# 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

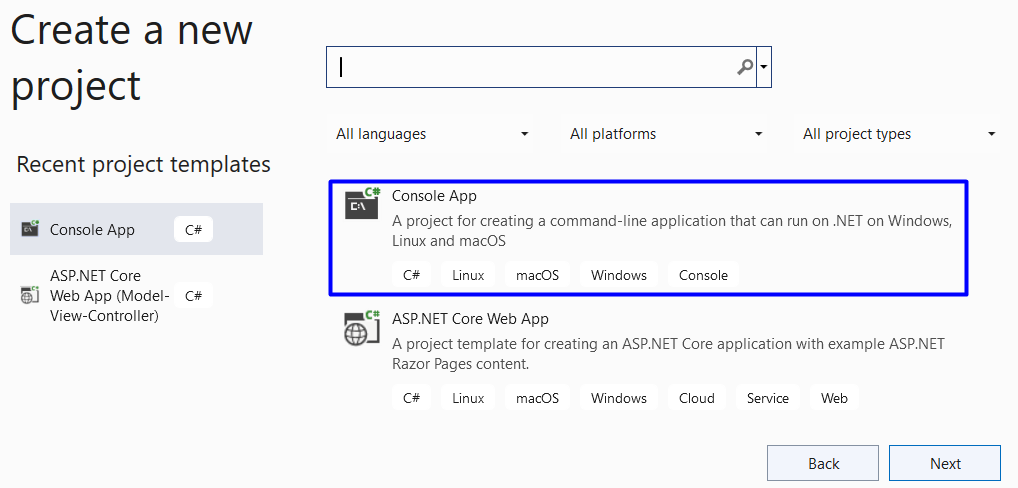
Description automatically generated

## 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**.

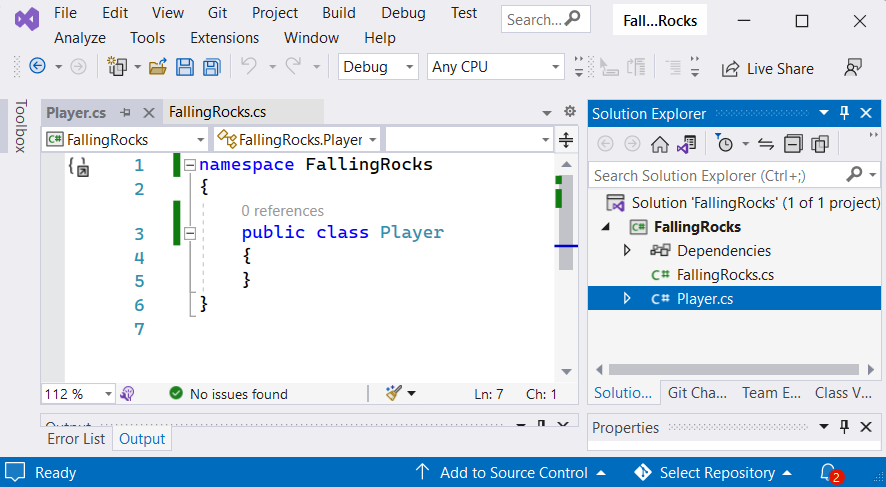
* int ProcessPlayerRocksSpawnRateChoice(string inputSpawnRate) – обработва **входа** на играча за **скоростта на генериране на камъни**
* int ProcessPlayerRockFallSpeedChoice(string inputFallSpeed) – обработва **входа** на потребителя за **скоростта на падане** на камъните
* void SetWindowProperties() – 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

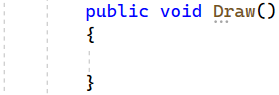
Description automatically generated with low confidence

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

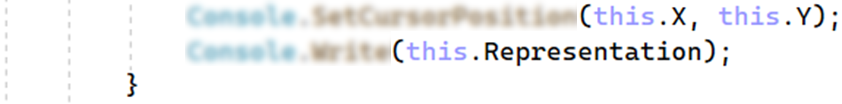


Това е всичко, от което се нуждаем за нашия **конструктор**. Следващата стъпка е да **създадем методите**.

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**.

