# Практически проект: Падащи камъни

Днес ще разработим конзолната игра "**Падащи камъни**":



"**Падащи камъни**" е игра, чието цел е да **избягвате падащи камъни** и да **оцелеете възможно най-дълго**, за да изкарате **най-висок резултат**.

Graphical user interface, text

Description automatically generated

Background pattern

Description automatically generated Background pattern

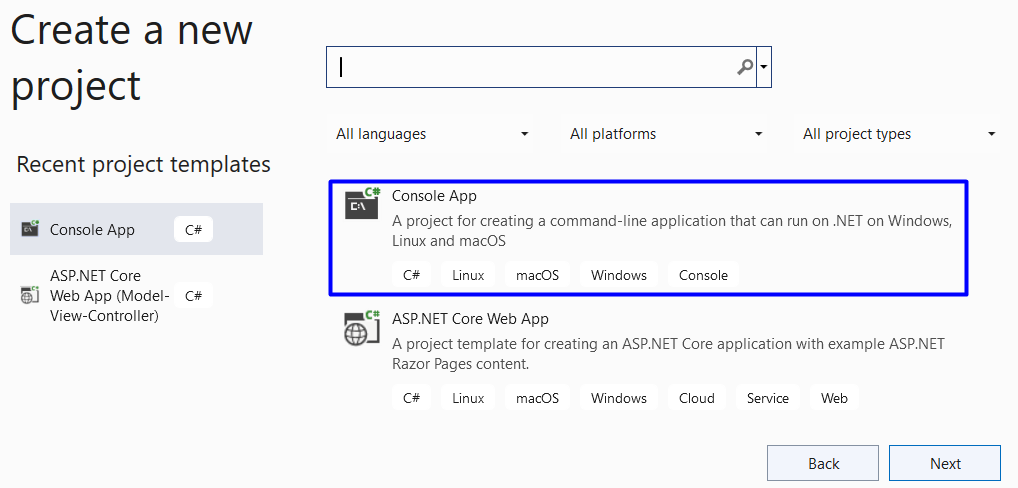
Description automatically generated

## Писане на кода на играта

Нека да започнем да създаваме нашата игра.

### Създаване на Visual Studio проект

Първо трябва да **стартираме Visual Studio** и да създадем **ново C# конзолно приложение**:



Създайте **.NET 6 Console App** и преди да продължим, променете името на **главния клас** от Program.cs на нещо **по-смислено**, например **FallingRocks.cs**.

## Създаване на GitHub репо

Изберете **един от начините**, които вече сте учили, и **създайте GitHub репо** за проекта – от сайта на **GitHub** или директно през **Visual Studio**.

|  |  |
| --- | --- |
| Icon  Description automatically generated | Важно е да изберете **ваше оригинално име** за проекта!  Вашият GitHub профил трябва да бъде **уникален** и да не бъде същият като този на вашите съученици.  Може да следвате тези инструкции за разработка на проекта, но можете и да **правите промени** и да **имплементирате проекта по различен начин** от вашите съученици. |

Graphical user interface, text, application, email

Description automatically generated

### Стъпки за имплементация на играта

Ще имплементираме **два класа**.

Първият, **class Player**, ще представлява **играча**, който се **движи** чрез **лявата** и **дясната стрелка** на клавиатурата, за да **избягва камъните**. Класът **Player** ще има:

* Конструктор, който приема:
  + X координата на играча
  + Y координата на играча
* **Четири** свойства:
  + int X – **хоризонталната координата** на играча
  + int Y – **вертикалната координата** на играча
  + string Representation – означението на играча чрез символи
  + bool HasBeenHit – връща дали играчът и някой камък са на една и съща позиция (тогава има сблъсък)
* Четириметода:
  + void Draw() – рисува играча на конзолата с текущите **X** и **Y** **координати**
  + void Move() – движи играча в зависимост от стрелката на клавиатурата, която е натисната
  + bool IsOutOfConsoleBoundaries() – проверява дали **текущата позиция** на **играча** е **извън границите** на конзолата
  + void ClearKeyAvailableBuffer() – изчиства буфера с метода Console.KeyAvailable()

Вторият клас, Rock, представлява **камъните**, които **падат** от горната част на конзолата. Класът **Rock** ще има:

* **Конструктор**, който приема:
  + **X** **координатата** на камъка
* Три **свойства**:
  + int X – **хоризонталната координата** на камъка
  + int Y – **вертикалната координата** на камъка
  + char Symbol – означението на всеки камък чрез символ

След като сме **готови с класовете**, ще имплементираме **главната логика** на играта. За нея ще ни трябват **четиринадесет метода**, които ще си взаимодействат, за да **работи играта правилно**.

