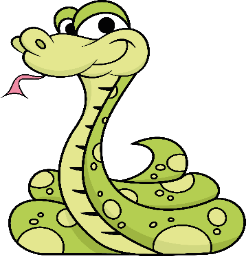
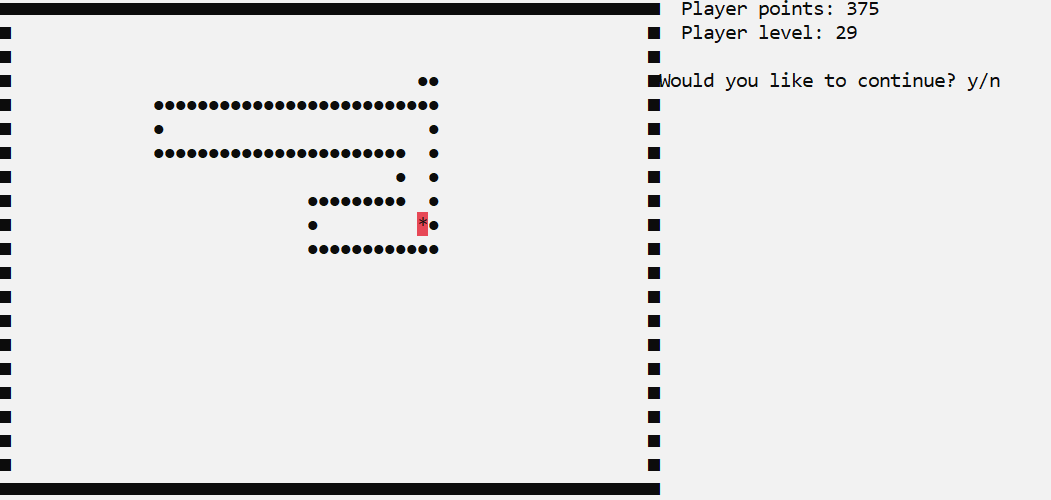
# Snake

### Overview

In this workshop, you need to build a snake game on your own, following the basic principles of OOP.

### Setup

You are provided with a **skeleton**,which contains the following items:

- StartUp class – your program entry point

- Core folder – the main program functionality

- Enums folder – data about directions for the snake

- GameObjects folder – holding the main objects for our game

- Utilities folder – already written (contains information about your ConsoleWindow)

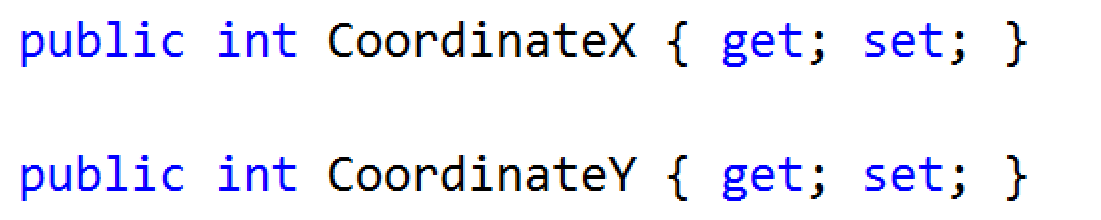
## Task 1: Structure

### Game objects

In the GameObjects folder create the following classes:

#### Coordinate

The Coordinate class contains information about the **2D space** where all the objects exist. It has two properties, which indicate the **horizontal and vertical position** of the object. You can give them the names CoordinateX and CoordinateY and both are **integers**.



#### Constructor

A Coordinate accepts the following values upon initialization:

int CoordinateX, int CoordinateY

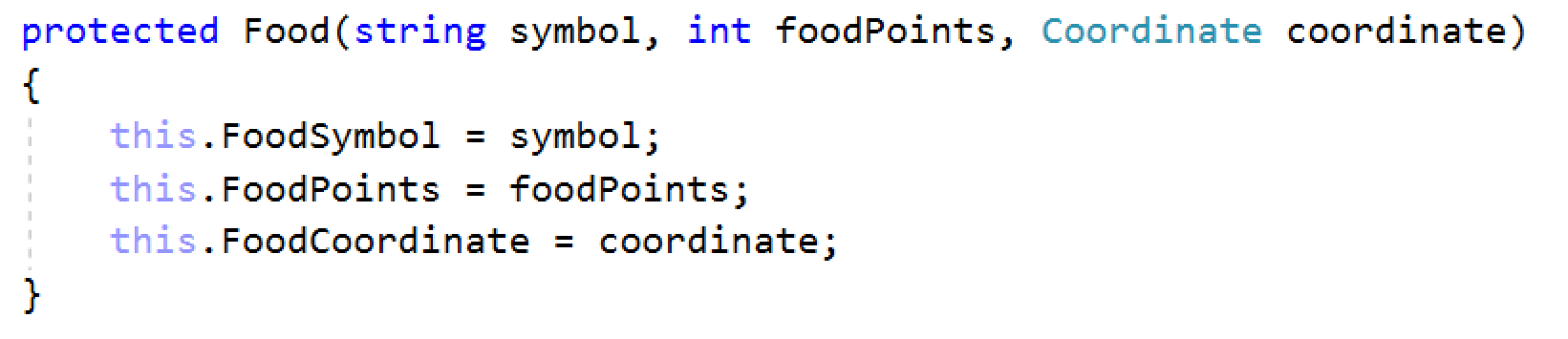
### Food

Food is an **abstract class** which implements all the logic for the types of food our game will have. We must have the following properties:

* **FoodPoints** : int
* **FoodSymbol** : string
* **FoodCoordinate** : Coordinate

#### Constructor

Your constructor should look like this:

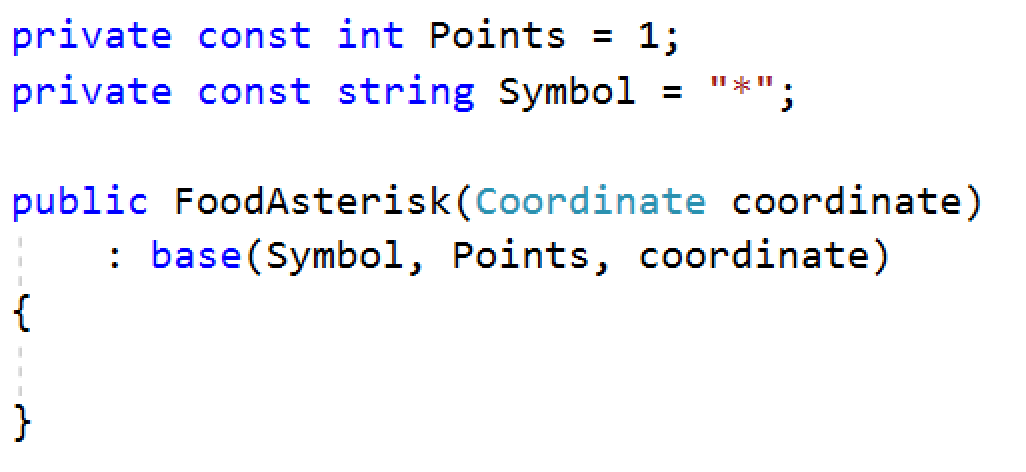


#### Child Classes

There are several concrete types of **food**:

* FoodAsterisk
* FoodDollar
* FoodHash

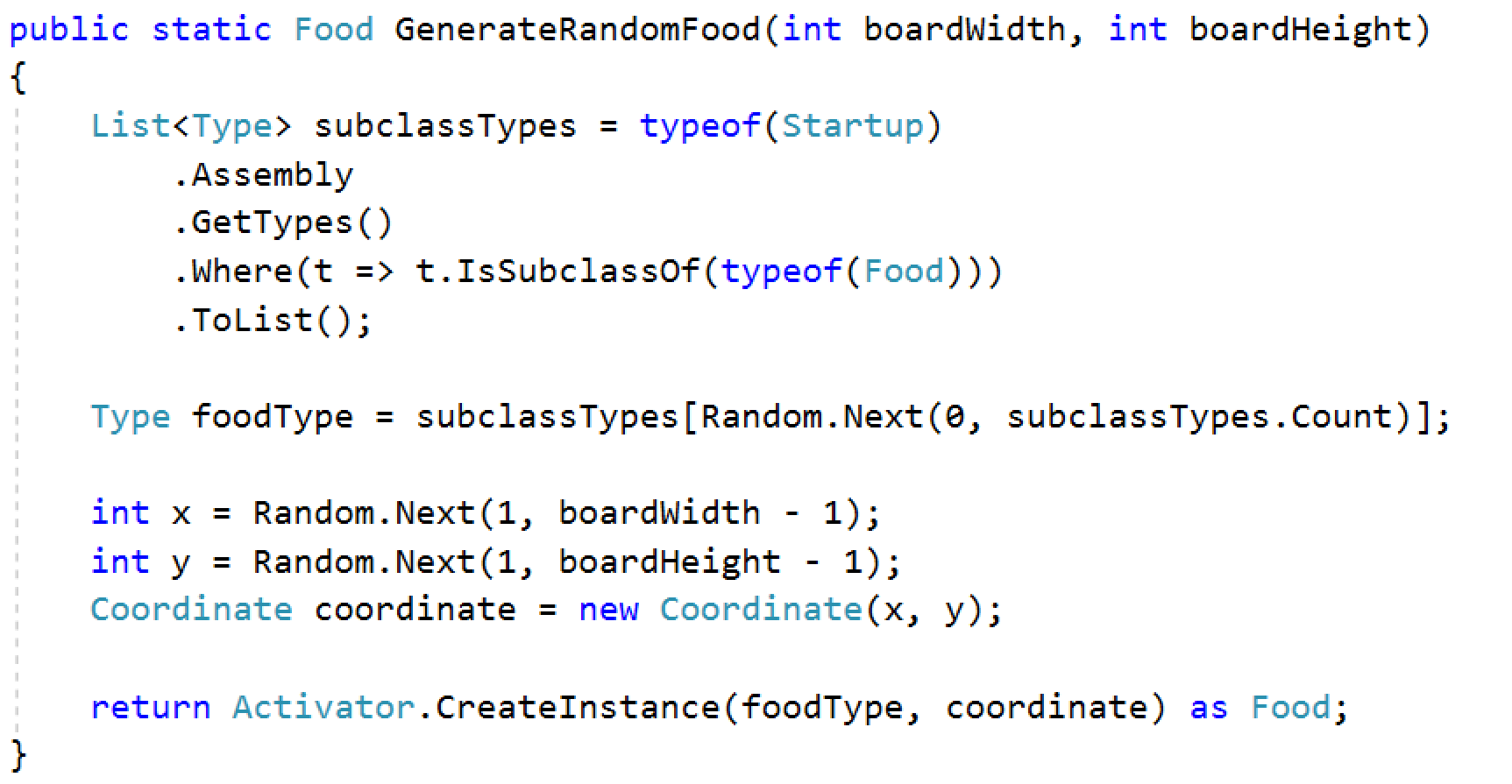
FoodAsterisk inherits Food and its constructor. It has two constant values. One for the food symbol (“\*”) and one for the points it gives to the snake – **1**