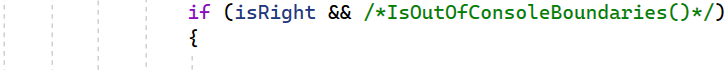


Сега трябва да създадем **две булеви променливи**. Първата ще проверява дали **натиснатият клавиш** е **дясната стрелка** (→), а втората ще проверява дали **натиснатият клавиш** е **дясната стрелка** (→). Тези проверки за **важни**, защото ако ги нямаме, играта ще се **счупи**, ако потребителят натисне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

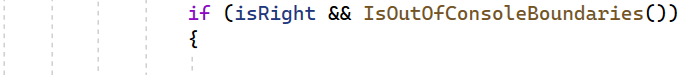
Description automatically generated

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**.

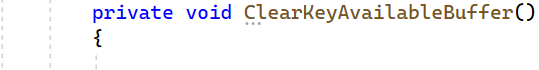
Трябва да изглежда така:



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**.

### Клас Rock

Създайте **нов клас** и го наименувайте **Rock**. Вече знаете как да направите това:

A picture containing text

Description automatically generated

Първо, нека да добавим **три свойства**. **Първите две** ще бъдат от тип **int** и ще представляват **X** и **Y координатите** на **камъка**.

Text

Description automatically generated with low confidence

**Третото свойство** ще бъде от тип **char** и ще включва **символа за камъка**.



Преди да **създадем конструктора**, се нуждаем от **две частни полета**. Първото ще бъде от тип char[] и ще съдържа **всички различни символи за камъка**.



**\* Бележка:** Може да **копирате** **тези символи** или да **добавите ваши**, така че проектът ви да е **по-различен** от този на вашите съученици.

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

Следващото **частно поле** ще бъде от тип **Random** и ще ни помогне да вземем **нов символ** всеки път, когато **създаваме нов камък** (за да може символите на падащите камъни да бъдат **различни**).



Вече имаме **всички свойства** и **полета**. Остава да добавим **конструктор** на класа.

Създайте **конструктор** и го направете така, че да **приема X координата** на камъка. Тук **нямаме нужда от Y координата**, защото камъните винаги **започват да падат** от **Y = 0**.

A picture containing text

Description automatically generated

Сега в конструктора трябва да зададем **стойност на свойствата**, както в предишния клас. Този път **Y координатата** ще е **равна на 0**.

Icon

Description automatically generated

Свойството Symbol ще бъде равно на **елемент** на **случаен индекс** от нашия **Representation масив**. Вече знаете как да го вземете:



Това е **всичко**, от което се нуждаем в класа **Rock**. Нека да продължим с **инициализацията** на **играта**.

## Инициализация на играта

### Подготовка на настройките

Първо, кликнете на файла "FallingRocks.cs" в [SolutionExplorer]. В този файл ще **инициализираме играта**. Първо ще имплементираме **главната логика** на играта, като ще **закоментираме** някои части. След като **имплементираме методите**, ще ги **откоментираме** и ще **тестваме** играта.

Нуждаем се от **три** int **константи**, които ще ни помогнат да **зададем ширина** и **височина** на конзолата, както и **размера** на **панела с резултати**. Ще използваме тези константи **по-късно**.

Text, chat or text message

Description automatically generated

**\* Бележка:** Може да използвате **същите числа** или да зададете **различни**.

Следващата променлива е от тип int и ще пази **резултата** на потребителя (променливата ще се **увеличава с 1**, когато играчът **успешно заобиколи камък**).



Сега трябва да **прочетем входа** на потребителя за **скоростта** на **генериране** на **камъни** и **скоростта** на **падане** на всеки камък (трудността на играта). Вместо да пишем всичко тук, нека да създадем **два метода**, които да отговарят за това. По този начин правим кода **по-четим**.

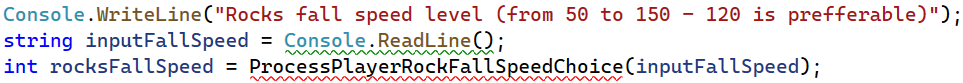
Преди да използваме **първия метод**, трябва да **напишем съобщение** на **конзолата**, за да обясним на потребителя **какво да въведе**, след което да **прочетем входа**.



