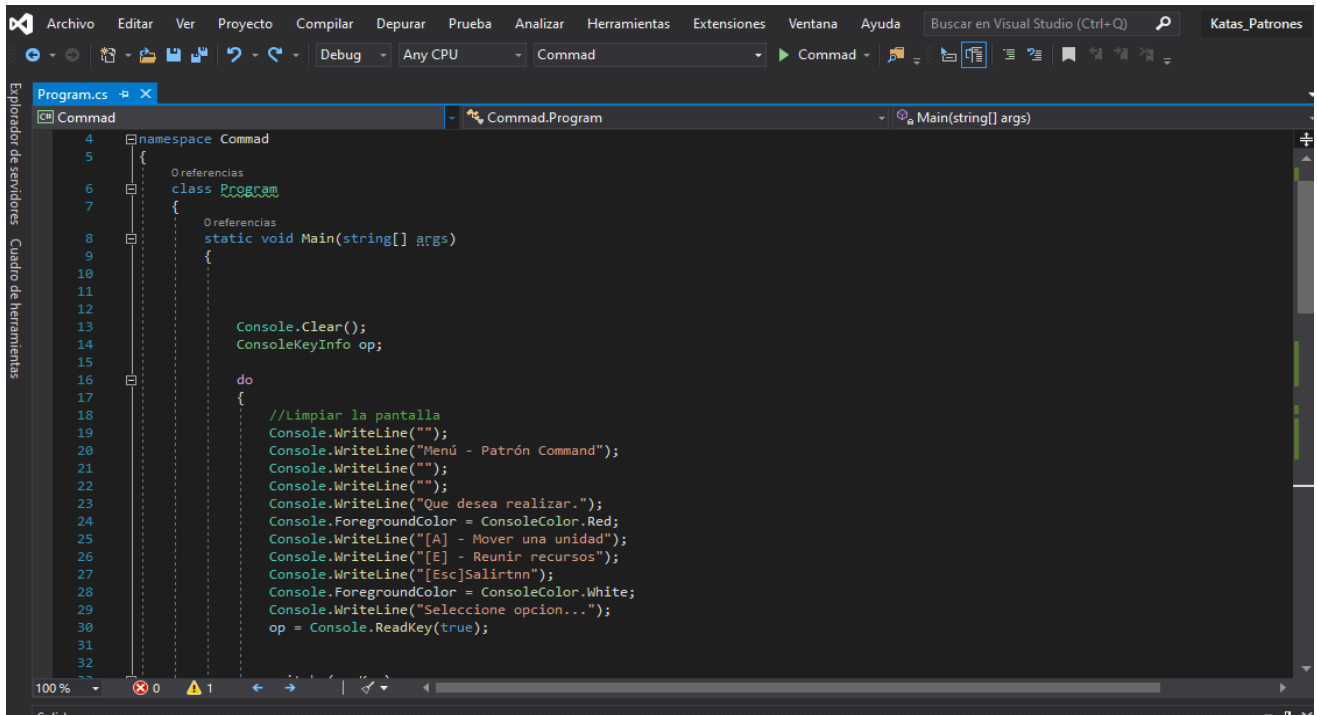