Implement the same logic for the other types of Food. The FoodDollar has “**$**” symbol and gives **2** points. The FoodHash has “#” symbol and gives **3** points.

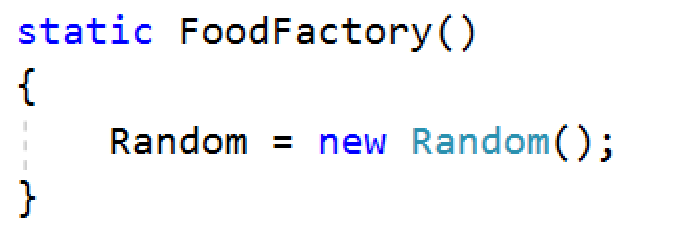
### Food Factory

Let's create a factory for our food. Create static class **FactoryFood** and inside create static method **GenerateRandomFood.** This method generates random coordinates and random food. To generate a random coordinates for our food we have to get the max coordinateX and max coordinateY, which in our case is the board width and height. To generate random type food we have to get all types (use reflection). If you managed to return food using reflection, your code should look like this:



#### Constructor

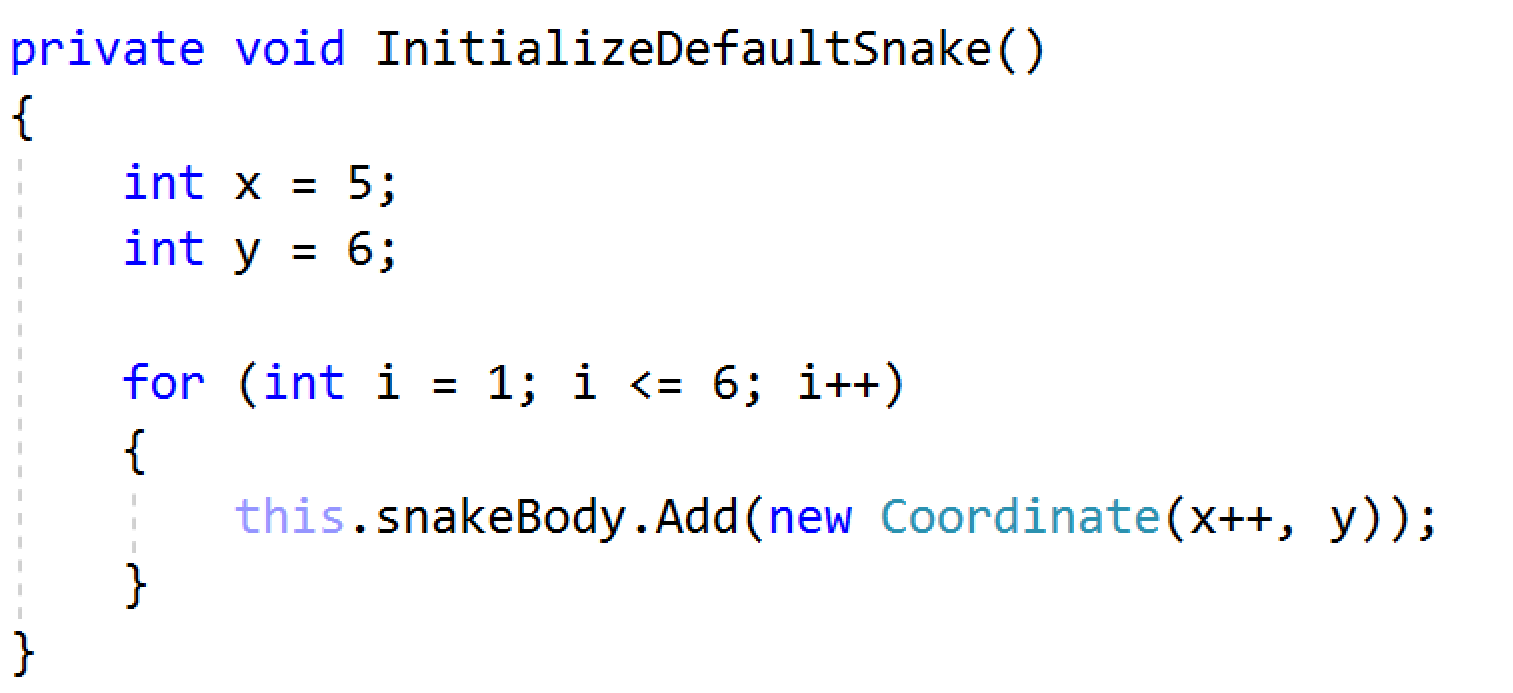
In the factory constructor only create an instance of the random class.



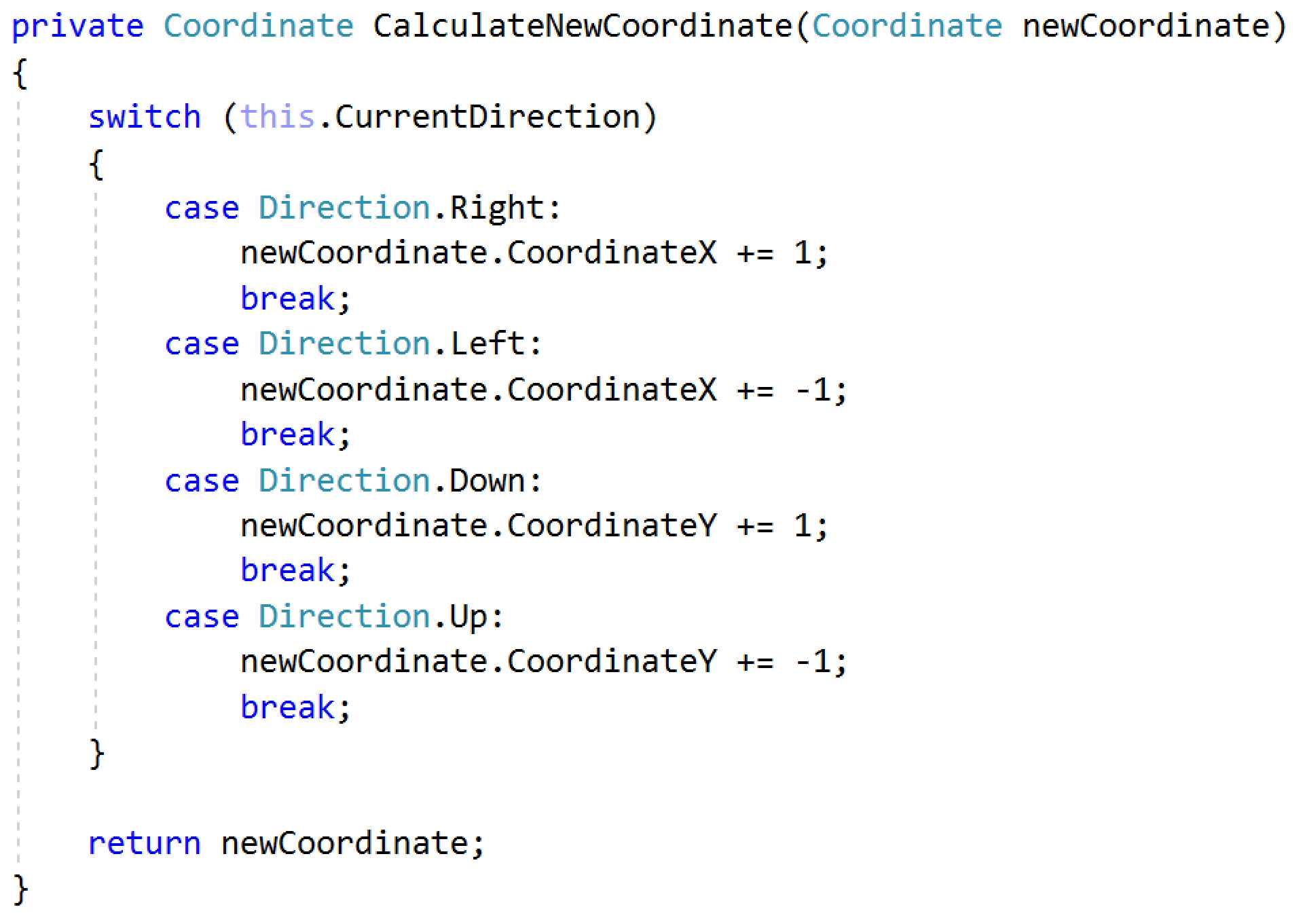
**Hint**: The field should also be static.