Сега създайте int **променлива** за **скоростта** на **генериране** на камъни и я направете **равна** на **резултата от** ProcessPlayerRocksSpawnRateChoice(string inputSpawnRate).



Направете същото нещо за **скоростта** на **падане**. За целта използвайте метода ProcessPlayerRocksFallSpeedChoice(string inputFallSpeed).



**\* Бележка** – Закоментирайте **методите**, за да не получите **грешки**.

Следващата стъпка е да зададем свойствата на **конзолния прозорец**. Ще използваме метода SetWindowProperties().



Сега се нуждаем от **още три** int **променливи**, които ще ни помогнат да зададем X и Y координатите на **играча**. За да вземем **X координатата**, се нуждаем от **ширината** на конзолния прозорец, **разделена на 2**.



За да вземем Y **координатата**, се нуждаем от **височината** на конзолния прозорец минус **размера** на **панела с резултати** (имаме константа за това).



За да вземем **точната Y координата**, трябва да **извадим 1** от резултата (иначе играчът ще се **позиционира** на **реда** на **панела** с **резултати)**.



След като имаме X и Y координатите, сме готови да **създадем инстанция** на **играча**. Този път, когато създаваме инстанцията, ще имаме **два параметъра**. Те идват от **конструктора** на **класа Player**, който създадохме **по-рано**.

A picture containing application

Description automatically generated,

Това, което трябва да направим, е да **подадем предишните две променливи**, които създадохме.



Това е всичко, от което се нуждаем, за да **създадем инстанция** на нашия **играч**. Последното нещо, преди да разработим **цикъла на играта**, е да създадем List<Rock>, в който ще **добавяме новите камъни**, преди да **паднат** и да се опитат да **убият играча**.



Сега сме **готови** да продължим с **цикъла на играта**.

### Инициализация на цикъла на играта

Сега имаме **всичко подготвено** за **цикъла** на **играта**. Играта започва с цикъла и **свършва**, когато **потребителят загуби**.

Първо, отворете един **while-цикъл**.

Graphical user interface, application

Description automatically generated

Първият **метод**, от който ще се нуждаем в цикъла, е RedrawConsole(). Този метод само ще **пренарисува конзолата**. Засега **закоментирайте метода**, докато дойде време да го **имплементираме**.



Следващата ни задача е да **създадем камъните**. Ще използваме **метода** CreateRocks(), но засега ще го **закоментираме**.



Сега трябва да **нарисуваме играча**. За тази цел ще използваме метода Draw() **метода**, който създадохме по-рано в **класа** **Player**. Ще достъпим метода от **инстанцията** на **класа** **Player**, която създадохме преди.



След това трябва да **нарисуваме камъните**. За да го направим, ще използваме **метода** DrawRocks(). Нека засега да го **закоментираме**:



Сега **играчът** и **камъните** са подготвени. Следващата задача е да **започнем** да ги **движим**. Ще **местим** **играча**, като извикаме неговия Move() **метод**, а **камъните** ще **местим** с **метода** MoveRocks().

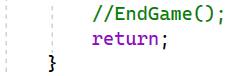


Последното нещо, което трябва да направим, е да **проверим** дали играчът е **ударен от камък**. Ще го направим, като използваме **свойството** HasBeenHit.

A picture containing logo

Description automatically generated

Ако играчът е бил **ударен**, ще използваме метода EndGame(), който ще изпише "**Game Over**" на конзолата и ще **прекрати цикъла**.



Накрая ще използваме **метода** Thread.Sleep(), който **спира цикъла** за даден **брой милисекунди**. Милисекундите ще бъдат **определени** от **потребителя** и ще се съхраняват в **променливата** rockFallSpeed.



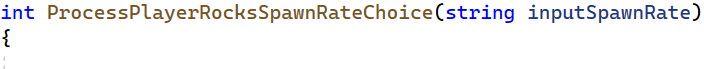
Това е всичко, което ни трябва за **цикъла** на **играта**. Нека сега да започнем **имплементацията** на **методите**.