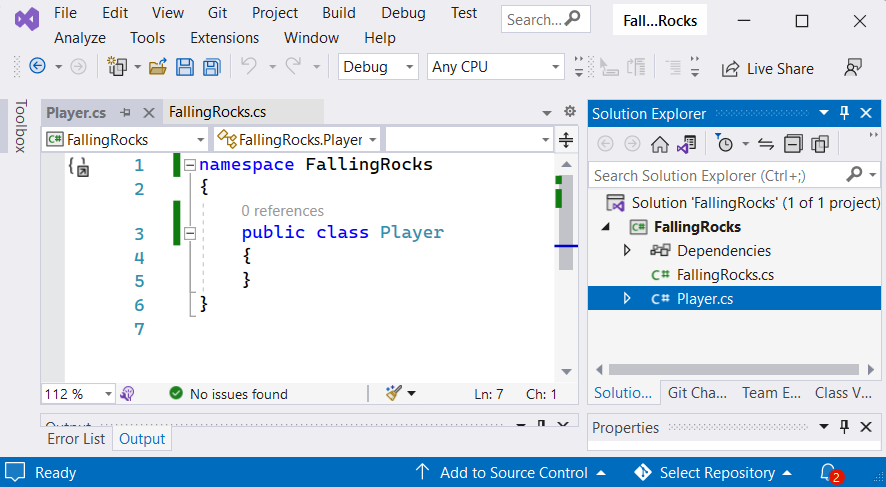
* int ProcessPlayerRocksSpawnRateChoice(string inputSpawnRate) – обработва **входа** на играча за **скоростта на генериране на камъни**
* int ProcessPlayerRockFallSpeedChoice(string inputFallSpeed) – обработва **входа** на потребителя за **скоростта на падане** на камъните
* void SetWindowProperties() – задава **ширината** и **височината** на **конзолата**
* void RedrawConsole() – рисува **наново конзолата**, за да избегнем **премигване на конзолата**
* void DrawScorePanel() – рисува **панел с резултата**, където ще се визуализират **точките**
* void CreateRocks() – **създава камъни**, за да можем да ги **нарисуваме на конзолата**
* bool ShouldGenerateRock() – дава знак на програмата кога да **генерира нови камъни**
* void DrawRocks() – **рисува** генерираните **камъни**
* void MoveRocks() – **премества** **камъните**
* bool ThereIsCollision(Rock rock, Player player) – проверява дали има **сблъсък** между **играча** и някой **камък**
* bool RockAndPlayerAreOnSameWidth(Rock rock, Player player) – проверява дали **играчът** и **камъкът** са на **една и съща ширина**
* bool RockAndPlayerAreOnSameHeight(Rock rock, Player player) – проверява дали **играчът** и **камъкът** са на **една и съща височина**
* void RemoveRocks(List<Rock> rocksToRemove, List<Rock> rocks) – **премахва камъните**, които са **задминали играча**
* void EndGame() – изписва "**Game Over!**", след като **играчът** **загуби**

Това е **всичко**, от което се нуждаем. Нека да продължим с **имплементацията**.

## Дефиниране на класовете

### Class Player

Нека да **създадем клас** и да го наименуваме **Player**. Вече знаете как да го направите:



Първото нещо, което ще направим в класа, е да **добавим свойствата**, които са ни нужни. **Първите две** свойства ще бъдат **X** и **Y** (**координатите на играча**). Те ще бъдат от **тип** int и ще ни помогнат да знаем **къде** **на конзолата** трябва да бъде **играчът**, за да го **нарисуваме**.

Text

Description automatically generated with medium confidence

Следващото свойство е Representation. То ще бъде от **тип string** и ще съдържа **означението** на играча чрез **символи**.



**Последното** свойство, което ще ни трябва в този клас, еHasBeenHit. То ще бъде от **тип bool** и ще ни показва дали **играчът** е бил **ударен от камък**. Първоначално това свойство ще има **стойност false**, в противен случай играта **няма да започне**, защото **главната логика** на играта ще установи, че **играчът е ударен**, и ще **прекрати цикъла**.



Готови сме със **свойствата**. Сега трябва да добавим **конструктора**, който **приема x** и **y координатите**:



Следващата стъпка е да зададем подадените през **конструктора x** и **y координати** като стойностина **X** и **Y свойствата**:

Text

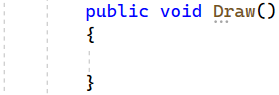
Description automatically generated with low confidence

Накрая задаваме свойството **Representation**, като избираме **символ**, с който да **означим играча** на конзолата. В този пример ще използваме "(0)".

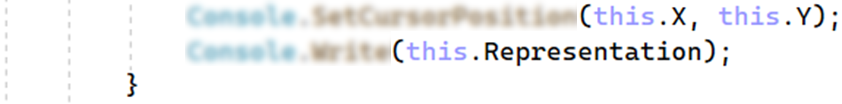


Това е всичко, от което се нуждаем за нашия **конструктор**. Следващата стъпка е да **създадем методите**.

Създайте метода Draw(). Този метод ще бъде от тип **void** и ще **рисува на конзолата** символите на **играча** в зависимост от **X** и **Y координатите**.



В този метод трябва да зададем **позицията на курсора** на текущите **X** и **Y** **координати** и да **напишем на конзолата** избраните **символи** (**репрезентация**):



Това е **всичко за този метод**. Нека да продължим със **следващия**.

Създайте метода Move(). Той ще бъде от тип **void** и ще **движи** играча **наляво** или **надясно** в зависимост от това **коя стрелка** на клавиатурата е **натисната**.

A picture containing logo

Description automatically generated

В този метод първо **проверяваме** дали **има натиснат клавиш**. Можем да го направим с Console.KeyAvailable:

A picture containing text

Description automatically generated

Ако има **натиснат клавиш**, трябва да го **запазим** в **променлива**, използвайки метода Console.ReadKey(). Тази променлива ще бъде от **тип** ConsoleKeyInfo. Трябва да изглежда така:



Следващото нещо, което ще използваме, е метода ClearKeyAvailableBuffer()– той ще **пречи** на **ConsoleKey** да запазва **натиснатите клавиши**. Тъй като този метод **още не е имплементиран**, само го **извикайте** и след това го **закоментирайте**.

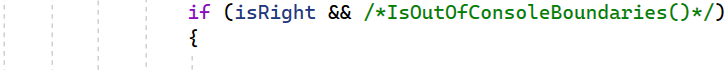


Сега трябва да създадем **две булеви променливи**. Първата ще проверява дали **натиснатият клавиш** е **дясната стрелка** (→), а втората ще проверява дали **натиснатият клавиш** е **дясната стрелка** (→). Тези проверки за **важни**, защото ако ги нямаме, играта ще се **счупи**, ако потребителят натисне **някой друг клавиш**.

Може да го направим, като достъпим **свойството** Keyна **променливата** pressedKey и проверим дали е **равно** на ConsoleKey.RightArrow и респективно LeftArrow. Кодът трябва да изглежда така:



Сега трябва да проверим дали **натиснатият клавиш** е **дясна стрелка** и дали не е **извън границите**. Можем да проверим дали е **извън границите** с метода IsOutOfConsoleBoundaries(). Тъй като методът **още не е имплементиран**, го **извикайте** и го **закоментирайте**.



Ако проверката върне стойност **true**, **добавяме** **1** към свойството **X**.