### Snake

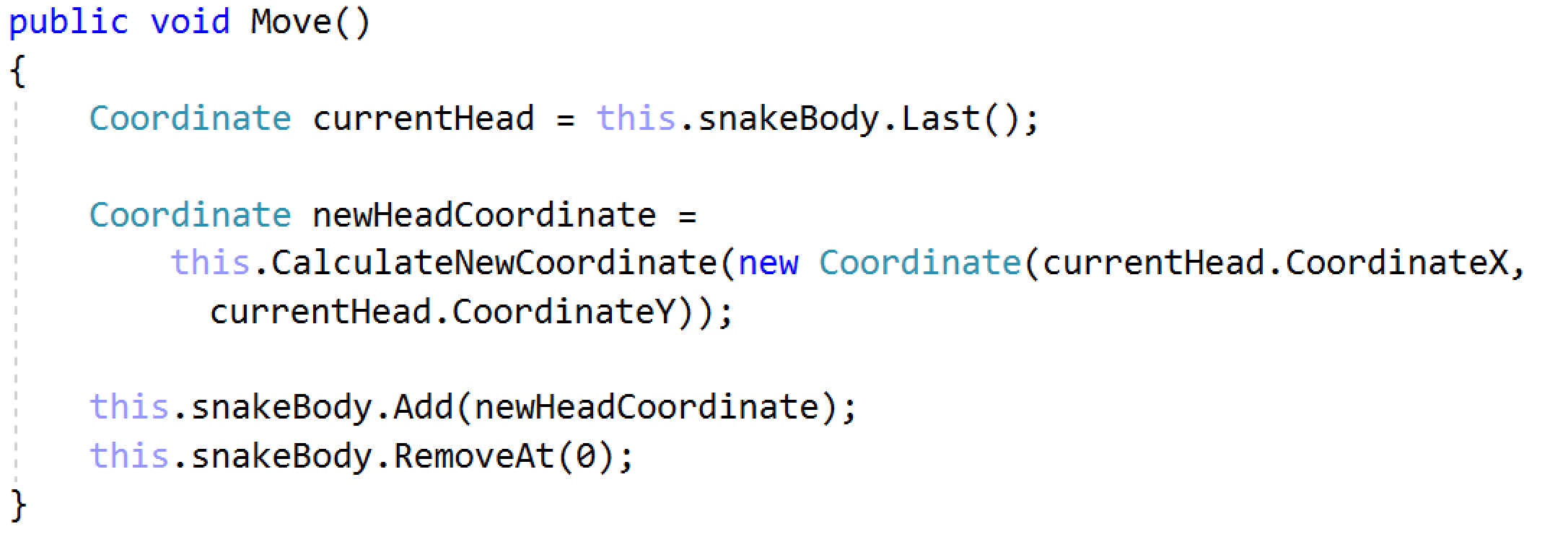
TheSnake class is a class, which is used to design, control movement and define Snake behavior. Here we are going to implement methods for the movement of the snake and the types of food she eats. Let’s create our snake. Create a field, which holds all point positions of the snake parts. Declare **InitializeDefaultSnake** method which iterates from **1** to **6** (this is our snake’s starting length) and each time, adds a new Coordinate in its list of snake parts.



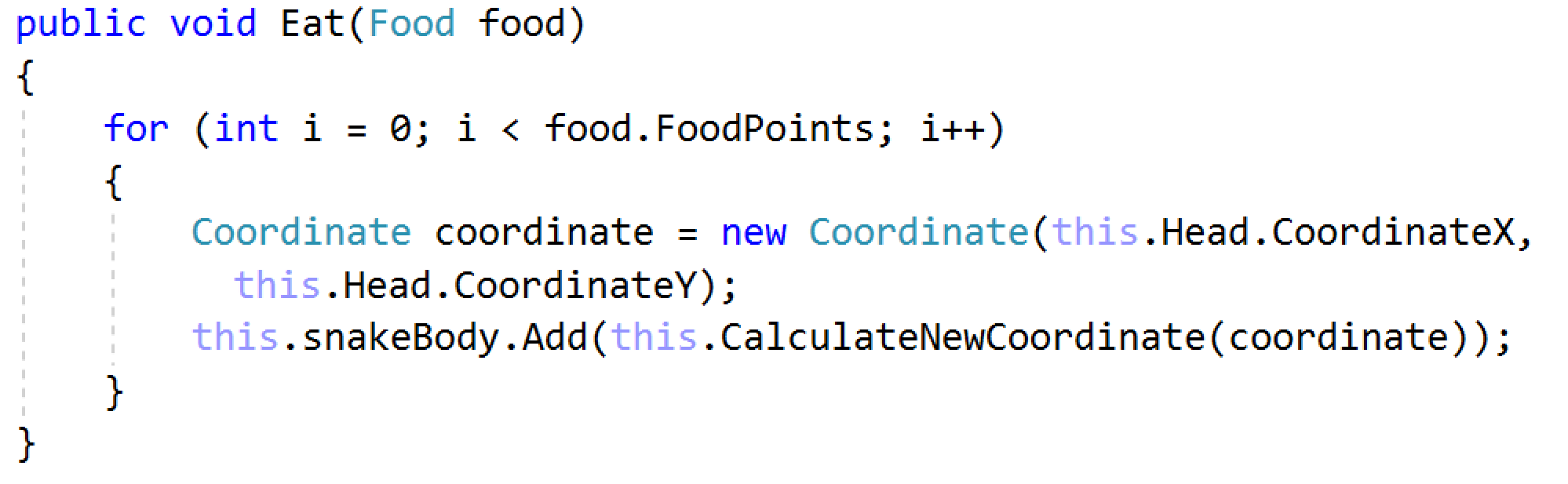
For now, our snake can’t move. Let’s change that. Create a void method Move. To move the snake, we must first find its head. This happens when we take the **last element** of our list of snake parts. We have to check which is the next position of the snake. That’s why we are going to implement the CalculateNewCoordinate method, which accepts a **Coordinate** of the snake’s head’s position and returns new coordinate of the head. All this method does is to get the next coordinates of the snake head, depending on the direction. For now the snake does not now about the direction which is moving. Let's make a property **CurrentDirection** of type **Direction. Depending on the direction and the next Head Coordinate will be changed differently.** For example if our current direction is **RIGHT** we have to **increase** the **coordinateX** with **1.**



After that, in our Move method, we have to add the coordinate that CalculateNewCoordinate method has returned and remove the first element of the snake body. In the end the **Move** method should look like this:

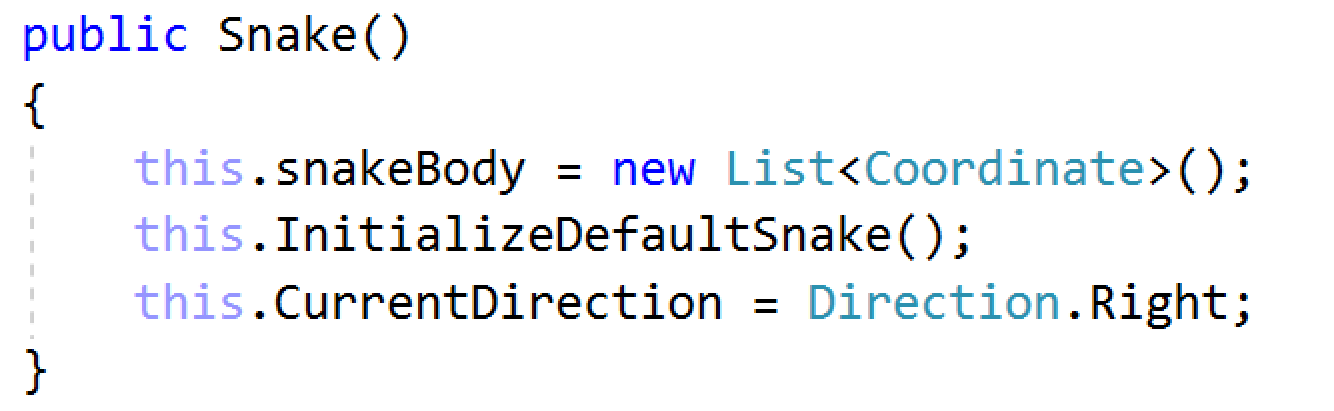


Now we have to give the snake the ability to eat, so we have to create the **Eat** method which will accept Food as parameter. After the snake eats some food, she grows with the points of the food.



