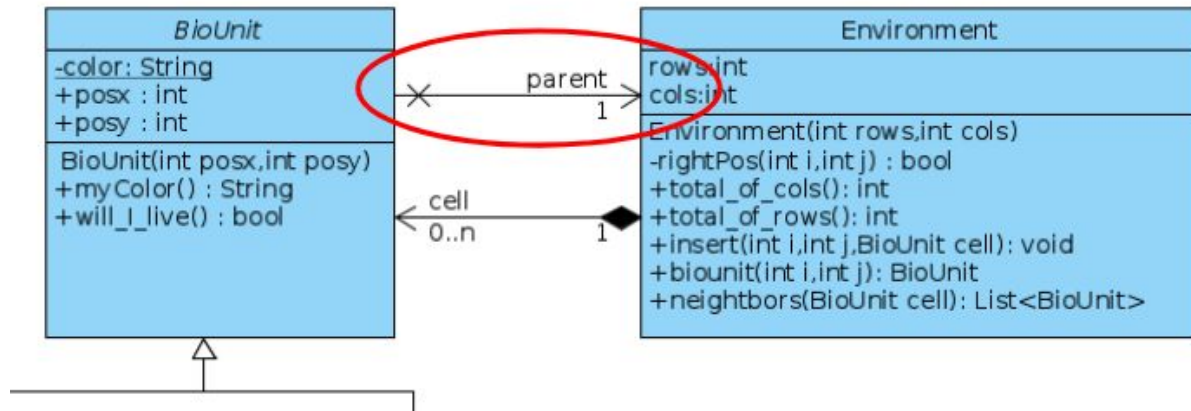
Output:



Required:

- 1) Screenshot of the code but starting at least 5 lines before.
- 2) Screenshot of the similar output

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 to navigate 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.

```

7  c{
8  //only the first time (initial pattern)
9  if(currentCount==0) {
10     u = new Data.Rabbit(3,3,e);
11     e.insert(3,3,u);
12     u = new Data.Rabbit(3,4,e);
13     e.insert(3,4,u);
14     u = new Data.Rabbit(3,5,e);
15     e.insert(3,5,u);
16     u = new Data.Rabbit(2,5,e);
17     e.insert(2,5,u);
18     u = new Data.Rabbit(1,4,e);
19     e.insert(1,4,u);
20
21     u = new Data.Carrot(11,14,e);
22     e.insert(11,14,u);
23     u = new Data.Carrot(12,14,e);
24     e.insert(12,15,u);
25     u = new Data.Carrot(13,14,e);
26     e.insert(13,14,u);
27     u = new Data.Carrot(0,0,e);
28     e.insert(0,0,u);
29
30     int count1 = e.neighbors(2,4).Count();
31     int count2 = e.neighbors(13,14).Count();
32     int count3 = e.neighbors(1,1).Count();
33     Console.WriteLine("Neighbors");
34     Console.WriteLine(count1);
35     Console.WriteLine(count2);
36     Console.WriteLine(count3);
37 }
38
39 }
```

Therefore, we need a new BioUnit's constructor

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

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

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  {
5      6 references
      public class Environment
```

Now, I show here an implementation of neighbors() 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)

Rabbits and Carrots

Current count: 0

```
watch : Exited
watch : File c
watch : Starte
Pages/Counter.
[/home/carlos/
info: Microsof
Now list
info: Microsof
Now list
info: Microsof
Applicat
info: Microsof
Hosting
info: Microsof
Content
Neighbors
5
1
1
Neighbors
5
1
1
```

Required:

- 1) Change the file Rabbits_and_carrots.razor in order to the output shows 1,2 and 3 as neighbors of some biounits.
- 2) Screenshot of the changed lines in Rabbits_and_carrots.razor
- 3) Screenshot of the output considering the terminal with the corresponding output (1,2 and

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 found 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 start again 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 either it has, at least, two surrounding rabbits. (b) The space will have a carrot, by the density of seeds, either 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 to 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 |         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:

```
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 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      6 references
7      public class Environment
8      {
9          8 references
10         private int rows = 1;
11         8 references
12         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 surroundingNeighbors(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

```

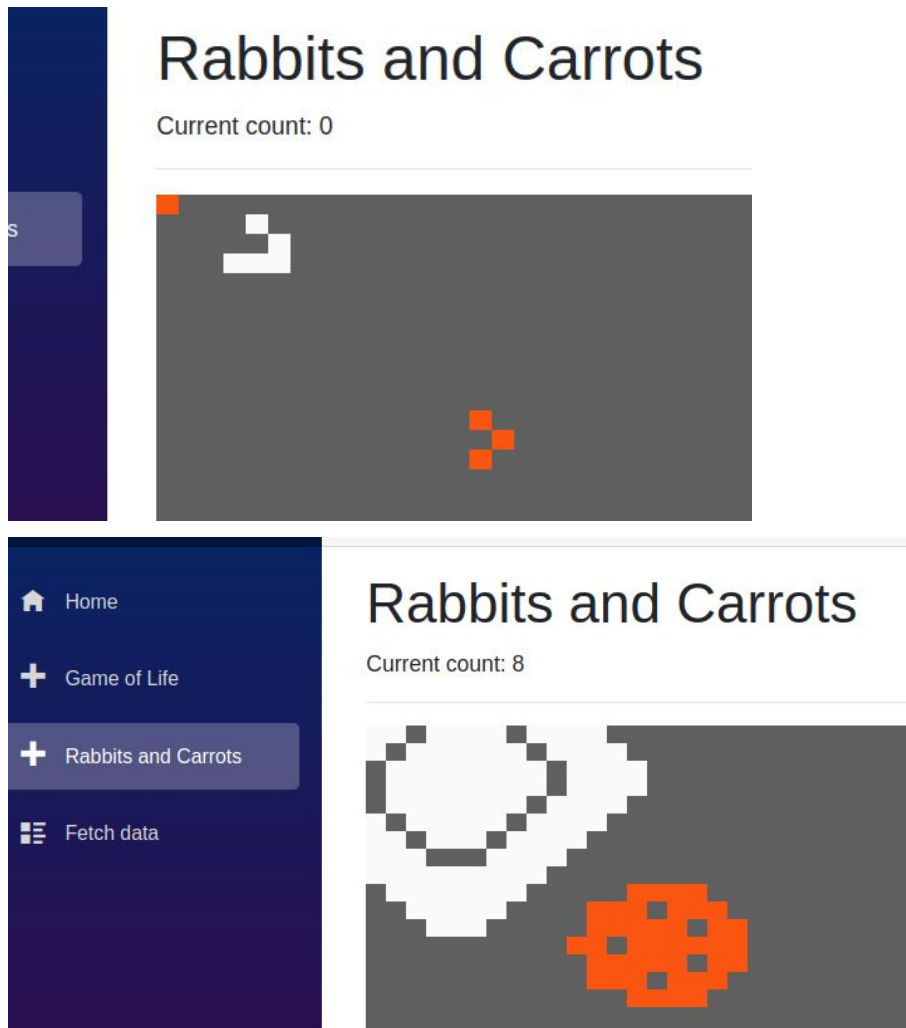
To obtain the first rabbit in the list of neighbors the method firstRabbit was implemented. Here is used a cast for saying that the object unit should be returned as a member of the class Rabbit.

```
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 step implementation 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.surroundingNeighbors(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.surroundingNeighbors(i,j,"Rabbit")>=2)
105                             aux[i,j] = new Rabbit(i,j,this);
106                         else if (this.surroundingNeighbors(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:

- 1) The complete code of the firstRabbit method
- 2) The complete code of the next_Rabbit_Carrot_step
- 3) The function that calls next_Rabbit_Carrot_step in the razor file
- 4) The output (screenshot) with no clicks
- 5) The output (screenshot) after 8 clicks