A screenshot of a computer

Description automatically generated with medium confidence

Направете същото за **лявата стрелка**, само че **извадете 1** от свойството **X**. Също помислете как да проверите дали играчът е **извън лявата граница**.

A picture containing graphical user interface

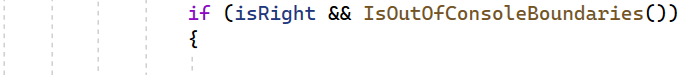
Description automatically generated

Това е всичко за този метод. Сега нека да имплементираме метода IsOutOfConsoleBoundaries(), за да можем да **откоментираме** извикването му от метода **Move()**. Методът ще ни покаже, ако достигнем **максималната ширина** на конзолата. Трябва да направим тази проверка, иначе играчът ще излезе извън конзолата и играта няма да работи. Методът ще върне стойност **true**, ако текущата **X координата** **плюс** дължината на **Representation** е **по-малка** от **ширината на конзолата минус 1**. Ще използваме ламбда израз. Методът трябва да бъде **частен**, защото не искаме да се достъпва от друг клас освен от **Player**.

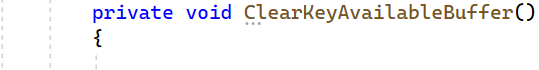
Трябва да изглежда така:



След като **приключим с метода**, можем да се върнем в метода Move() и да **откоментираме** тази част**.**



Сега само трябва да имплементираме метода ClearKeyAvailableBuffer(). Този метод е необходим, защото ще попречи на **ConsoleKey** да запази **натиснатите клавиши** (ако някой клавиш е задържан натиснат, ConsoleKey ще запомни, че този клавиш е **натиснат** и **няма да можете да преместите играча**). Този метод трябва да е **частен**, защото не искаме да се достъпва от друг клас освен от **Player**.



В метода трябва да създадем while-цикъл, който ще итерира докато има **наличен клавиш**.

A picture containing text

Description automatically generated

В цикъла ще **четем клавишите**.

A picture containing text

Description automatically generated

В момента, в който се **натисне клавиш**, цикълът ще **спре** и играчът ще се **премести** в **избраната посока**. Това е **всичко за този метод**. Върнете се в метода Move() и **откоментирайте** новосъздадения метод.

Готови сме с класа **Player**. Следващата стъпка е да имплементираме класа **Rock**.

### Class Rock

Create a **new** **class** and **name** it **Rock**. You should already know how to do this:

A picture containing text

Description automatically generated

In the class first, we will need **three** **properties**. The **first** **two** will be of **type** int and will **represent** the X and Y **coordinates** of the **rock**.

Text

Description automatically generated with low confidence

The **third** **property** will be of type **char** and will be the **representation** of the **rock**.



Now, before we **create** the **constructor**, we will need **two** private fields. The **first** will be of **type** char[] and will **contain** all the **different** **representations** of the **rock** (what the rock will look like when it is created).



**\*Note** - You can **copy** these or **create** your **own** so your **project** is **different** from the other **students**.

|  |
| --- |
| '^', '@', '\*', '&', '+', '%', '$', '#', '!', '.', ';' |

The next **private** **field** will be of **type** Random and will **help** us to **get** a **new** **symbol** **every** **time** a **new** **rock** is **created** (so every other falling rock is different from the previous one).



We have the **properties** and the **fields**. What is **left** is to **create** the **constructor** and our **class** is **done**.

Create the **constructor** and **let** it **accept** the X **coordinate**. Here we **need** **only** the X **coordinate** because the **rocks** will **always** **start** to fall from Y **zero**(0).

A picture containing text

Description automatically generated

Now we have to **set** the **properties** in the **constructor** like in the **previous** **class**. This time Y **coordinate** will be **equal** to **zero**:

Icon

Description automatically generated

The Symbol **property** will be **equal** to our **representation** **array** on a **random** **index**. You should know how to do that:



This is all we need for our Rock class. Let's **continue** with the **initialization** of the **game**.

## Initialize the Game

### Prepare Required Settings

First, **click** on the file "FallingRocks.cs" in the [SolutionExplorer]**.** In this file, we will **initialize** the **game.** We will first **implement** the **main** **logic** of the **game** and **comment** **it** **on**. Then by **implementing** the **methods** we will **slowly** **uncomment** the **code** and **test** the **game**.

Now we'll **need three** (3)int constants that will **help** us **set** the **width** and **height** of the **console**, as well as the **size** of the **score** **panel** (which we'll draw on the console). We'll need these **constants** **later** in the **implementation**.

Text, chat or text message

Description automatically generated

**\*Note** – You can **use** the **same** **numbers** as us or **set** **up** **your** **own**.

The next thing is the int **variable**, which will **store** the **result** the **user** has **achieved** (this variable will increase by one whenever a rock pass next to the player).



Now we have to **read** the **user** **input** for the **rock** **spawn** **rate** and **rock** **fall** **speed** (this is the difficulty of the game). **Instead** of **writing** **everything** **there**, we will **create** **two** **methods** to **do** **this** **for** **us** (so the code is easier to read).

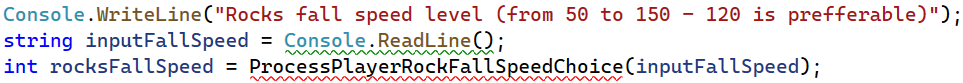
Before we use the first **method**, we should **write** a **message** on the **console** to **explain** to the **user** what he has to **choose**. Also, **read** the **user** **input**.



Now create a int **variable**, and **assign** the ProcessPlayerRocksSpawnRateChoice(string inputSpawnRate) method to it.



Do the same thing for the rock fall speed. Use the method ProcessPlayerRocksFallSpeedChoice(string inputFallSpeed)



**\*Note** - **Comment** on **methods** so we **don't get errors**.

The next **step** is to **set** the **console** **windows** **properties**. Here we will use the SetWindowProperties() method.



Now we need **three** **more** int **variables**, which will **help** **us** to **set** the X and Y **coordinates** of our **player**. To get the X **coordinate**, we need to get the **width** of the **console** **windows** and **divide** it by **two**.