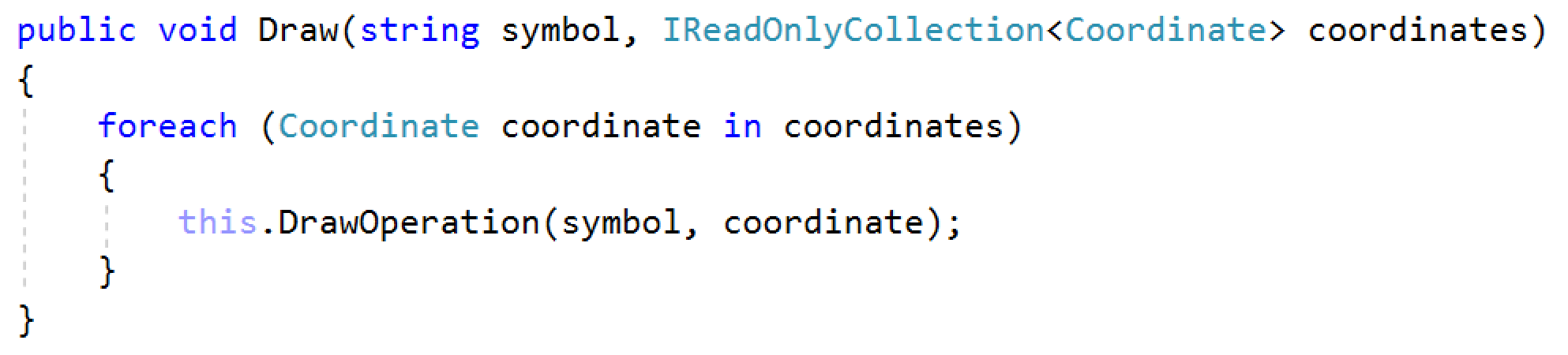
#### Constructor

If you have managed to implement this class, your constructor should look like this:

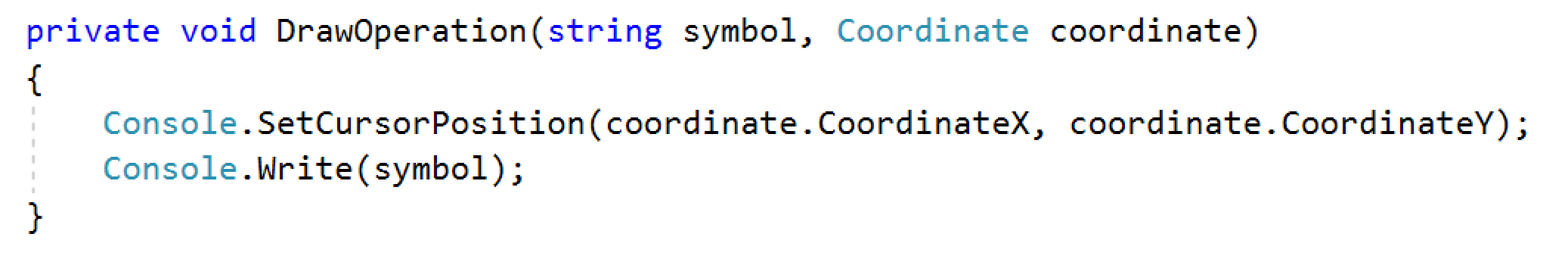


### Draw Manager

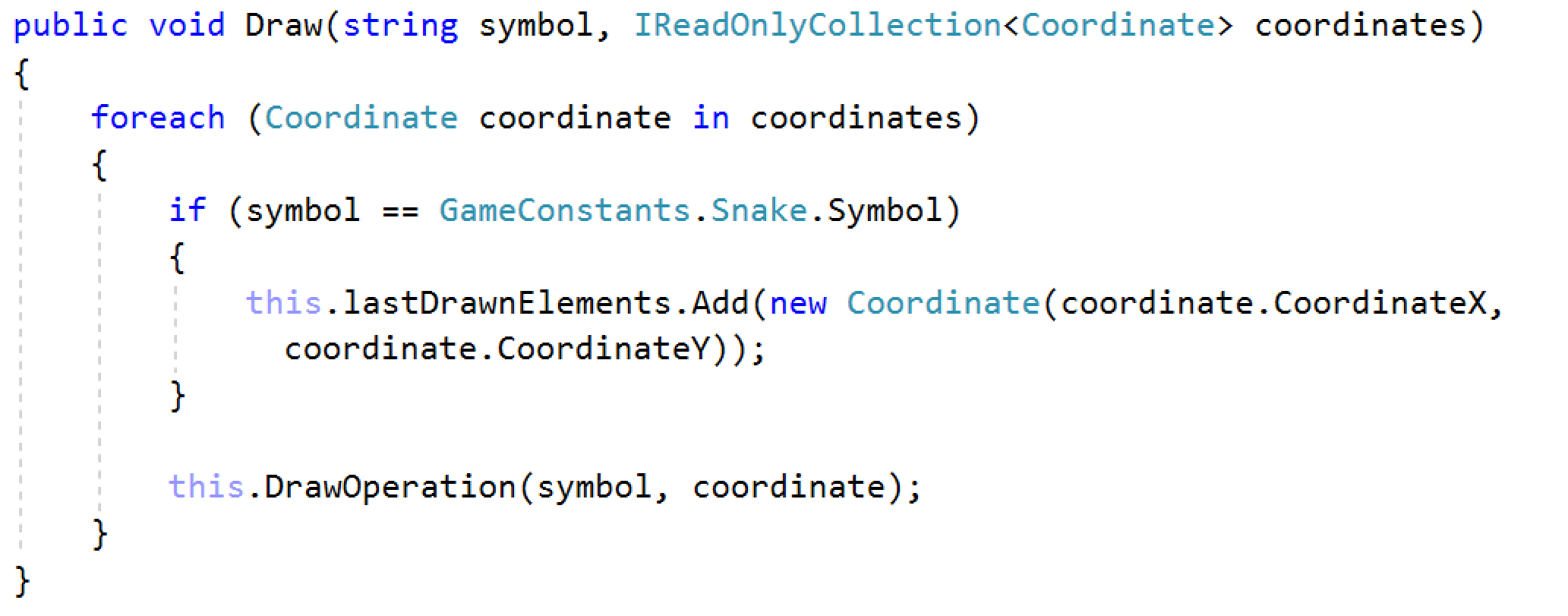
Our draw manager will be responsible for the drawing on the console. Now we have to create a **Draw** method to visualize the snake and the food. Our Draw method will need symbol and collection of coordinates to do the job. Go through all the coordinates and visualize them.



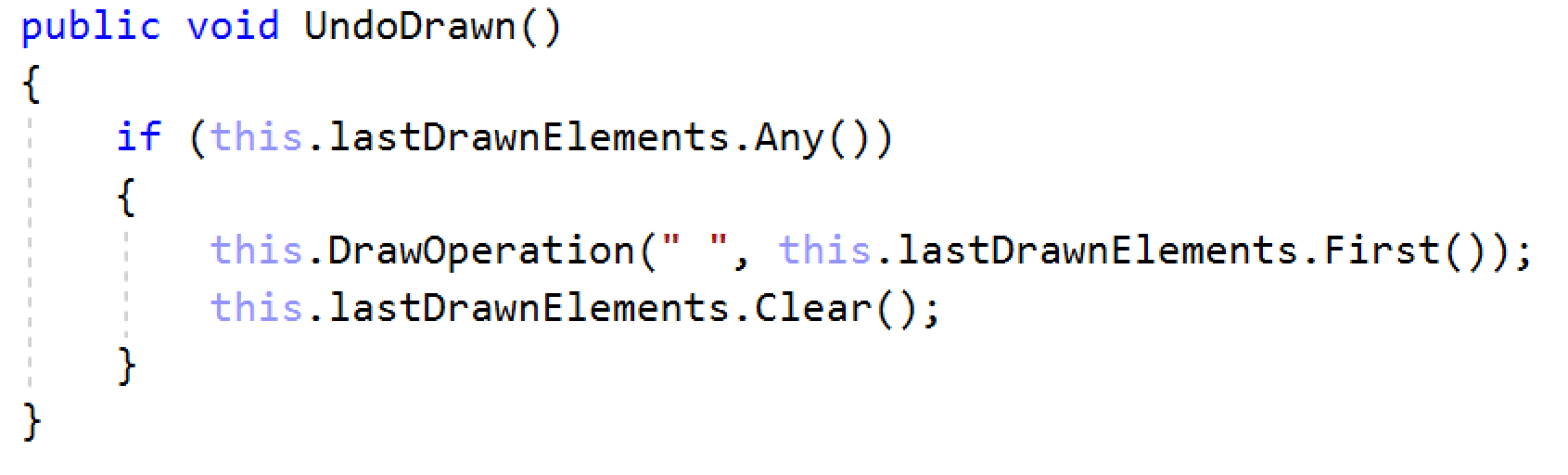
The **DrawOperation** will draw a symbol on given coordinate.



