# Практически проект: Hangman

Днеска ще играем играта "Hangman":

Diagram

Description automatically generated

[Hangman](https://en.wikipedia.org/wiki/Hangman_(game)) е **известна игра** от миналото. Целта на играта е да се **познае скритата дума**. Ще получите **празни места**, които са **неизвестните букви в думата**. Използвайте **клавиатура**, за да пишете.

* Ако **избраната буква** съществува в думата, **всички празни места** на дадената **буква**, ще бъдат **разкрити**.
* Имайте предвид, че когато **напишете грешна буква**, губите част от **живота** си. При всеки грешен опит на конзолата се отпечатват част **човек**.Когатосе **изпълни цялата фигура** на човека под бесилото, вие ще загубете играта.
* **Разгадайте думата, преди човекът да бъде обесен.**

Text

Description automatically generated Text

Description automatically generated

## Създаване на GitHub Repository

Тъй като вече имате акаунт в **GitHub**, можете директно да създадете ново repository-и.

Създайте **ново** **repository-и** от: <https://github.com/new>. Изберете **смислено име** като "HangmanByUsername". След това добавете **кратко описание** и сложете **repository-то** на **public**:

Graphical user interface, text, application, email

Description automatically generated

|  |  |
| --- | --- |
| Icon  Description automatically generated | Моля изберете **ваше собствено и уникално име** за проекта!  Вашето име на GitHub профила трябва да бъде **уникално** и различно на този от вашите съученици.  Може да следвате **ръководството** или да **направите промени** и **да имплементирате** вашия проект **различен** от този на вашите съученици. |

Също така добавете файла README.md и .gitignore за **Visual Studio**, както е показано по-долу:

A screenshot of a computer

Description automatically generated

В Git проекта, файлът **.** gitignore показва кои файлове не са част от source кода и трябва да бъдат игноирани (да не се качват в **repository-то**). Обикновено в **GitHub** качваме **само source кода**. Не качваме компилирани или временни файлове.

Накрая **променяме лиценза** на "MIT " (това е най-популярния използван open-source линценз) или друг лиценз по избор. След това натискаме бутона [Create] за **създаване** на **вашете repository-и**:

A screenshot of a computer

Description automatically generated

Така трябва да изглежда **създаденото** **repository-и**:

Graphical user interface, text, application, email

Description automatically generated

Сега да започнем да **пишем кода** за нашата игра.

## Писане на кода за игра

Нека да създадем играта и да я играем.

### Създаване на проект във Visual Studio

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

A screenshot of a computer

Description automatically generated

После избираме **подходящо име** и **папка за проекта**. Трябва да отбележите [Place solution and project in the same directory],за да нямаме допълнително папки в нашите файлове. После кликнете на [Next]. На следващия екран трябва да изберете [.NET 6 (Long-term support)] и да натиснете [Create]:

Нашият **проект трябва да бъде създаден** и да изглежда така:

Graphical user interface, text, application

Description automatically generated

Преди да продължим, трябва да променим името на нашият **основен клас** – Program.cs на нещо **смислено**. Направете го така:

Graphical user interface, application

Description automatically generated 🡪 Graphical user interface, text, application, email

Description automatically generated

Нека да започнем да работим по нашия код.

### Стъпки за имплементиране на играта

Ще започнем със **създаването на променливи** и **константи**, които ще ги **използвате за анимации на играта**. **В ресурсите** сме ви **осигурили** **документ**, който **съдържа** **всички** **низове**, от които ще се нуждаете за **създаването на тези анимации**. Трябва да създадете:

* **Два константни низа**, които ще съдържат анимацията за **победа** и **загуба**.
* **Два масива от низ**, които ще съдържат кадрите за **грешно позната буква** и **анимацията за смърт**.

След това ще създадем **метод** ReadWordsFromFile(), който ще чете **всички думи**, които ще използваме за четене за играта Hangman, от външния осигурен файл **words.txt**.

Да започваме с инициализирането на играта. Помислете за методите, които ще ни трябват, за да работи играта.

Накрая ще имплементираме осем метода, които ще си кореспондират помежду си, за да **работи играта**.