To get the Y **coordinate**, first, we **need** to **get** the **height** of the **console** **window** and **subtract** from it the **size** of the **score** **panel** (we have a constant for this).



Now to get the **exact** Y **coordinate** we have to **subtract** **one** from the **result** (otherwise the player will be positioned at the row of the score panel).



We have the X and Y **coordinates**, all we **need** to do is **create** an **instance** for our **player**. This time when we **create** the **instance**, the **player** **body** will **require** **two** **parameters**. These **two** **parameters** **come** from the **constructor** of the Player class that we **initialize** **earlier**.

A picture containing application

Description automatically generated

What we have to do is to **pass** the **previous** **two** **variables** that we **create** for the X and Y **coordinates** of the **player**.



This is all we need to **create** an **instance** of our **player**. The last thing **before** the **game** **loop** is to **create** a List<Rock> in **which** we will **add** the **newly** **created** **rocks** before **letting** them **drop** **down** and try to kill the **player**. You should know how to do this.



This is all we need to prepare before the game loop.

### Initialize Game Loop

Now we have **everything** **ready** for the **game** **loop**. Here the **game** will **start** and **end** when the **user** **losses**.

First, open a while loop.

Graphical user interface, application

Description automatically generated

The first **method** that we are going to **need** in the **game** **loop** is RedrawConsole(). The **method** will **simply** **redraw** the **console**. For now, let’s **comment** it until we **write** the **method**.



The next thing is to **create** the **rocks**. We will use the **method** CreateRocks() to do the work, but for now we will **comment** it.



Now we have to **draw** the **player**. For this purpose, we can use the Draw() **method** we **created** **earlier** in the Player **class**. We will **access** the **method** from the **instance** of the Player **class** we **created** **earlier**.



Next is to **draw** the **rocks**. To do this we will use the **method** DrawRocks(). Let’s **comment** it for now



Now the **player** and the **rocks** are **set** **up**, next thing is to **start** **moving** **them**. We'll move the **player** by **calling** its Move() **method**, and the **rocks** **will** be **moved** using the MoveRocks() **method**.

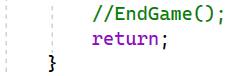


The last thing that we have to do is to **check** if the **player** **has** **been** **hit** by a **rock**. We will do this by **calling** the HasBeenHit **property** of our **player**.

A picture containing logo

Description automatically generated

If the **player** **has** **been** **hit** we will use the EndGame() **method** which will draw "**Game Over**" on the **console** and then **break** the **loop**.



Finally, we will use the Thread.Sleep() **method**, which will **stop** the **loop** for a **certain** **number** of **milliseconds**. The **milliseconds** will be **defined** by the **users** and will be **stored** in the rockFallSpeed **variable**.



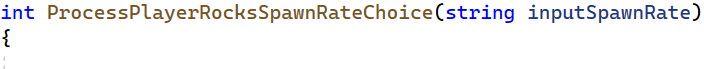
That's all we need for the **game** **loop**. **Comment** on the **code** which is **giving** you **errors**, we will **uncomment** it **slowly** with the **implementation** of the **methods**. Now let's **create** the **methods**.

### Implement the Methods

#### Implement the ProcessPlayerRocksSpawnRateChoice(string inputSpawnRate) Method

This **method** will **process** the **user input** for the **rock's spawn rate**. The **hardness** will be from **one** (1) to **ten** (10), we will have a **preferred** **option** which will be **four** (4). When the **user** **input** is **processed** the **method** will **return** the **value**.

Create the **method** and let it accept the **input spawn rate** decidedbythe **user**.



The **next** **step** is to **validate** the user **input**. Think of a way to **validate** the **user** **input** by yourself.

**Validation hints:**

* The **input** **should** be a **number** (integer).
* The **number** **should** be on a **scale** from **one** (1) to **ten** (10).
  + Not **less** **than** **one**(1).
  + Not **bigger** **than** **ten** (10).
* If the **number** is **bigger** than **ten** (10) or **less** than **one** (1), **set** the **number** to the **preferable** **value** of **four** (4).

Finally, **return** the **result**.

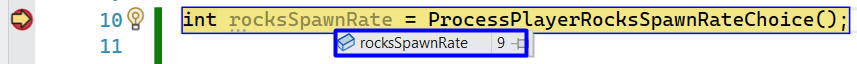
Graphical user interface, application, Teams

Description automatically generated

When you are **done** with the **method**, go back to the **code** **uncomment** it and **test** (debug) the **method** to see if it **works** **correctly**.

Graphical user interface, text

Description automatically generated

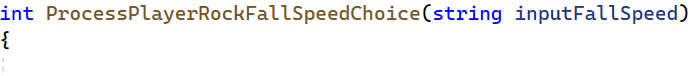


Let's continue with the next one.

#### Implement the ProcessPlayerRockFallSpeedChoice() Method

This **method** will **process** the **user input** for the **rock's fall speed**. The **hardness** will be from **fifty** (50) to **one hundred and fifty** (150), we will have a **preferred** **option** which will be **one hundred and twenty** (120). When the **user** **input** is **processed** the **method** will **return** the **value**.

Create the **method** and let it accept the **input fall speed** decidedbythe **user**.



Now think of a way to **validate** the **user** **input** by yourself.

**Validation hints:**

* The **input** **should** be a **number** (integer).
* The **number** **should** be on a **scale** from **fifty** (50) to **one hundred and fifty** (150).
  + Not **less** **than** **fifty** (50).
  + Not **bigger** **than** **one hundred and fifty** (150).
* If the **number** is **bigger** than **one hundred and fifty** (150) or **less** than **fifty** (50), **set** the **number** to the **preferable** **value** of **one hundred and twenty** (120).

Graphical user interface, application, Teams

Description automatically generated

Before the **fall** **speed** is **returned**, we have to **make** one **more** **calculation**. **Because** we are **working** with **milliseconds** we hit a **problem** **here**.

**Problem** – We want **fifty** (50) to be the **slowest** **fall** **speed** and **one hundred and fifty** (150) to be the **fastest**. If the user **enters** **fifty** (50) the **game** will **run** **faster** (because we are **working** with **milliseconds**). To **resolve** this **problem** we just have to **reverse** the **logic**.