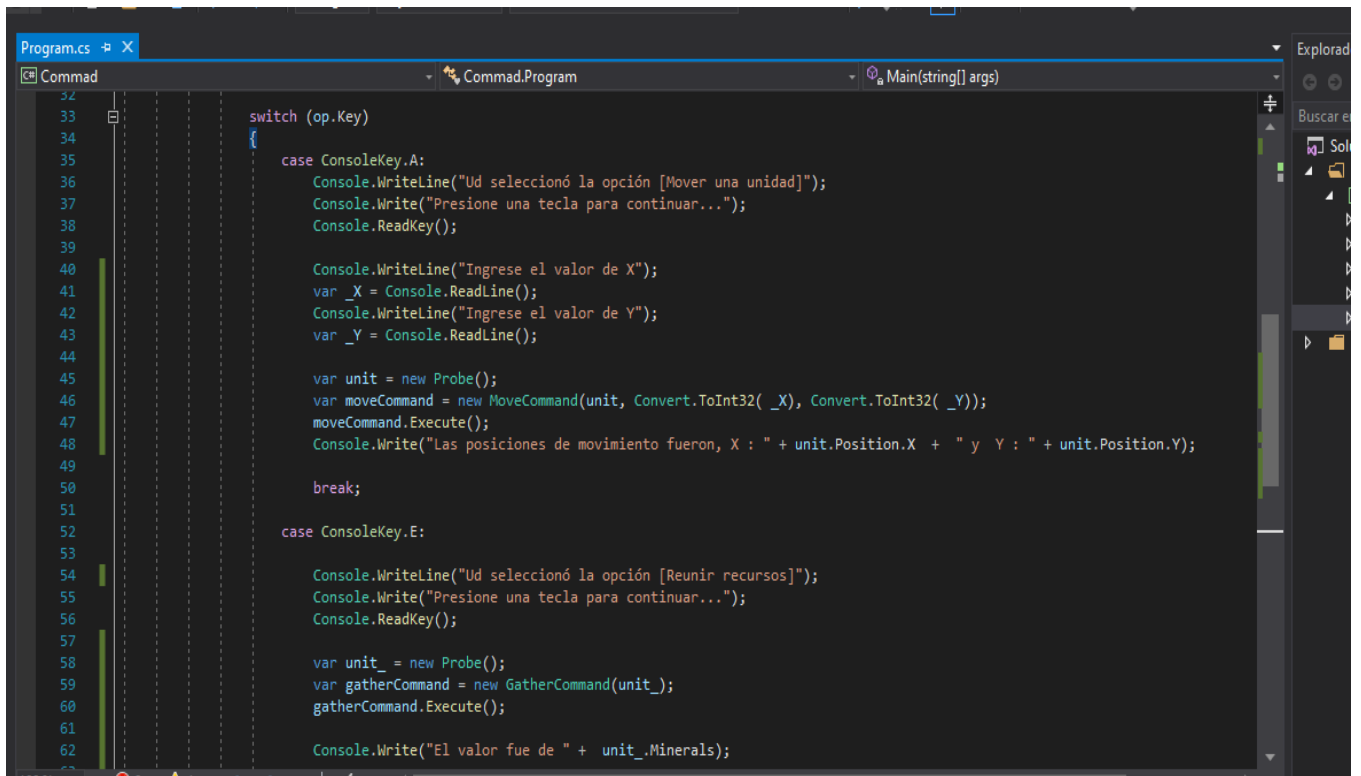