So far so good. We managed to achieve the drawing of our snake on the console, but we have to make it move. Create a method called **UndoDraw.** This method should remove the first point of the body from the console. To achieve that we have to keep track of the snake body coordinates. Create a field which will contain list of Coordinates. We have to **modify** our **Draw** method for our purposes. While we are drawing on the console we have to save the snake coordinates. In the end the Draw method looks like this:



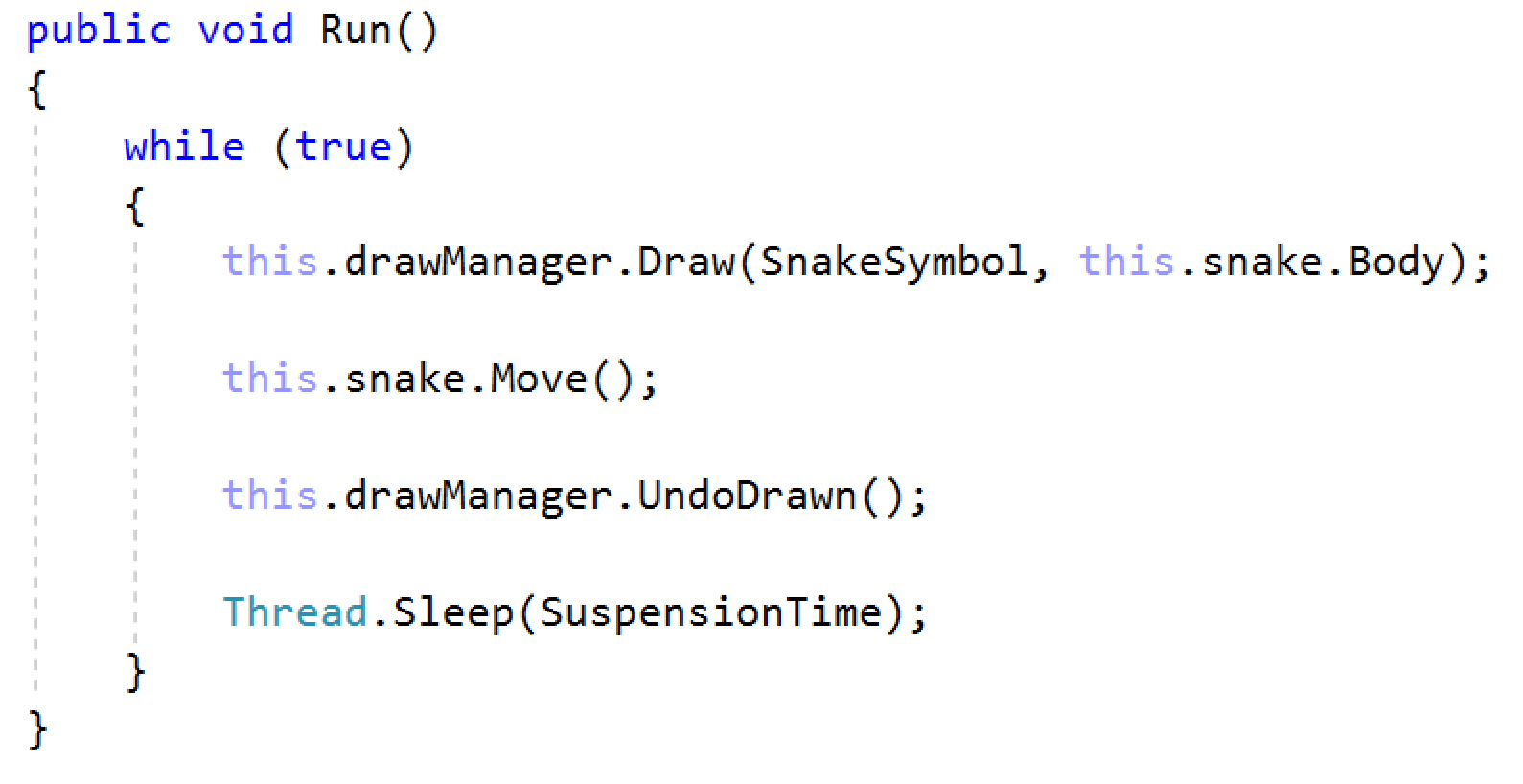
And finally in **UndoDraw** method replace the first snake symbol with space and clear our collection with coordinates.



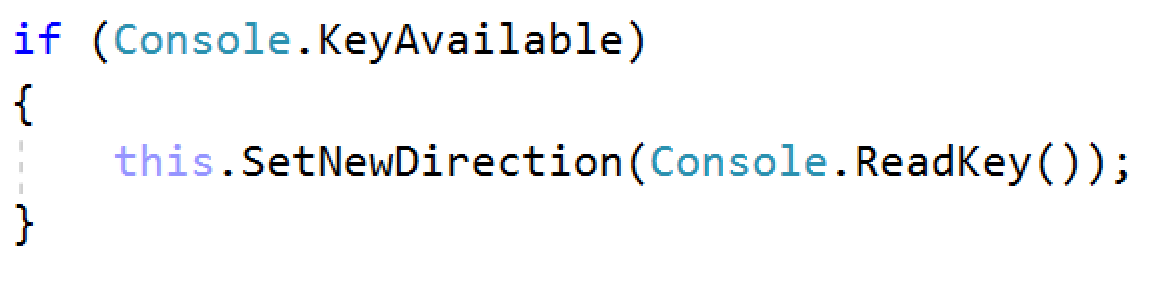
### Game Engine

Our engine class will be responsible for the user interface. This class has the most complex logic. The engine will take care of all the buttons clicked by the user and visualize all the logic we have already written. It will have only one public method Run() which will do the main logic for the application.

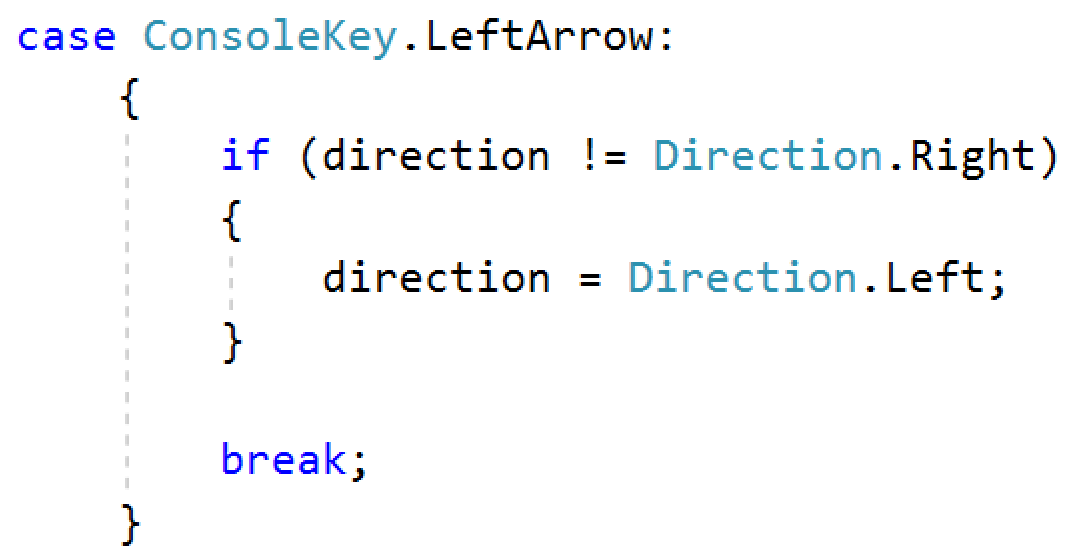
Our engine must have fields for our snake and draw manager for start to make our snake move. In our Run method we have to make one while loop and repeat all the moving until we crash to the border of the console. Draw method needs the snake symbol to draw it on the console. - "\u25CF". We use **Thread.Sleep** to slow the snake so we can see it.



Now the snake moves but only in one direction. To change the direction we have to make one check if our player have pressed any key. Depending on the key, change the direction of the snake.

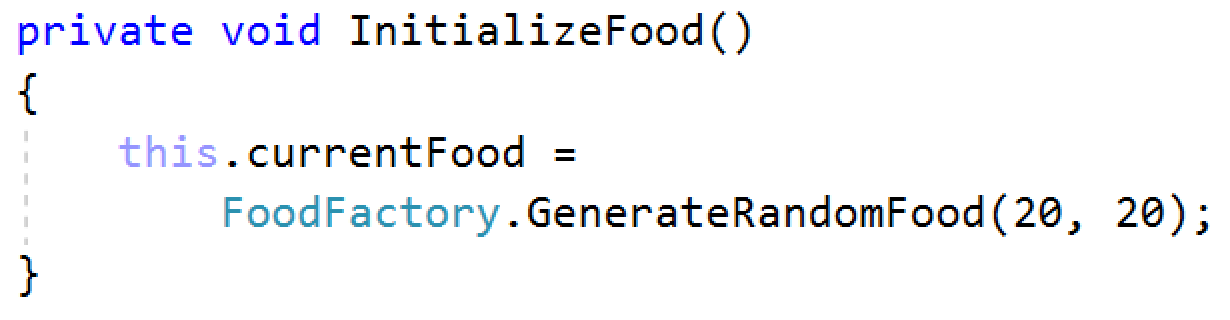


**SetNewDirection** is responsible for changing the direction correctly. What's incorrect move? For example your current direction is **Left**. The snake must not have the chance to go **Right**. Our moving will be done by the arrow keys.