**Hint** – The **total** **rock** **fall** **speed** is **two hundred** (200).

Try to **resolve** the **problem** by yourself and **return** the **value**.

Graphical user interface

Description automatically generated

When you are **done** with the **method**, go back to the **code** **uncomment** it and **test** (debug) the **method** to see if it **works** **correctly**.

Text

Description automatically generated



Let's continue with the next one.

#### Implement the SetWindowProperties() Method

This **method** will **set** the **properties** we will **need** for our **console**. The **width** and **height** of the **console**, also the **visibility** of the **cursor** will be **set** to **false**. The **method** will **make** **adjustments** **before** the **game** **starts**.

Create the **method**, **clear** the **console** and **set** the **cursor** **visibility** to **false**.

Text

Description automatically generated

Now we will use two **build-in methods** which will **help** us to **set** the **console** **windows'** **width** and **height**. The first **method** is Console.SetWindowSize(). The **method** **sets** the **height** and **width** of the **console** **window** to the **specified** **values**. As **values** for the **method**, we will **pass** the **constants** WindowsHeight and WindowsWidth, which we **created** **earlier**.



**More** **about** it here: <https://learn.microsoft.com/en-us/dotnet/api/system.console.setwindowsize?view=net-6.0>.

The second **method** is Console.SetBufferSize(). The **method** **sets** the **height** and **width** of the **screen** **buffer** **area** to the **specified** **values**. A **screen** **buffer** is **a two-dimensional array of character and colour data for output in a console window**. The **active** **screen** **buffer** is the **one** that is **displayed** on the **screen**. We will **pass** the **same** **values** as in the **previous** **method**.



**More** **about** the method: <https://learn.microsoft.com/en-us/dotnet/api/system.console.setbuffersize?view=net-6.0>.

**More** **about** the screen buffer: <https://learn.microsoft.com/en-us/windows/console/console-screen-buffers>.

This is all we need for this **method**. Go back to the **code** **uncomment** it and **test** (debug) the **method** to see if it **works** **correctly**.

If everything works correctly, the **console** **windows** should **shrink** to the **given** **values** **immediately** after you **enter** the **user** **input** **values**.

Text

Description automatically generated 🡪Text

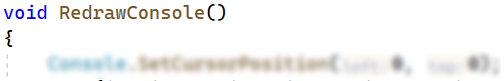
Description automatically generated

Let's continue with the next method.

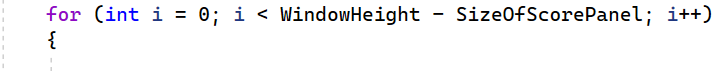
#### Implement the RedrawConsole() Method

The **method** will **redraw** the **console** **without** **clearing** it. That means we will **start** from the **top-left** **corner** of the **console** **window**, and **draw** an **empty** **string** till the **end** of the **console**. This will **help** us to **escape** the **console** **blinking** which is **caused** by the Console.Clear() **method**.

**Create** the **method** and **set** the **cursor** **position** to **(0, 0)** which is the **top** **left** **corner** of the **console**.



The **next** **step** is to **create** a for loop which will **iterate** **until** i is **less** than the **value** **retrieved** from the **console** **window** **height** **minus** the **size** **of** **the** **score** **panel** (we have a constant set for these two values). We will **subtract** the **size** **of** **the** **score** **panel** from the **console** **window** **height** because it will **always** be in the **same** **place**, and will **stay** **there** **until** the **end** of the **game**.



In the **loop**, we just have to **write** a **white** **space** **fifty** **times** (50) **which** is the **console** **window** **width** **size** (there is a constant for this). Do this by yourself.



The last thing is to **use** the DrawScorePanel() **method** – the **method** will **draw** the **score panel** on the **console**. The **method** is **not** **implemented** yet, just type its **name** and **comment** it out.

A picture containing logo

Description automatically generated

This is all we need for this **method**, we can **test** it **after** **implementing** the DrawScorePanel() **method** because **both** **methods** work **together**, and in the **game** **loop** we **call** **only** the RedrawConsole() **method** (that means we can test both of them together). Let's continue with it.

#### Implement the DrawScorePanel() Method

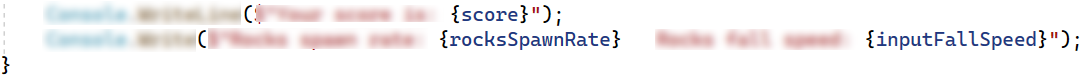
The **method** will **separate** the **battlefield** (the top part where the **rocks** are **falling**) with the (=) **symbol** and **display** on the **console** the **score** of the **user**, the rocks' **fall** **speed** and the **spawn** **rate** **selected** by the **user**.

**Create** the **method** and **figure** out a **way** to **draw** the **symbol** (=) **across** the **width** of the **console** **window**. There are **several** **ways** to do it, pick yours and do it.

Graphical user interface, text, application, chat or text message

Description automatically generated

What is **left** is to **display** the **score** of the **user**, the **fall** **speed** and the **spawn** **rate** of the **rocks**. To **display** the **fall** **speed** we will **use** the **user** **input** **value** **otherwise**, it will **display** the **value** that we have **changed** in the ProcessPlayerRocksFallSpeedChoice(string inputFallSpeed) method. Do it yourself, you have the **necessary** **variables**, just come up with a **message** to **display** **them**.



This is all we need for this **method**. Go back to the RedrawConsole() **method** and **uncomment** it,then **go** **back** to the **game** **loop** and **uncomment** theRedrawConsole() **method** so we can **test** them.

Let's **run** the **application** and see what the **results** are.

A screenshot of a computer

Description automatically generated with medium confidence

If everything **works** **correctly**, after **running** the **application** the **console** **should** be **split** into **two** (2) **parts**. The **bottom** **part** should have the **score**, the spawn **rate** and the **fall** **speed** of the **rocks**. Let's continue with the next method.