Solución:

Clase Program

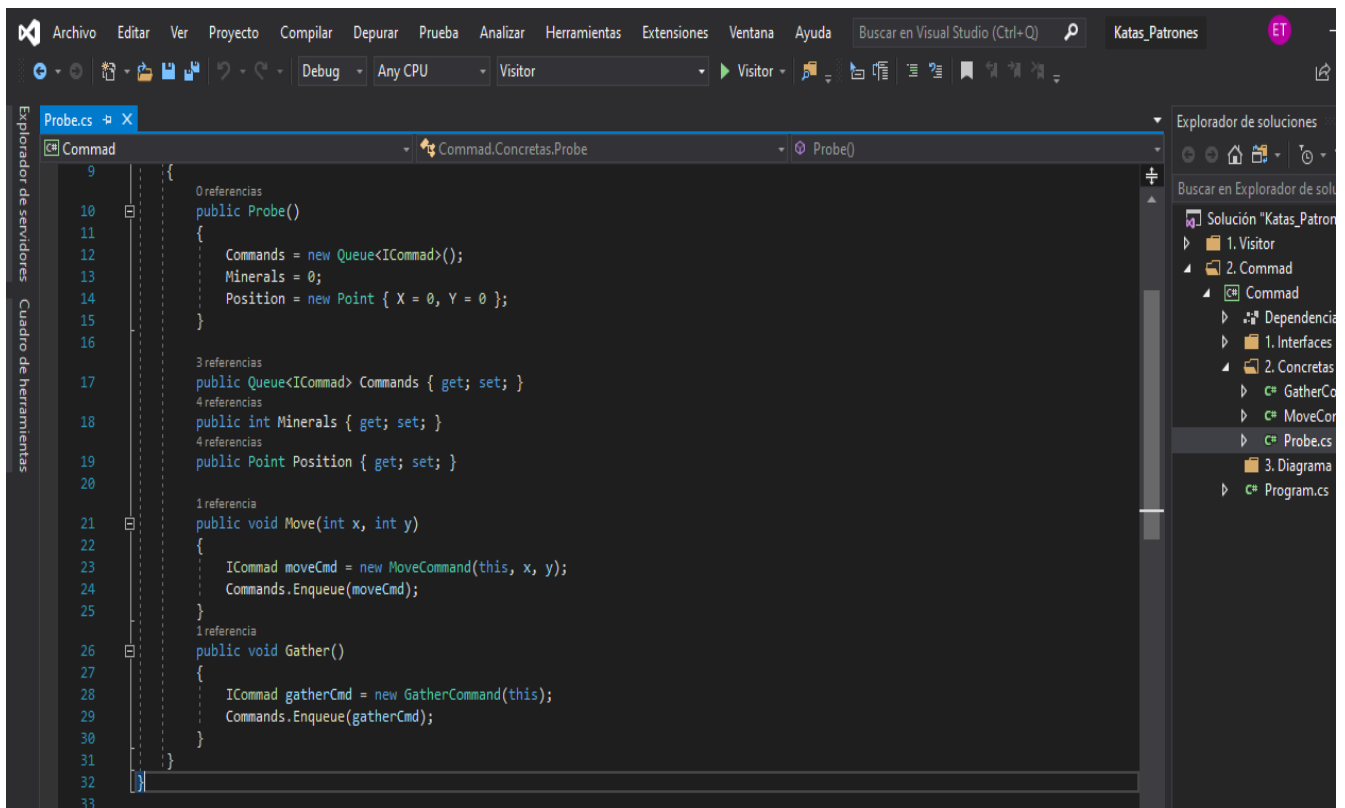


```
1 namespace Command
2 {
3     //Referencias
4     class Program
5     {
6         //Referencias
7         static void Main(string[] args)
8         {
9
10
11
12
13             Console.Clear();
14             ConsoleKeyInfo op;
15
16             do
17             {
18                 //Limpiar la pantalla
19                 Console.WriteLine("");
20                 Console.WriteLine("Menú - Patrón Command");
21                 Console.WriteLine("");
22                 Console.WriteLine("");
23                 Console.WriteLine("Que desea realizar.");
24                 Console.ForegroundColor = ConsoleColor.Red;
25                 Console.WriteLine("[A] - Mover una unidad");
26                 Console.WriteLine("[E] - Reunir recursos");
27                 Console.WriteLine("[Esc] Salir");
28                 Console.ForegroundColor = ConsoleColor.White;
29                 Console.WriteLine("Seleccione opción...");
30                 op = Console.ReadKey(true);
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```