### Имплементация на методите

#### Имплементация на метода ProcessPlayerRocksSpawnRateChoice(string inputSpawnRate)

Този **метод** ще обработвапотребителския **вход** за **скоростта** на **генериране** на **камъни**. Трудността ще бъде от **едно** (1) до **десет** (10), ще имаме и **предпочитана опция**, която ще бъде **четири** (4). Когато потребителският вход е **обработен**, методът ще върне **стойността**.

Създайте метод, който **приема скоростта**, избрана от потребителя.



Следващата стъпка е да **валидираме** потребителския вход. Обмислете начин да го направите **сами**.

**Насоки за валидация:**

* Входът трябва да бъде **цяло число**
* Числото трябва да бъде по скалата от **1** до **10**
  + Не по-малко от **едно**
  + Не повече от **десет**
  + Ако числото е **по-голямо от 10** или **по-малко от 1**, направете го **равно на предпочитаната** опция, т.е. **4**

**Върнете резултата**:

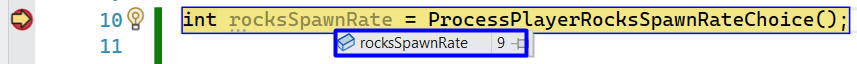
Graphical user interface, application, Teams

Description automatically generated

Когато сте **готови** с **метода**, се върнете в **цикъла** на **играта**, **откоментирайте** го и го **тествайте**, за да проверите дали **работи правилно**.

Graphical user interface, text

Description automatically generated

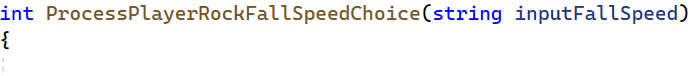


Нека да продължим със следващия.

#### Имплементация на метода ProcessPlayerRockFallSpeedChoice()

Този метод ще обработва потребителския **вход** за **скоростта** на **падане** на **камъка**. **Трудността** ще бъде от **петдесет** (50) до **сто и петдесет** (150) и ще имаме **предпочитана опция** от **сто и двадесет** (120). Когато

Създайте **метод**, който приема **входа** за **скоростта** на **падане**.



Сега обмислете как да я **валидирате**.

**Насоки за валидация:**

* Входът трябва да бъде **цяло число**
* Числото трябва да бъде по скалата от **50** до **150**
  + Не по-малко от **50**
  + Не повече от **150**
* Ако числото е **по-голямо от 150** или **по-малко от 50**, направете го **равно на предпочитаната** опция, т.е. **120**

Graphical user interface, application, Teams

Description automatically generated

Преди да **върнем скоростта** на **падане**, трябва да направим **още едно изчисление**. Тъй като работим с **милисекунди**, ще възникне **грешка**. Ние искаме **50** да е **най-бавната** скорост, а **150** да е **най-бързата**. Само че ако въведем **50**, играта ще върви **по-бързо** (защото 50 милисекунди са по-малко от 150). За да **разрешим проблема**, можем да **обърнем логиката**.

**Насока** – **Общата скорост** на падане на камъка е **двеста** (200).

Опитайте се да разрешите проблема сами и да **върнете стойността**.

Graphical user interface

Description automatically generated

Когато сте **готови с метода**, се върнете в **цикъла** наиграта, **откоментирайте** го и го **тествайте**.

Text

Description automatically generated



Нека да продължим със **следващия метод**.

#### Имплементация на метода SetWindowProperties()

Този метод ще **задава свойствата**, които ни трябват за **конзолата**. **Ширината** и **височината** на конзолата, както и **видимостта** на **курсора**, ще бъдат зададени на **false**. Методът ще прави настройки **преди играта** да **започне**.

Създайте **метода**, **изчистете конзолата** и задайте **видимостта** на **курсора** на **false**.

Text

Description automatically generated

Сега ще използваме **два вградени метода**, които ще ни помогнатда зададем **ширината** и **височината** на конзолния прозорец. Първо ще използваме метода Console.SetWindowSize(). Като стойности ще **подадем** константите WindowsHeight у WindowsWidth, които създадохме по-рано.