#### Implement the CreateRocks() Method

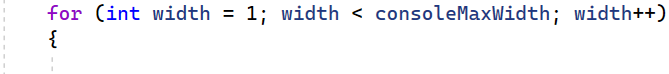
The **method creates** new **rocks depending** on the **rock spawn rate** that the **user** **chooses** and **adds** **them** to the **rocks** **List** that we have **created** **earlier**.

**Create** the **method** and a **variable** which will **hold** the **console** **max** **width** **minus** **one** (1). We want the **max** **width** **minus** **one** (1) **otherwise**, we will **escape** the **boundaries** of the **console** **window** and the **program** will **stop**.

A picture containing logo

Description automatically generated

After that **open** a for loop **which** will **iterate** until i (width) is **less** than the **console max width**. This will help us to **create** a **rock** and **set** its X **coordinate** to the **current width**, so when the **rock** is **drawn**, it will **start** to **fall** from that **coordinate**.



Before creating a **rock** we **check** if we **should** **create** **one**. Here we will **use** the ShouldGenerateRock() **method**. The **method** is **not** **implemented** yet, just type its **name** and **comment** it out.

A picture containing logo

Description automatically generated

If we should, just **create** a new Rock with the **current** **width** as a **parameter** and **add** it to the **rock** **List**. Do this yourself.

Graphical user interface, text, application

Description automatically generated

This is all we need for this **method**, we can **test** it **after** **implementing** the ShouldGenerateRock() **method** because **both** **methods** work **together**, and in the **game** **loop** we **call** **only** the CreateRocks() **method** (that means we can test both of them together). Let's continue with it.

#### Implement the ShouldGenerateRock() Method

The **method** will **return** true or false, **depending** on a **random** **number** between (**0** and **101**) and the **rock** **spawn** **rate** **chosen** by the **player**.

**Create** the **method** and a **new** **random** **variable**.

Graphical user interface, application

Description automatically generated

Now we need a **variable** for the **spawn** **rate** **max** **value**. The **spawn** rate **max** **value** is **one hundred** (100).



After that, we just have to **check** if the **random** **number** **between** (**1** and **101**) is **bigger** or **equal** to the **variable** spownRateMaxValue **minus** the **rocks' spawn rate**. We do this **because** the **smaller** the **subtraction** **value**, the **more** **rocks** are **generated**.



This is all we need for this **method**. Go back to the CreateRocks() **method** and **uncomment** it,then **go** **back** to the **game** **loop** and **uncomment** theCreateRocks() **method** so we can **test** (debug) them.

Let's run the application to see if it works properly.

A screenshot of a computer

Description automatically generated with medium confidence

We can see that the **console** is the **same** as **before**, but if we **set** a **breakpoint** and debug the **application**, we can see that at each **iteration** of the while loop **more** and **more** **rocks** are **added** to the **list** of **rocks**.Table

Description automatically generated with low confidence 🡪 Table

Description automatically generated

Let's continue with the next method.

#### Implement the DrawRocks() Method

This **method** will **display** the **rocks** to the **console**. We will use the **rocks** **list** in **which** we **add** the **rocks** in the **previous** **method**. We will **set** the **cursor** **position** to the **current** rock X and Y **coordinates** and **display** the **symbol** of the **rock** on the **console**.

**Create** the **method** and **open** a foreach loop which will **iterate** **through** the **list** of **rocks**.

A picture containing graphical user interface

Description automatically generated

Finally, **set** the **cursor** **position** to the **current** rock X and Y **coordinates** (by accessing the rock properties) and **write** the **rock** **symbol** on the **console**.

Graphical user interface

Description automatically generated with low confidence

This is all that we need for this **method**. Go back to the **game** **loop** and **uncomment** it. Also, **uncomment** the **previous** and the next **row** where we **draw** and move the **player** because we **already** have **created** its **method**.

Text

Description automatically generated

Now let's run the application and test it.

A picture containing graphical user interface

Description automatically generated A picture containing graphical user interface

Description automatically generated

We can see that when we **run** the **application**, the **rocks** are **beginning** to **appear** on the **console**, the **player** is **there** **too** and we **can** **move** **him**.

The next **method** that we are **going** to **implement**, is **going** to **move** the **rocks**. Let's continue with it.

#### Implement the MoveRocks() Method

This is the **most** **complex** **method** of **all** in this **application**. The **method** will **move** **rocks**, **remove** **rocks** when their Y **coordinate** **reaches** the **maximum** **height** of the **console**, and **check** for **collision** **between** the **player** and a **rock**.

Create the **method** and a List of **rocks** which will **contain** the **rocks** that **have** to **be** **removed**.

A picture containing logo

Description automatically generated

After that **open** a foreach loop **which** will **iterate** **through** the **list** of **created** **rocks** (the list **where** we **store** the **newly** **created** **rocks**).

A picture containing logo

Description automatically generated

The first thing in the loop is to **add** **one** (1) to the Y **coordinate** of the **current** **rock**. This will **move** the **rock** one **row** **down** in the **console**.



The **next** **step** is to **check** if the **current** rock Y **coordinate** has **reached** the **console** **max** **height**. If the **rock** **reaches** the **console** **max** **height** that **means** that we **avoid** the **rock**, so we can **add** it to the **list** of **rocks** to **remove** and **add** **one** (1) to the **user's score**.

Graphical user interface, text, application, chat or text message

Description automatically generated

Now we have to **check** if **there** is a **collision** **between** the **rock** and the **player**. Here we will **use** the ThereIsCollision(rock, player) **method**. The **method** is **not** **implemented** yet, just type its **name** and **comment** it out. If we have a **collision** we **have** to **set** the **player's** HasBeenHit **property** to true (this is how we will **know** when to **stop** the **game** **loop**).

Graphical user interface, text

Description automatically generated

Finally, we just have to **remove** the **rocks** from the **list**. Here we will **use** the RemoveRocks(rocksToRemove, rocks) **method**. The **method** is **not** **implemented** yet, just type its **name** and **comment** it out.