```
101 switch (op.Key)
102 {
103     case ConsoleKey.A:
104         Console.WriteLine("Ud seleccionó la opción [Mover una unidad]");
105         Console.WriteLine("Presione una tecla para continuar...");
106         Console.ReadKey();
107
108         Console.WriteLine("Ingrese el valor de X");
109         var _X = Console.ReadLine();
110         Console.WriteLine("Ingrese el valor de Y");
111         var _Y = Console.ReadLine();
112
113         var unit = new Probe();
114         var moveCommand = new MoveCommand(unit, Convert.ToInt32(_X), Convert.ToInt32(_Y));
115         moveCommand.Execute();
116         Console.WriteLine("Las posiciones de movimiento fueron, X : " + unit.Position.X + " y Y : " + unit.Position.Y);
117
118         break;
119
120     case ConsoleKey.E:
121
122         Console.WriteLine("Ud seleccionó la opción [Reunir recursos]");
123         Console.WriteLine("Presione una tecla para continuar...");
124         Console.ReadKey();
125
126         var unit_ = new Probe();
127         var gatherCommand = new GatherCommand(unit_);
128         gatherCommand.Execute();
129
130         Console.WriteLine("El valor fue de " + unit_.Minerals);
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
```

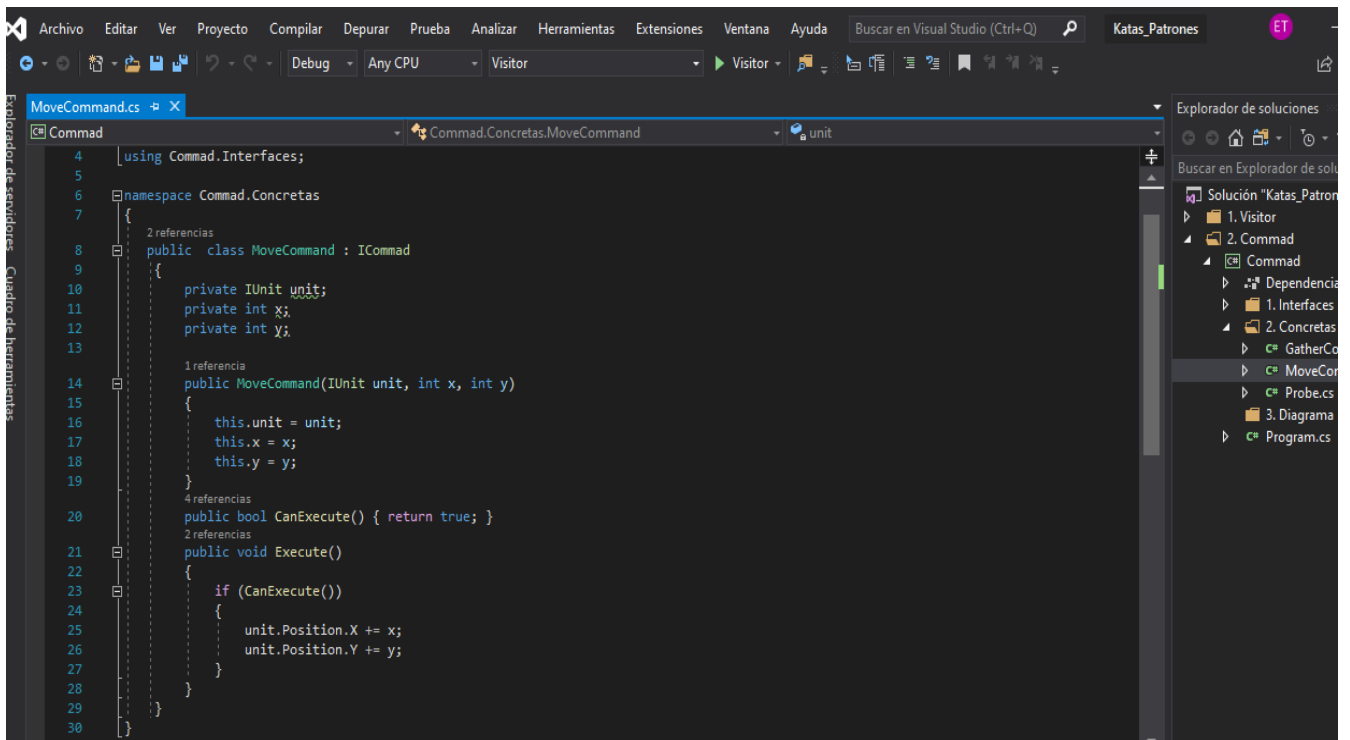
Clase Probe



The screenshot shows the Visual Studio IDE with the file `Commad.Concretas.Probe.cs` open. The code defines the `Probe` class, which implements the `ICommad` interface. The class has a constructor `Probe()` that initializes `Commands` as a `Queue<ICommad>`, `Minerals` as 0, and `Position` as a `Point` with `X = 0` and `Y = 0`. It also has three public properties: `Commands` (type `Queue<ICommad>`), `Minerals` (type `int`), and `Position` (type `Point`), each with `get` and `set` methods. Additionally, there are two public methods: `Move(int x, int y)` and `Gather()`. Both methods create a corresponding `ICommad` object (`MoveCommand` or `GatherCommand`), instantiate it with `this` and the required parameters, and then enqueue it into the `Commands` queue.