**Повече** за метода: <https://learn.microsoft.com/en-us/dotnet/api/system.console.setwindowsize?view=net-6.0>.

Вторият метод е Console.SetBufferSize(). Той ще задава **височината** и **ширината** на екранния **буфер** на конкретни стойности. **Екранният буфер** е **двумерен масив** с данни за **символи** и **цветове** за **изхода** на конзолата. Ще подадем **същите** стойности като в **предишния метод**.



**Повече** за метода: <https://learn.microsoft.com/en-us/dotnet/api/system.console.setbuffersize?view=net-6.0>.

**Повече** за екранния буфер: <https://learn.microsoft.com/en-us/windows/console/console-screen-buffers>.

Това е всичко за този метод. Върнете се в **цикъла** на **играта**, **откоментирайте** го и го **тествайте**.

Ако всичко работи правилно, **конзолният прозорец** трябва да се **събере** до **подадените стойности** веднага след като ги **въведете**.

Text

Description automatically generated 🡪Text

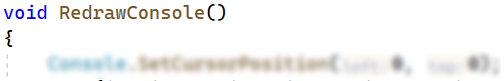
Description automatically generated

Нека да продължим със **следващия метод**.

#### Имплементация на метода RedrawConsole()

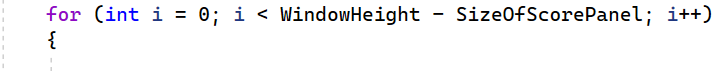
Този метод ще **прерисува** конзолата **без да я чисти**. Това означава, че ще **започнем** от **горния ляв ъгъл** на конзолния прозорец и ще рисува **празен стринг** до **края** на **конзолата**. Това ще ни помогне да **избегнем мигането** на **конзолата**, което е причинено от **метода** Console.Clear().

Създайте метод, който **позиционира курсора** на координати **(0, 0)**, т.е. **горния ляв ъгъл** на конзолата.



Следващата стъпка е да създадем **for-цикъл**, който ще итерира, докато **i е по-малко** от **височината на прозореца минус размера** на **панела** с **резултати**.

Ще извадим **размера** на **панела** с **резултати** от **височината** на **прозореца**, защото панелът **винаги** ще бъде на **едно и също място** и ще си остане там **до края на играта**.



В цикъла ще напишем **интервал петдесет пъти**, което е **ширината** на **прозореца** (имаме константа за това):



Последното нещо е да използваме **метода** DrawScorePanel() – методът ще чертае **панела с резултати** на конзолата. Тъй като **още не е имплементиран**, само го **извикайте** и го **закоментирайте**.

A picture containing logo

Description automatically generated

Това е всичко за този метод – можем да го **тестваме**,след като имплементираме **метода** DrawScorePanel(), защото **двата метода работят заедно**. Нека да продължим с него.

#### Имплементация на метода DrawScorePanel()

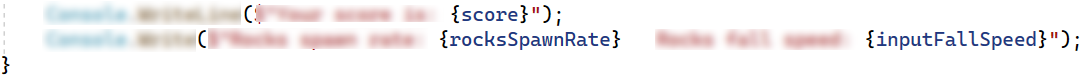
Методът ще **разделя бойното поле** (горната част, откъдето падат камъните) със символа "**=**" и ще визуализира на конзолата **резултата** на потребителя, **скоростта** на **падане** на камъните, както и **скоростта** на **генериране** на **камъни**, избрана от потребителя.

Създайте **метода** и измислете начин да нарисувате символа "**=**" през **цялата ширина на прозореца**.

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

Description automatically generated

Остава да **визуализираме резултата** на потребителя, **скоростта** на **падане** на камъните и **скоростта** на **генериране** на **камъни**. За визуализация на **скоростта** на **падане** ще използваме стойността, избрана от **потребителя**, иначе ще покажем **променената стойност** на метода ProcessPlayerRocksFallSpeedChoice(string inputFallSpeed). Можете да направите това сами – имате **необходимите променливи**, остава да измислите **съобщение**, което да визуализирате.



Това е **всичко за този метод**. Върнете се в **метода** RedrawConsole() и го **откоментирайте**, след това се **върнете** в **цикъла** на **играта** и **откоментирайте метода** RedrawConsole(), за да може да го **тествате**.