1. string GetRandomWord(string[] words) – **метода** ще ни помогне на да **вземем дума на случаен принцип**.
2. void DrawCurrentGameState(bool inputIsInvalid, int incorrectGuess, string guessedWord, List<char> playerUsedLetters) – **метода** ще ни помогне да **отпечатаме** текущото състояние на играта на конзолата.
3. void PlayGame(string word, string wordToGuess, int incorrectGuessCount, List<char> playerUsedLetters) – **метода** ще изпълнява **основната логика** на **играта**.
4. bool CheckIfSymbolIsContained(string word, char playerLetter) – **метода** проверява дали **въведеният символ съществува** в **думата**.
5. string AddLetterToGuessWord(string word, char playerLetter, string wordToGuess) – метода добавя **въведеният символ** в **думата**, ако **той съществува**.
6. bool CheckIfPlayerWins(string wordToGuessChar) – **методът** проверява дали **играчът** е **победил** в **играта**.
7. bool CheckIfPlayerLoses(int incorrectGuessCount) – **методът** проверява дали **играчът** е **загубил** в **играта**.
8. void DrawDeathAnimation(string[] deathAnimation) – **методът** ще ни помогне с **рендирането на анимация за смърт**, когато играчът **загуби**.

Това е всичко, за да **завършим нашия проект**. Сега да започнем с анимацията на играта.

### Създаване на анимации за играта

Започваме с отварянето на документа "**06. Project-Hangman-Game-Animations.docx** ". Сега се върнете на проекта във **VS** и създайте **два константи низа** с име **"Win"** и **"Loss"**. Копирайте **низовете за анимация** от **документа** и ги поствавете като **стойности** на низвете "**Win**" и "**Loss**". Накрая низовете трябва да изглеждат така:

A picture containing text

Description automatically generated

След това създайте два масива от низ, които ще съдържат анимацийте за грешно позната буква. Наименувайте първия масив с име "wrongGuessesFrames", копирайте **низовете за анимация** от **документа** и ги поствавете между **скобите** на **масива**. Накрая масивите трябва да изглеждат така:

Chart, box and whisker chart

Description automatically generated

A picture containing chart

Description automatically generated

A picture containing diagram

Description automatically generated

Наименувайте втория масив "deathAnimationFrames", копирайте **низовете за анимация** и ги поставете **между** скобите на **масива**. Вече знаете как да направите това. Това трябва да бъде резултатът:

A screenshot of a computer

Description automatically generated

### Четене на думите от външен файл

В ресурсите сме ви осигурили файла words.txt, който **съдържа** над **2999 различни думи**. Ще прочетем всички **думи** от файла (words.txt) и ще ги **добавим** в **масив от низ**. Първо **копирайте осигурения файл** и го **поставете** в **директорията на проекта**. Директорията трябва да изглежда така:

Graphical user interface, application

Description automatically generated

Сега се върнете на **проекта** във **VS**. Създайте **метод** и го наименувайте ReadWordsFromFile(). В **него** ще трябва да прочетете **думите** от файла "**words.txt**" и да ги **върнете** като **масив от низове**.

A picture containing text

Description automatically generated

В **метода** първо ще създадете променлива от тип **низ**. **Тя** ще **съдържа** **директорията** на компилацията на програмата. Използвайте следния метода, за да **взмете сегашната директорията**:

|  |
| --- |
| Directory.GetCurrentDirectory() |

**Методът** ще ни помогне да имаме **достъп до сегашната директорията** на компилацията на програмата в нашия проект и ще върне резултата като низ. Примерно резултатът трябва да бъде така"D:\My Projects\03.Project-Hangman\bin\Debug\net6.0".





След това ще създадем още една **променлива от тип низ**. Тя ще съдържа **директорията на проекта**. Използвайте метода по-долу, за да върнете името на **родител-родител-родител директория** (директорията, където се намира файла **words.txt** ):

|  |
| --- |
| Directory.GetParent().Parent.Parent.FullName |

**Методът** ще ни помогне да имаме достъп до **директорията на проекта**, която съдържа файла **words.txt** (Примерно **D:\My Projects\03.Project-Hangman**).





Следващото нещо е да имаме **достъп до директорията** нафайла **words.txt**. За да направим това, **трябва да създадем константа**, която ще е **името** на **файла**.



Сега **обединяваме** **променливата и директорията** с името на файла **words.txt**, за да създадем пътя до **файла**. Пътя трябва да изглежда така:





Последния нов **метод**, който ще използваме е за **четене на текст от файл**:

|  |
| --- |
| File.ReadAllLines() |

Сега създайте **масив от низ**, който ще **съхранява** **прочетения текст** като всеки ред ще съдържа **само** **една** **дума**.



Накрая връщаме думите в масив.



Нашият пръв **метод** е вече готов. Сега може да го извикаме и да видим да ли **работи правилно**. Създайте **масив от низ** и извикайте метода.



Проверете дали всичко работи правилно. Затова нека да **дебъгнем** нашата **програма**.