```
9 {
10     0referencias
11     public Probe()
12     {
13         Commands = new Queue<ICommad>();
14         Minerals = 0;
15         Position = new Point { X = 0, Y = 0 };
16     }
17
18     3referencias
19     public Queue<ICommad> Commands { get; set; }
20
21     4referencias
22     public int Minerals { get; set; }
23
24     4referencias
25     public Point Position { get; set; }
26
27     1referencia
28     public void Move(int x, int y)
29     {
30         ICommad moveCmd = new MoveCommand(this, x, y);
31         Commands.Enqueue(moveCmd);
32     }
33
34     1referencia
35     public void Gather()
36     {
37         ICommad gatherCmd = new GatherCommand(this);
38         Commands.Enqueue(gatherCmd);
39     }
40 }
```

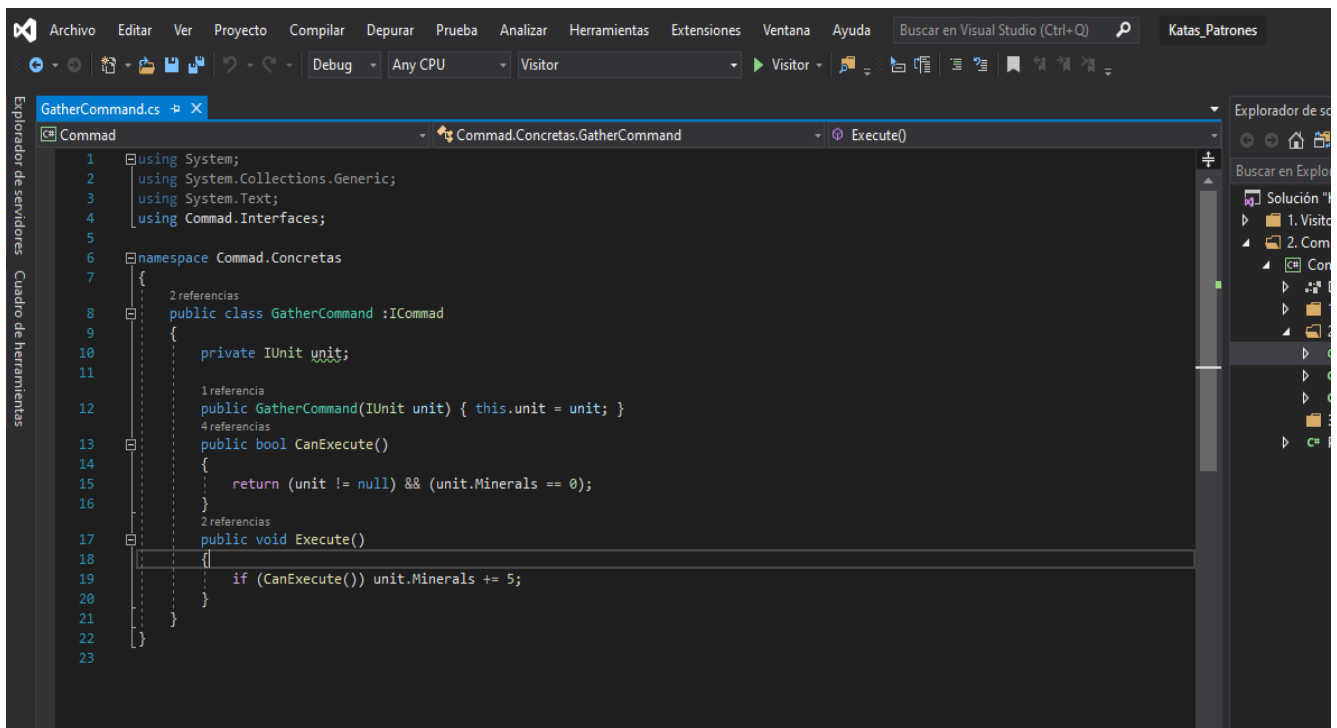
Clase MoveCommand



The screenshot shows the Visual Studio IDE with the file `Commad.Concretas.MoveCommand.cs` open. The code defines the `MoveCommand` class, which implements the `ICommad` interface. The class is nested within the `Commad.Concretas` namespace. It has three private fields: `IUnit unit`, `int x`, and `int y`. The constructor `MoveCommand(IUnit unit, int x, int y)` initializes these fields. The class also has two public methods: `CanExecute()` and `Execute()`. `CanExecute()` returns `true`. `Execute()` checks if `CanExecute()` returns `true`, and if so, it increments the `X` and `Y` coordinates of the `unit.Position` by `x` and `y` respectively.