Logo

Description automatically generated with medium confidence

This is all we need for this **method**, we can **test** it **after** **implementing** the ThereIsCollision(rock, player) and RemoveRocks(rocksToRemove, rocks) **methods** because **they** work **together**, and in the **game** **loop,** we **call** **only** the MoveRocks() **method** (that means we can test the methods together). Let's continue.

#### Implement the ThereIsCollision(Rock rock, Player player) Method

The **method** will **receive** the **player** and the **current** **rock** and will **check** if they **collide** with **each** **other** (they are at the **same** **point** on the **console**). **Returns** **true** or **false** **depending** on the **outcome**.

**Create** the **method**.

A picture containing logo

Description automatically generated

Now we have to **check** if the **rock** and the **player** are the **same** **width** and **height**. Here we will **use** the RockAndPlayerAreOnSameWidth(rock, player) and RockAndPlayerAreOnSameHeight(rock, player) **methods**. The **methods** are **not** **implemented** yet, just type their **name** and **comment** it out. If they are on the **same** **width** and **height** **return** **true** **otherwise** **return** **false**.

Graphical user interface, text, application

Description automatically generated

This is all we need for this **method**, let's **implement** the **necessary** for it **methods** so it can **work** **properly**.

#### Implement the RockAndPlayerAreOnSameWidth(Rock rock, Player player) Method

The **method** will **accept** the **player** and the **current** **rock** and **check** if **any** of the **player's avatar** **characters** **collide** with the **rock** (our **player** **avatar** is **created** by **three** (3) **symbols** "(0)"). The **method** will **return** true or false **depending** on the **outcome**. We will **create** the **method** by **using** **lambda** **expressions**.

**Create** the **method** and **check** if the X **coordinate** of the **player** is **equal** to the X **coordinate** of the **current rock**. We want to do that for **every symbol** in the **player's avatar**. Try to do it yourself.



This is all we need for this method. Let's create the next one.

#### Implement the RockAndPlayerAreOnSameHeight(Rock rock, Player player) Method

The **method** will **accept** the **player** and the **current** **rock** and **check** if the **rock's** Y **coordinate** is **equal** to the **player's** Y **coordinate** **plus** **one** (1)**.** The **method** will **return** true or false **depending** on the **outcome**. We will **create** the **method** by **using** **lambda** **expressions**. Try to create the method yourself.



The **methods** are **ready**, go back to the ThereIsCollision() **method** and **uncomment** **them**. Then go back to the MoveRocks() **method** and uncomment the ThereIsCollision() **method**.

 🡪



We still can't test the **methods**, **because** we **need** to **create** one more RemoveRocks(rocksToRemove, rocks). Let's create it.

#### Implement the RemoveRocks(List<Rock> rocksToRemove, List<Rock> rocks) Method

The **method** will **accept** the **list** of **rocks** and the **list** of **rocks** **that** **has** to be **removed**. It will go **through** the **list** **of** **rocks** that **has** **to** **be** **removed** and **remove** **them** from the **list** of **rocks**.

**Create** the **method** and **open** a foreach loop **which** will **iterate** **through** the **list** of **rocks** **that** **has** to be **removed**.

Graphical user interface, text

Description automatically generated with medium confidence

In the loop **remove** the **rock** from the **list** **of** **rocks**.

A picture containing text

Description automatically generated

This is all we need for this **method**. Now **go** **back** to the MoveRocks() **method** and **uncomment** this **one**.



Now **before** we **test** the **methods**, go **back** to the **game** **loop** and **uncomment** the MoveRocks() **method**, also the last **row** of the **loop** **where** we **stop** the **method** for a **given** **amount** of **time**.

A picture containing logo

Description automatically generated

**Run** the **application** and see if it **works** **properly**.

Background pattern

Description automatically generated with medium confidence

We can see that we **have** a **fully** **functioning** **game**. We can **move** our **player**, the **rocks** are **falling** and the **score** is **updated** with **every** **passed** **rock**. But we have **one** **more** **thing** to do. To **stop** the **game** **when** the **player** is **hit** by a **rock**. We have the **logic** **ready** in the **game** **loop**, but **before** we **uncomment** it we have to **implement** the **last** **method** EndGame().

#### Implement the EndGame() Method

The **method** will **set** the **cursor** **position** to a **specific** **coordinate** and **write** "**GAME OVER!!**".

**Create** the **method** and an int **variable** which will **contain** the **specific** **coordinate**. The specific coordinate is **thirty-three** (33). This is the **last** **row** in the **score** **panel**.

A picture containing text

Description automatically generated

Now **set** the **cursor** **position** to **zero** (0) and as a **second** **parameter** give thenewly created **variable**. After that **write** on the **console** "**GAME OVER!!**".

Text

Description automatically generated with medium confidence

This is all we need for this last **method**. Go **back** to the **game** **loop** and **uncomment** the **last** **part**.



This is what the **game** **loop** **should** **look** like. Now run the application and test it again.

Background pattern

Description automatically generated with medium confidence Background pattern

Description automatically generated

We can see now that, when the **player** is **hit** by a **rock**, the **game** **stops** and the "**GAME OVER!!!**" **text** is **written** on the **bottom** of the **score** **panel**. This is all we need for this game.

## Upload Your Project to GitHub

**Upload** your **project** changes to **GitHub**, this time it's **up** **to** **you**. **Upload** it with **TortoiseGit**, **Git** **Bash** or **Visual** **Studio**. Choose the **best** **way** for **you**.

Graphical user interface, text, application, email

Description automatically generated

## \* Modify the Code, Write Your Own Features

Now, it’s time to **play with the code** and **modify it**.

|  |  |
| --- | --- |
| Icon  Description automatically generated | This is your own project. **Be unique**. Don’t be a copy/paster!   * Implement your **own features**. * **Implement the code yourself**, using your own coding style, code formatting, comments, etc. * Make the project **more interesting**. Learn by playing with the code and adding your own changes. |

Below are a few **ideas** of what you can implement or modify as an addition to your code.

### Add Unicode Characters

Search for a **more** **interesting** **character** to **represent** the **player** and the **rocks**.