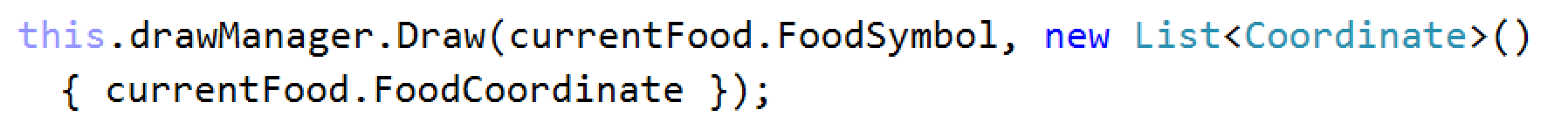


Do the rest for the other arrow keys and in the end change the snake current direction.

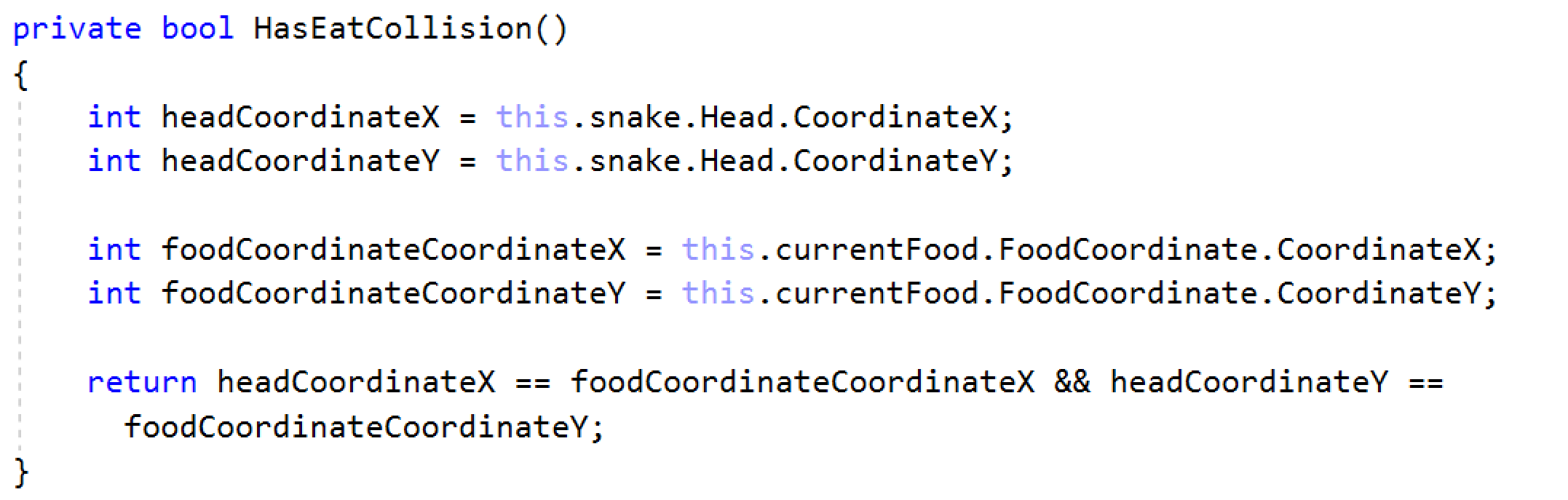
All we have done for now is just to make our snake move in the direction we want. Now let's add the food on the console and make our snake eat it. Make a field for the food and now we will use our **FoodFactory** that we made earlier. Make one method **InitializeFood** and set our field to our new generated food. Invoke it in the constructor of the Engine.



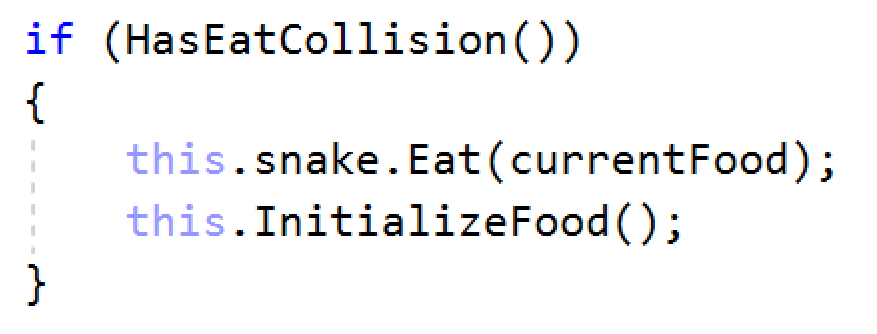
After we have created our food, now it is time to draw it on the console. In Run method invoke the Draw method with the food symbol and coordinates.



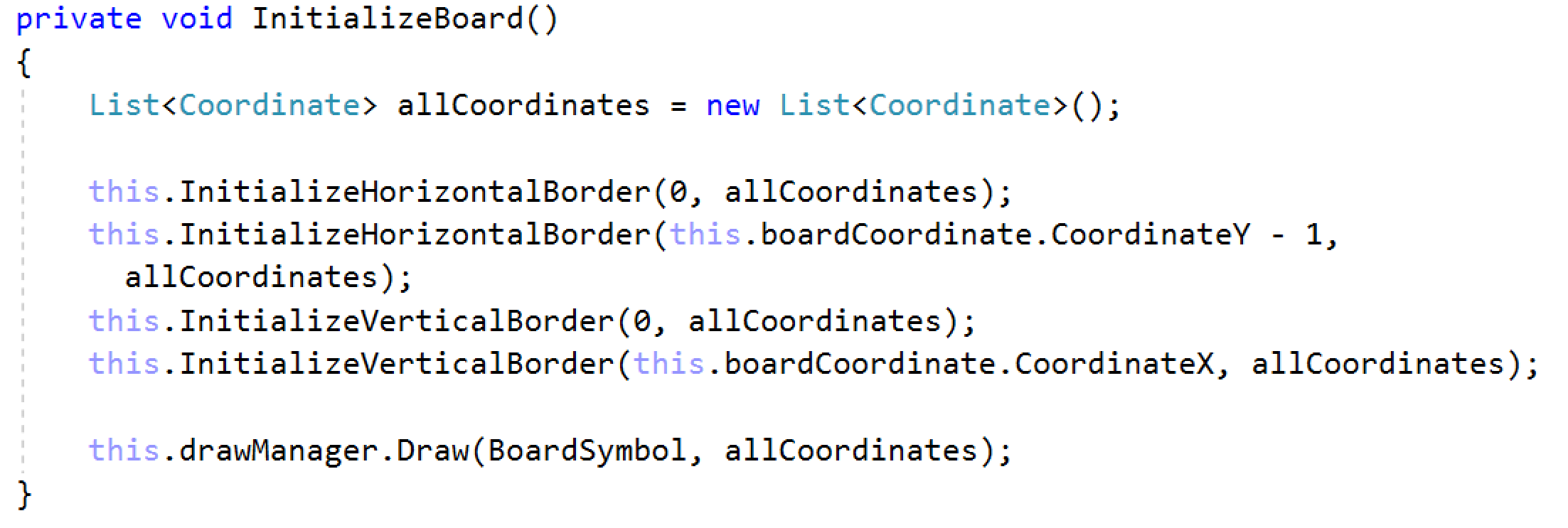
When the snake reaches the food nothing happens. Our snake doesn't eat the food, let's implement that. When our snake head has the same coordinates as the food coordinates that is the moment when we have to eat the food an make a new one. Create a method called **HasEatCollision** which will check whether the head and the food have the same coordinates.



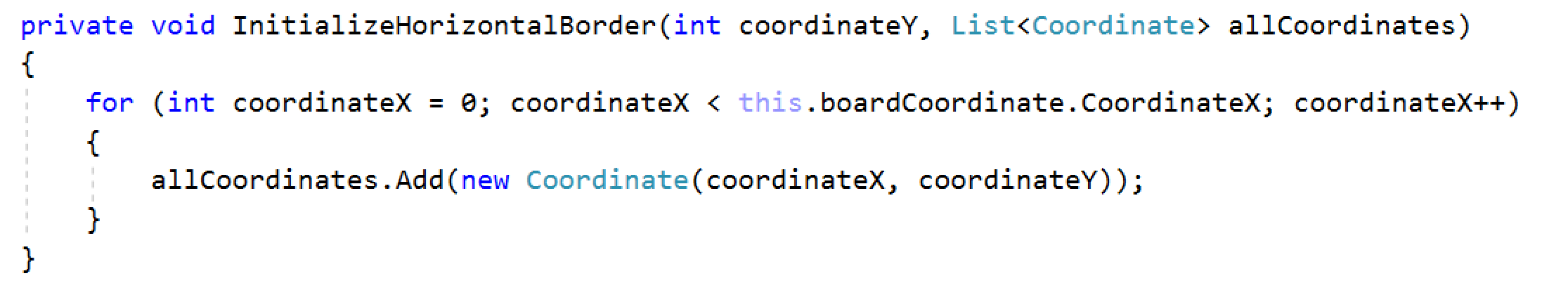
Now go back to the **Run** method and check whether this is true. If it is true call the **Eat** method of the **snake** and **initialize** a new **food**.