```
4 using Commad.Interfaces;
5
6 namespace Commad.Concretas
7 {
8     2referencias
9     public class MoveCommand : ICommad
10     {
11         private IUnit unit;
12         private int x;
13         private int y;
14
15         1referencia
16         public MoveCommand(IUnit unit, int x, int y)
17         {
18             this.unit = unit;
19             this.x = x;
20             this.y = y;
21         }
22
23         4referencias
24         public bool CanExecute() { return true; }
25
26         2referencias
27         public void Execute()
28         {
29             if (CanExecute())
30             {
31                 unit.Position.X += x;
32                 unit.Position.Y += y;
33             }
34         }
35     }
36 }
```

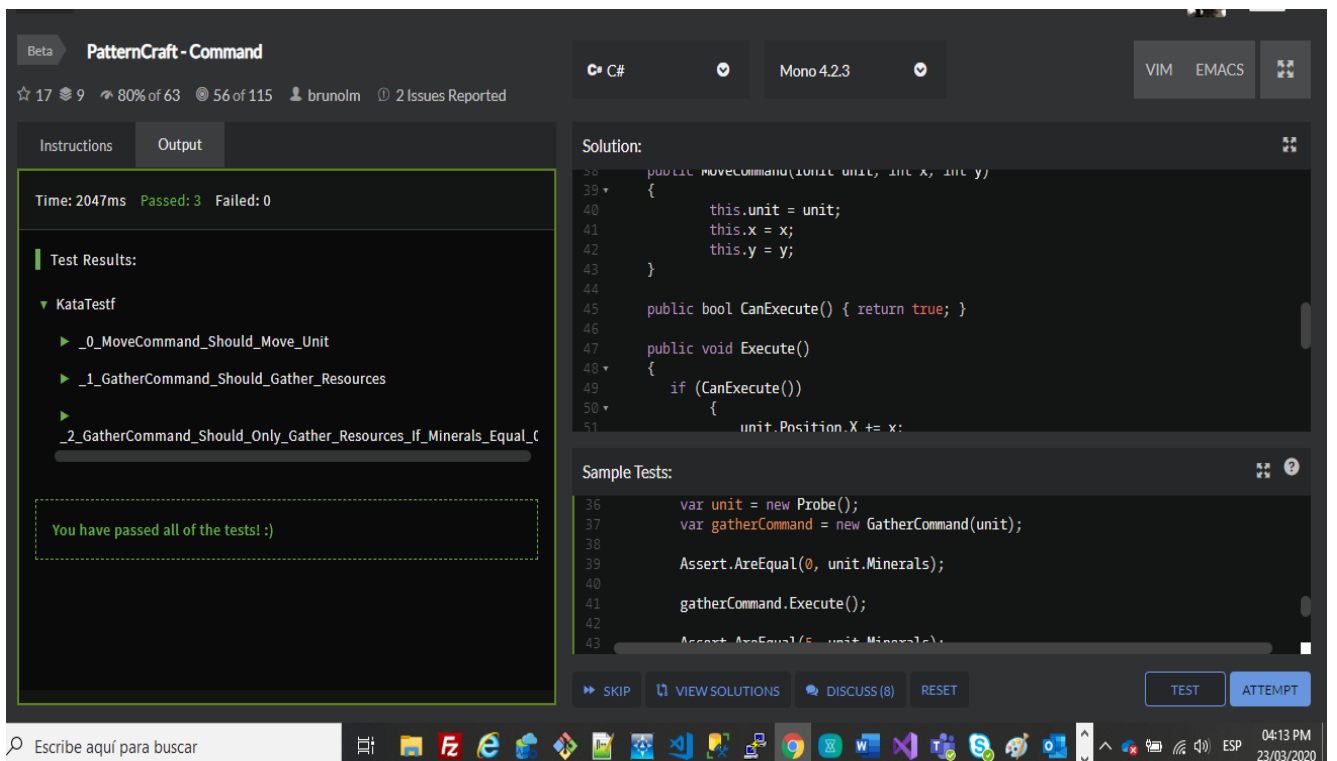
Clase GatherCommand



The screenshot shows the Visual Studio IDE with the file 'GatherCommand.cs' open. The code implements the 'GatherCommand' class, which inherits from 'ICommand'. It includes a private 'IUnit' field named 'unit'. The constructor 'GatherCommand(IUnit unit)' initializes 'this.unit' to 'unit'. The 'CanExecute()' method returns true if 'unit' is not null and 'unit.Minerals' is greater than 0. The 'Execute()' method calls 'CanExecute()' and increments 'unit.Minerals' by 5 if it returns true. The right sidebar shows the 'Explorador de soluciones' (Solution Explorer) with a tree view of the project structure.

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using Commad.Interfaces;
5
6 namespace Commad.Concretas
7 {
8     2 referencias
9     public class GatherCommand : ICommand
10     {
11         private IUnit unit;
12
13         1 referencia
14         public GatherCommand(IUnit unit) { this.unit = unit; }
15
16         4 referencias
17         public bool CanExecute()
18         {
19             return (unit != null) && (unit.Minerals > 0);
20         }
21
22         2 referencias
23         public void Execute()
24         {
25             if (CanExecute()) unit.Minerals += 5;
26         }
27     }
28 }
```