**Unicode characters table** – <https://www.rapidtables.com/code/text/unicode-characters.html>.

### Add Multiplayer

**Consider** a way of **adding** **one** **more** **player** so the **game** can be **played** by **two** **users**.

### Reverse the Game

One more idea is to **reverse the game** – you have your **player on the first row** (on the top of the console) and **rocks coming** **from below** (from the bottom of the console).

### Create GUI App

You can create a **GUI app** with **WinForms** for the game.

### Additional Ideas

* **Create** an **improvised** **database**, so you can **store** the **user's** **name** and **score** and **display** them on the **console**.
  + Use JSON - <https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/how-to?pivots=dotnet-6-0>.
  + Use XML - <https://learn.microsoft.com/en-us/dotnet/standard/serialization/how-to-serialize-an-object>.
* You can add anything else to your code, based on your own ideas.

### Commit to GitHub

Now **commit and push your code changes** to your GitHub repo!

|  |  |
| --- | --- |
| Icon  Description automatically generated | It is very important to **commit frequently** your code to GitHub. This way you create a **rich commit history** for your project and your GitHub contribution graph is growing:  A picture containing chart  Description automatically generated |

## Write a README.md File

It's highly recommended to provide **documentation as part of your project on GitHub** to describe what the project does. So, let's make one for this **project**. Let's start by editing the README.md file from our repo on GitHub:

Graphical user interface, text, application, email

Description automatically generated

### Documentation Sections

Add **information** about your project in your README.md file using **Markdown**: project goals, technologies used, screenshots, live demo, etc. Typically, you should have the following **sections**:

* **Project title** (should answer the question "What’s inside this project)
* **Project goals** (what problem we solve, e. g. we implement a certain game)
* **Solution** (should describe how we solve the problem 🡪 algorithms, technologies, libraries, frameworks, tools, etc.)
* **Source code link** (give a direct link to your source code)
* **Screenshots** (add screenshots from your project in different scenarios of its usage)
* **Live demo** (add a one-click live demo of your code)

### Project Goals

Start your documentation by describing your **project goals**. What problem does your project solve?

### Sample Documentation

This is an **example** of how you can document your project. **Don’t copy-paste**Graphical user interface, text, application, chat or text message

Description automatically generated

|  |  |
| --- | --- |
| Icon  Description automatically generated | **Write the project documentation yourself**. Don’t copy/paste it!  This is your **unique GitHub profile** and your own unique project. **Be different** from others. |

Find an **appropriate** **image** and add it. You can add **images** as follows:



You can add information about the **inputs** and **outputs** of the project:

Graphical user interface, text, application

Description automatically generated

### Controls

Add information about the **controls of the game**.

Graphical user interface, text, application

Description automatically generated

### Your Solution

Describe how you **solve the problem**: algorithms, technologies, libraries, frameworks, tools, etc.

### Link to the Source Code

Add a **link** to your **source code** as follows:



### Screenshots

Add **screenshots** of your project:

1. **Take a screenshot** with your favourite tool (e.g. the [Snipping Tool](https://support.microsoft.com/en-us/windows/open-snipping-tool-and-take-a-screenshot-a35ac9ff-4a58-24c9-3253-f12bac9f9d44) in Windows).
2. **Paste** the screenshot in the GitHub Markdown editor, using [Ctrl+V]:

Example screenshots for the "FallingRocks" game:

A picture containing timeline

Description automatically generated

## Upload Your App to Replit

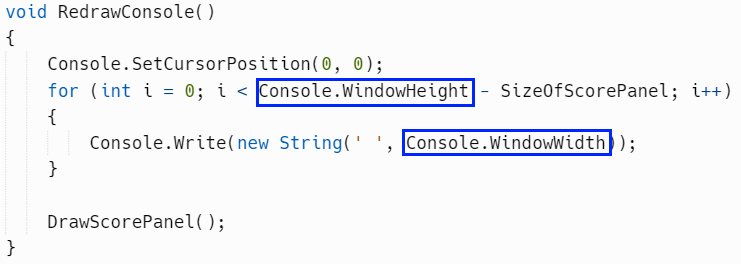
Let’s add our **project** to **Replit** so we can share it with our **friends** and add it to our **GitHub** profile. You already should know how to do that.

We have two **options** to **choose** from, first **option** is to use **Mono C#**. **Mono C#** is not **updated** yet and still works with **.NET 5.0**. This **means** we should **remake** our **code** to **.NET 5.0** in order to **work** in **Mono C#**. The **second** option is to **create** a **C# replit project**. The **C# replit project** is working with **.NET 6.0** and we can **simply** **copy** and **paste** our **project** **files** into it. The only **difference** in the **final** **result** is that the **Mono C# is working a little faster** than the **C# replit project**.

**Replit** **can't** work with Console.SetWindowSize() and Console.SetBufferSize(), we have to **comment** **them** out (we won't use them).Also **everywhere** we use our **constants** WindowHeight and WindowWidth, we **should** **replace** **them** with Console.WindowWidth and Console.WindowHeight. This is **because** the **Replit** **console** is **set** to a **certain** **width** and **height** and is **not** **allowed** to be **changed**.

Graphical user interface, text

Description automatically generated



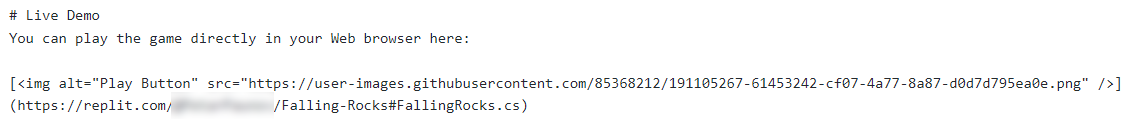
Graphical user interface, application, Word

Description automatically generated

Choose an option and do it yourself so you can **share** your app with your friends.

## Add Replit Link to Your README.md

Now add a "**one-click live demo**" of your project from your **GitHub** project documentation. You can do it as follows:



You can take a **screenshot** from Replit.com and **paste it** into the **GitHub** documentation editor directly with **[Ctrl+V]**.

This is what it should look like after the changes in your README.md documentation:

Graphical user interface, text, application

Description automatically generated