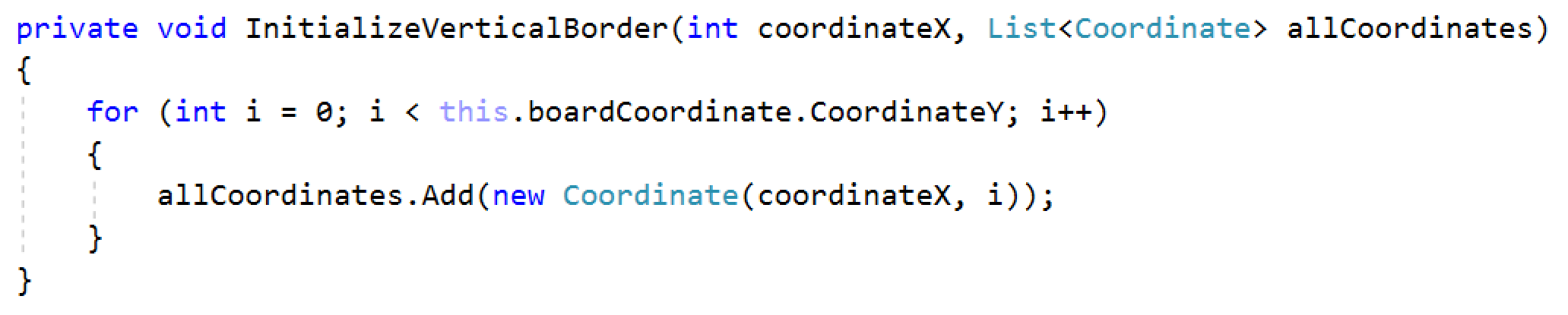
We have created the main functionality of our app. Now we have to draw a border for our snake. We will need one field to save the boarder coordinates. Than we have to create a method **InitializeBoard** which will generate the coordinates of the board. After that we pass them to the draw manager to visualize them. The board symbol is: "\u25A0"



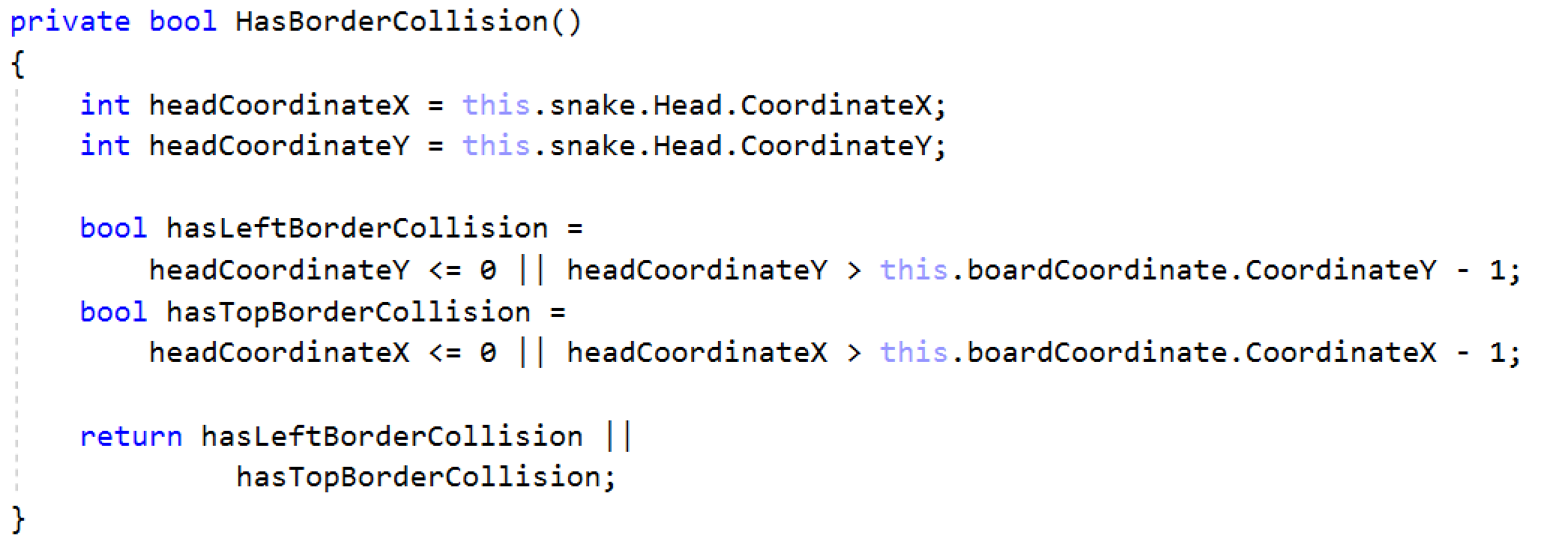
**InitializeHorizontalBorder** – this method generates the coordinates of the horizontal line



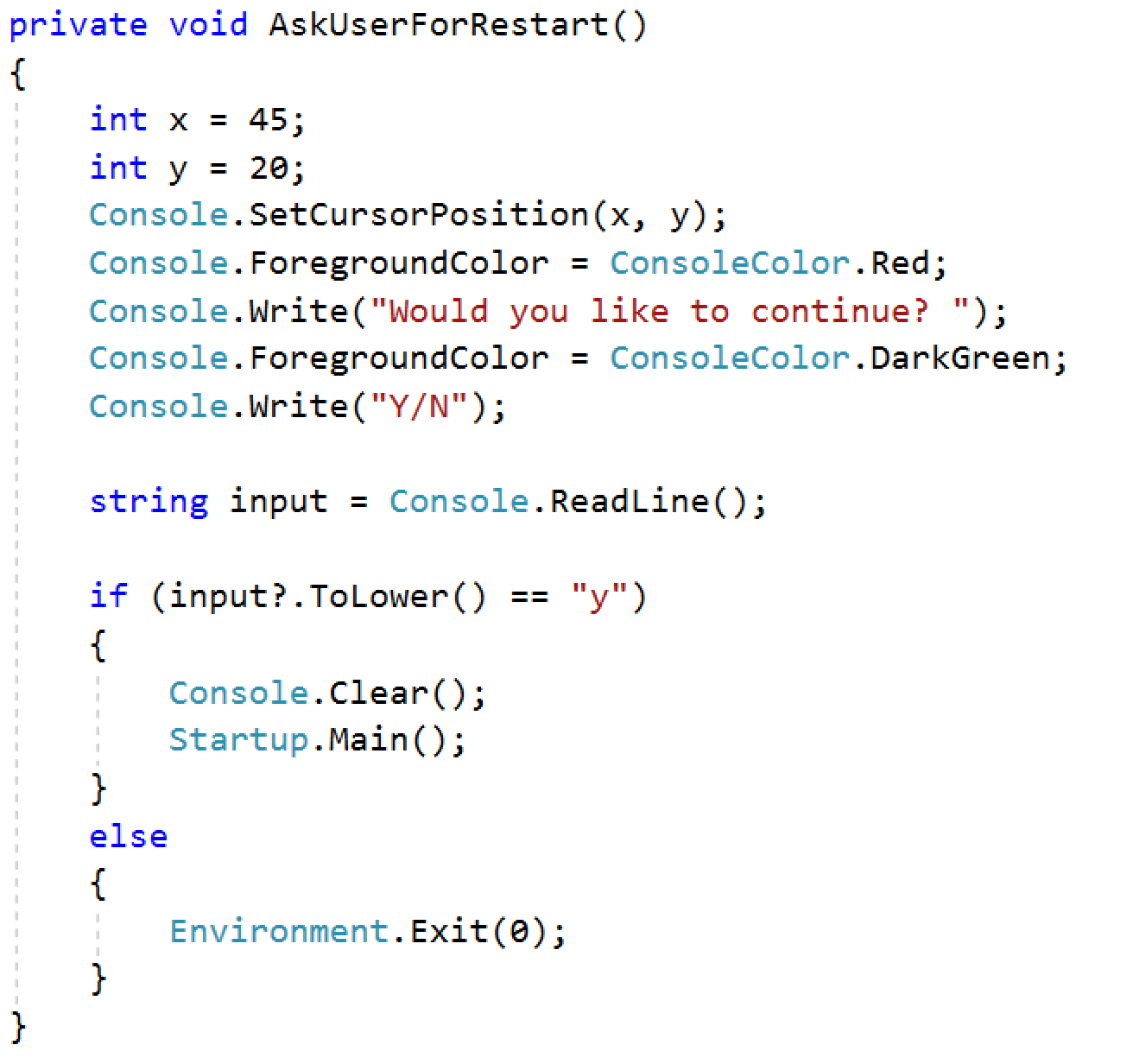
**InitializeVerticalBorder** – this method generates the coordinates of the vertical line



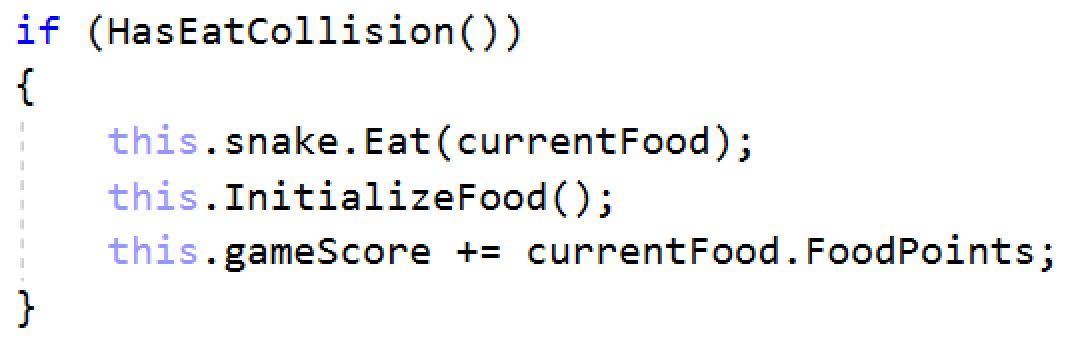
We have built our boarders but our snake can still pass through them. Following the same logic like we made for the food. We have to check whether the snake head is on the border. For this work we will create **HasBorderCollision** method.