A close-up of a sign

Description automatically generated

След като програма отиде на **breakpoint-а** на **масива с думи** трябва да имаме **2999 думи.**

Преди да продължим, сложете "Console.CursorVisible" на false , защото по-късно ще **рисуваме** на конзолата. **Console.CursorVisible** ще скрие **белия курсор,** който **показва на кой ред пишем в момента**. Причината е, че не искаме той да се вижда, докато играем играта.



Можете да научите повече за методите на:

* **Directory.GetCurrentDirectory()**:<https://docs.microsoft.com/enus/dotnet/api/system.io.directory.getcurrentdirectory?view=net-6.0>.
* **Directory.GetParent(string)**:<https://docs.microsoft.com/en-us/dotnet/api/system.io.directory.getparent?view=net-6.0>.
* File.ReadAllLines(path): <https://docs.microsoft.com/en-us/dotnet/api/system.io.file.readalllines?view=net-6.0>.

Нека да **продължим** с **иниалицизирането на играта**.

### Инициализиране на играта

След като сте нагласили курсора, създайте **while-цикъл**. В него ще се състои цялостния **поток на играта**.

A picture containing logo

Description automatically generated

В него, създайте **променлива от низ**, която ще **съдържа случайно избрана дума**. Променливата извиква метода "GetRandomWord()" с параметър **масива с думи**. **Методът** помага за **избирането на случайни думи всеки път**, когато **играта започна отново**.



**Методът** ще бъде **подчертан** с **червено**. Това е така, защото не е **имплементиран**. Не се притеснявайте. Когато сме готови с инициализирането на играта ще **коментираме** кода и ще **премахнем** **коментарите** бавно с **имплементирането на методите**.

След това създайте още една **променлива от тип низ**, която ще **представлята избраната дума**, но всички **букви ще бъдат заменени със символ** по ваш избор (с цел да скриете буквите от играча). В нашия случай ще използваме **долно тире**. Можете да направите така:



**\*Забележка**: Можете да **създадете** **константа** за символа **долно тире**, защото по-късно ще го **използваме в кода**. Знаете как да направите това.

Също така, създайте **променлива от тип int**, която ще съдържа **броя грешни опита** на въвеждане на символи и List<char>, който ще съдържа всички **букви въведени от потребителя**. List<char> е масив от тип символ с **променлива дължина** и съдържа **методи**, които ни помагат с **добавяне**, **изтриване**, **сортиране** и **търсене** на **елементи**. Можете да научите на: <https://www.geeksforgeeks.org/c-sharp-list-class/>.

Ще използваме тези **променливи** за **рисуване на конзолата**.



Ще създадем още един метод "DrawCurrentGameState**()**". Той трябва да **приема** **четири параметъра**:

1. bool inputIsValid – **булева променлива**, която ще казва дали **входа на потребителя** е **валиден или не**.
   1. **False** 🡪 валиден вход.
   2. **True** 🡪 грешен вход.
2. int incorrectGuess – променлива за **броя грешни въвеждания**.
3. string guessedWord – думата за отгатване.
4. List<char> playerUsedLetters – променлива за **символи**, **използвани от потребителя**Text

   Description automatically generated with low confidence

**Методът** ще ни помогне с **нарисуването** на **сегашния статут** на **играта** на **конзолата**. Трябва да **извикаме** **метода** и да **подадем необходимите методи** по следния начин:



След като **първия статус** на **играта** е **нарисуван**, трябва да създадем още един метод – PlayGame(). Методът ще приема **четири** **параметъра**:

1. string word – променлива за **случайно избраната дума**.
2. string wordToGuess – променлива за думата за отгатване.
3. int incorrectGuessCount – променлива за **броя грешни въвеждания**.
4. List<char> playerUsedLetters – променлива за **символи**, **използвани от потребителя**.

A picture containing text

Description automatically generated

Този **метод** ще **изпълнява основната логика на играта**. Трябва да **извикаме** **метода** и да **подадем необходимите методи** по следния начин:



След като приключим с това, отстава да **попитаме играча** **да ли иска да играе отново** или да **излезе от играта**. Тази логика може да я **имплементирате** **сами**.

Graphical user interface, text, application

Description automatically generated

Вече инициализирахме играта и можем да коментираме кода, за да имплементираме и тестваме методите, които създадохме. За да направите това, може да **маркирате** **кода** в **while-цикъла** и да **натиснете** [Ctrl + K + C].

A screenshot of a computer

Description automatically generated with medium confidence

Ще **махнем коментарите** от **кода** бавно с **имплемнетирането** на **методите**. Нека да започнем с **първия**, който ще ни помогне с избирането на **дума** на **случаен принцип**.