Ejecución de pruebas



The screenshot shows the 'PatternCraft - Command' interface. The top bar includes a 'Beta' label, the title 'PatternCraft - Command', and statistics: 17 stars, 9 forks, 80% of 63 tests passed, 56 of 115 tests completed, user 'brunolm', and 2 issues reported. The 'Instructions' tab is active, showing 'Time: 2047ms', 'Passed: 3', and 'Failed: 0'. The 'Test Results' section shows a list of tests under 'KataTestf': '_0_MoveCommand_Should_Move_Unit', '_1_GatherCommand_Should_Gather_Resources', and '_2_GatherCommand_Should_Only_Gather_Resources_If_Minerals_Equal_C'. A message states 'You have passed all of the tests! :)'. The 'Solution' section shows the implementation of the 'MoveCommand' class. The 'Sample Tests' section shows test code. At the bottom, there are buttons for 'SKIP', 'VIEW SOLUTIONS', 'DISCUSS (8)', 'RESET', 'TEST', and 'ATTEMPT'. The Windows taskbar is visible at the bottom with the time 04:13 PM on 23/03/2020.

Instructions **Output**

Time: 2047ms Passed: 3 Failed: 0

Test Results:

- KataTestf
 - _0_MoveCommand_Should_Move_Unit
 - _1_GatherCommand_Should_Gather_Resources
 - _2_GatherCommand_Should_Only_Gather_Resources_If_Minerals_Equal_C

You have passed all of the tests! :)

Solution:

```
38 public MoveCommand(IUnit unit, int x, int y)
39 {
40     this.unit = unit;
41     this.x = x;
42     this.y = y;
43 }
44
45 public bool CanExecute() { return true; }
46
47 public void Execute()
48 {
49     if (CanExecute())
50     {
51         unit.Position.X += x;
52     }
53 }
```

Sample Tests:

```
36 var unit = new Probe();
37 var gatherCommand = new GatherCommand(unit);
38
39 Assert.AreEqual(0, unit.Minerals);
40
41 gatherCommand.Execute();
42
43 Assert.AreEqual(5, unit.Minerals);
```

SKIP VIEW SOLUTIONS DISCUSS (8) RESET TEST ATTEMPT

Escribe aquí para buscar

04:13 PM 23/03/2020