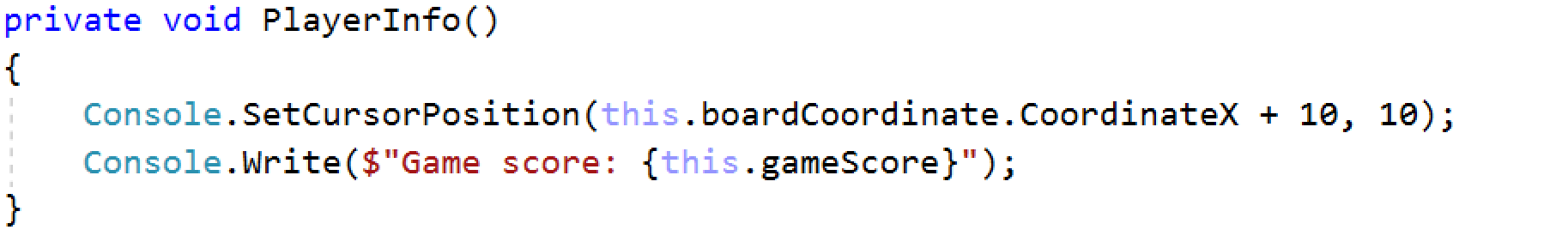
Now when we go to the border, we have to end the game and ask the player to play another game. Create method **AskUserForRestart.** This method will ask the user if he want to play another game.



Still we don’t have a player score and nothing is visualized on the console. Finally we will implement this. Create one field for the score and create **PlayerInfo** method. The place where we increase our score is when our snake eats. Let's go to that part of the code and add one more row.



Now implement the **PlayerInfo** method and invoke in the **Run** method.

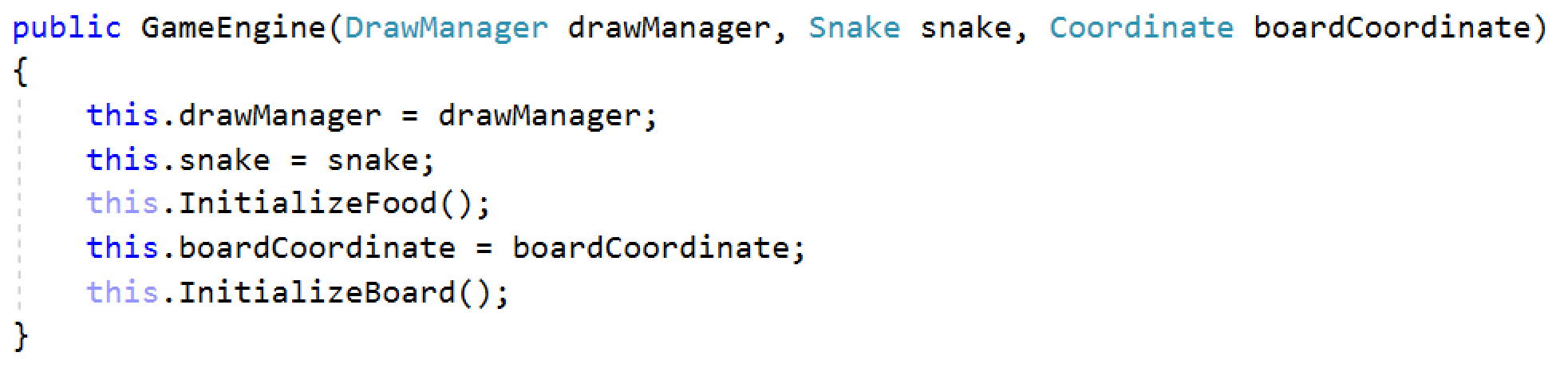


Finally your Run method should look like this:



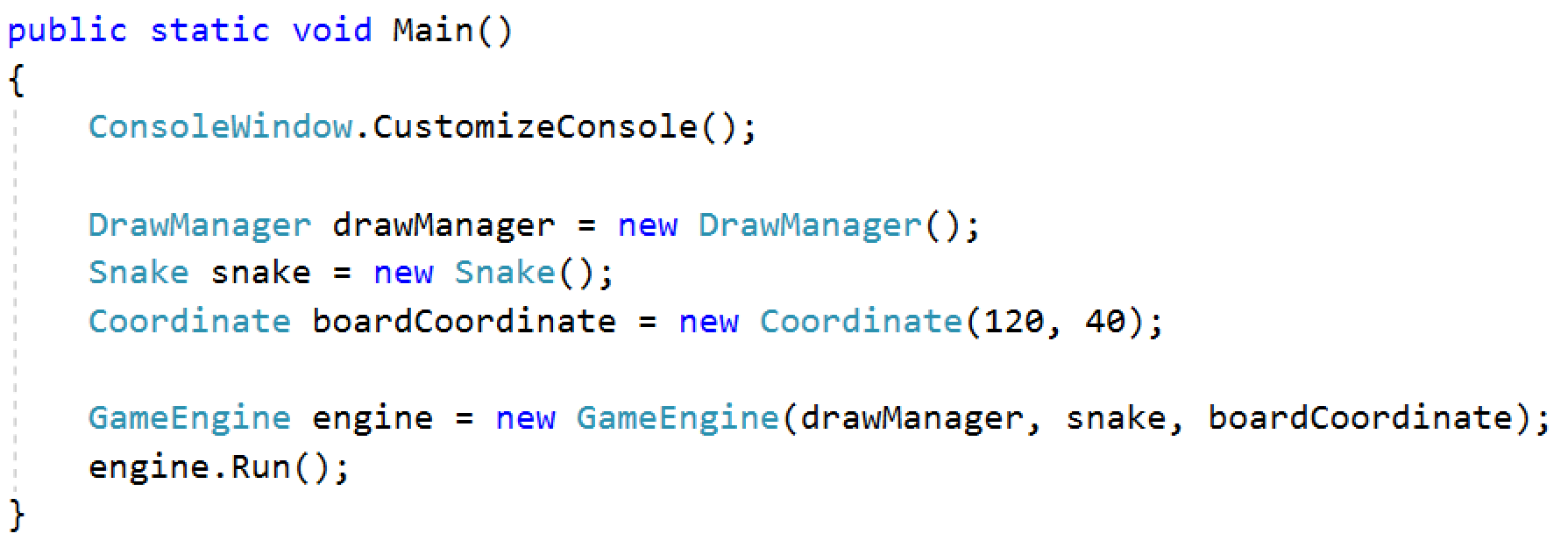
#### Constructor

Your engine constructor will take **DrawManager**, **Snake** and **BoardCoordinate** as parameters. It will also initialize our Food and Board.



### StartUp

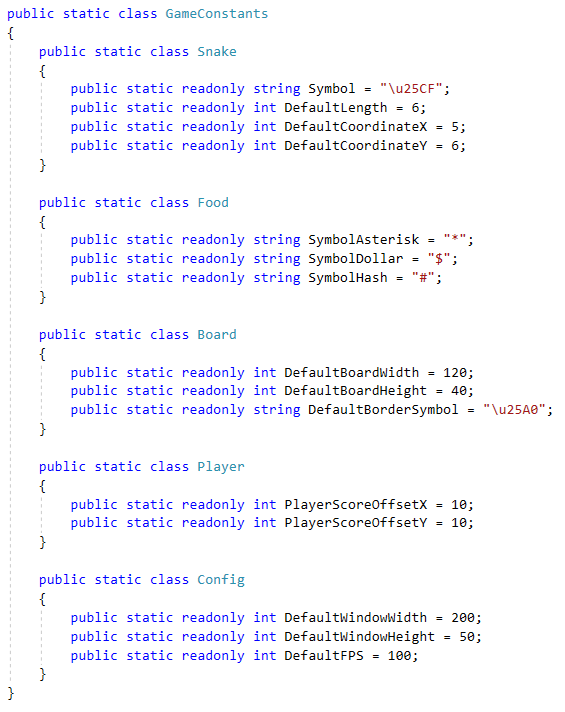
Our StartUp class should only initialize our **DrawManager**, **Snake**, **BoardCoordinate** and call the Engine Run method!



As you can see on the image of the game on the right side you have game statistic. You can think of a way to show on the console these game stats.

### Bonus

Export all the constants in one static class **GameConstants.**



Start the game and have a nice play 😊!