A screenshot of a computer

Description automatically generated with medium confidence

Ако всичко **работи правилно**, трябва конзолата да е разделена на **две части**, като долната част съдържа **данните** за **играта**. Нека да продължим със следващия метод.

#### Имплементация на метода CreateRocks()

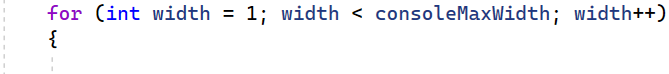
Този метод **създава камъни** в зависимост от **скоростта** на **генериране**, която потребителят е избрал, и ги добавя към **списъка с камъни**, който създадохме по-рано.

Създайте **метод** и **променлива**, които ще съдържат **максималната ширина** на конзолата **минус 1**. Ако не извадим 1, ще излезем **извън границите** на прозореца и програмата ще **спре**.

A picture containing logo

Description automatically generated

След това **създайте for-цикъл**, който ще итерира, докато **i** е по-малко от **максималната ширина** на конзолата. Това ще ни помогне да създадем камък и да **зададем** неговата **X координата** на **текущата ширина** (**i**).



Преди да **създадем камък**, трябва да проверим дали **е време да го създадем**. За тази цел ще използваме метода ShouldGenerateRock(). Тъй като **още не е имплементиран**, само го извикайте и го **закоментирайте**.

A picture containing logo

Description automatically generated

Ако е време, просто **създайте нов камък** с **текущата ширина** като параметър и го добавете към **списъка с камъни**.

Graphical user interface, text, application

Description automatically generated

Това е **всичко за този метод**. Можете да го тествате, след като имплементирате метода ShouldGenerateRock(), защото **двата метода** работят **заедно**. Нека да продължим с него.

#### Имплементация на метода ShouldGenerateRock()

Методът ще **върне true** или **false** в зависимост от случайно число между **0** и **101** и от **скоростта** на **генериране** на камъни, избрана от играча.

Създайте **метода** и **нова случайна променлива**.

Graphical user interface, application

Description automatically generated

Сега се нуждаем от **променлива** за **максималната скорост** на **генериране**, равна на 100.



След това трябва да проверим дали **случайното число** е **по-голямо** или **равно** на променливата spownRateMaxValue **минус скоростта на генериране**.



Това е **всичко за тоз метод**. Върнете се в **метода** CreateRocks() и го **откоментирайте**, след това се върнете в **цикъла** на **играта** и **откоментирайте** **метода** CreateRocks(). При **тестване** приложението трябва да работи така:

A screenshot of a computer

Description automatically generated with medium confidence

Можем да видим, че конзолата е **същата** като преди, но ако добавим **breakpoint** и **дебъгнем** приложението, ще видим, че всяка итерация добавя **повече** и **повече камъни** към списъка.

Table

Description automatically generated with low confidence 🡪 Table

Description automatically generated

Нека да продължим със **следващия метод**.

#### 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

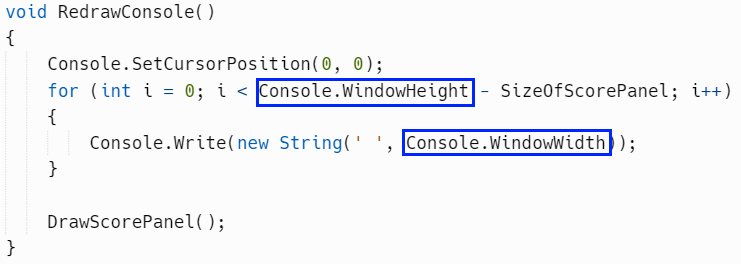
Let’s add our project to Replit so we can share it with our friends and add it to our GitHub profile. You already should know how to do that.

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



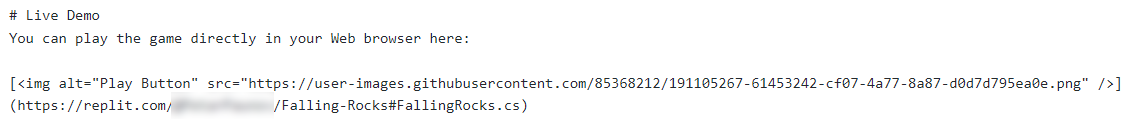
Graphical user interface, application, Word

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