#### Имплементиране на метода GetRandomWord()

Създайте метод с името "GetRandomWord()", който приема **масив от низове**. **Методът** трябва да **избира** **дума** на случаен индекс. Кодът трябва да изглежда така:

A close up of a word

Description automatically generated

Създайте променлива от тип "Random". Вече знаете как да направите това:



Сега създайте променлива от тип низ я наименувайте "word". Това ще бъде **нашата дума** на **случаен принцип**. Използвайте **променливата от тип Random**, за да генерирате **случаен индекс**. **Накрая** върнете **думата** с командата **.ToLower()**, за да **направите** думата с **малки букви**:



Нашият **метод** е **готов**. Върнете се в **цикъла** и **махнете коментара** на **първия ред** и сложете **breakpoint** на **метода**.

Timeline

Description automatically generated

Timeline

Description automatically generated

**Всеки път**, когато натисните [F10], трябва да ви изписва **различна дума**. Това е всичко, което ни е необходимо, за да работи **нашият метод**. Сега можем да **разкоментираме кода до следващия ни метод**. Нека започнем със следващия.

#### Имплементиране на метода DrawCurrentGameState()

Нека създадем метода DrawCurrentGameState(), който ни помага с **рисуването на различните статуси** на **играта** на **конзолата**. **Той** трябва да **приема** следните **параметри**, които ще ни помогнат:

|  |
| --- |
| **(bool inputIsInvalid, int incorrectGuess, string guessedWord, List<char> playerUsedLetters)** |

Методът трябва да иглежда така:

Text

Description automatically generated

In the **method**, we should clear the **console** first because we want to **start** the **drawing** of the **current** **stage** from the **beginning** of the **console**. On the next line write on the **console** the **array** "wrongGuessesFrames" and as an **index** of the **array** pass the incorrectGuessCount variable:

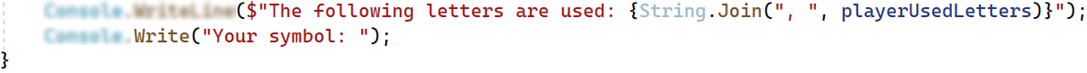
//Как се превежда??



На следващите **два реда** напишете на **конзолата** **думата за отгатване** и колко **символа** потребителя трябва да **отгатне**. Кода трябва да изглежда така:



След това напишете на конзолата **символите**, **които потребителят е въвел**.



Сега трябва да **проверим** дали **входа на потребителя е валиден**. Ако входа е **невалиден**, означава че потребителят е **въвел** **повече** **или** **по-малко от една буква**, затова трябва да **изпишем съобщение** на **конзолата**, че потребителят трябва да **въведе един символ**.

Logo, company name

Description automatically generated

Накрая, трябва да **отпечатате** на **конзолата** къде **потребителят** трябва да въведе **символ**.



Това е всичко, от което се нуждаем, за да работи нашият **метод**. Нека **разкоментираме** **метода**, за да видим дали **работи правилно**. **Прочетете** само един **ред** **от конзолата**, **защото** **методът** няма да спре да **итерира** и не можем да видим дали **работи правилно**.Graphical user interface, text, application, email

Description automatically generated

Накрая трябва да имате следния **резултат** на **конзолата**:

A screen shot of a computer

Description automatically generated

Graphical user interface, text

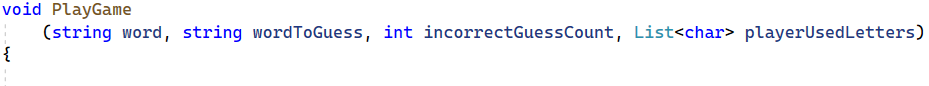
Description automatically generated

Нашият **метод** е **готов**. Нека да продължим към следващия.

#### Имплементиране на метода PlayGame()

Създайте **метода** PlayGame(). Този **метод** трябва да **изпълнява** **основната** логика на нашата **игра**. **Методът** трябва да **прочете** **входа** на играча, **да провери** дали **символът** **съществува** в случайната **дума** и да се появят **символите в думата**, ако символа **съществува**.

**Методът** ще приема **случайната дума**, **думата за отгатване**, **броя на грешните опити** на играча и **списъка на буквите, които играчът е използвал**.



В метода създайте **while-цикъл**, който ще **итерира** докато **играча** е или не е **познал** **думата**.

Text

Description automatically generated with medium confidence

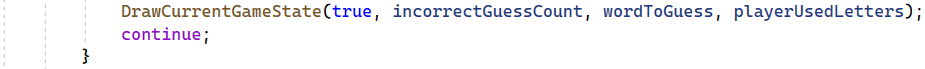
В **цикъла** **прочетете** **входа на играча** и го направете с **малки букви**. Това ще ви **предпази** от **главните букви**. След това проверете дали **входа** има **различна дължина** от **1**.

Картина, която съдържа текст

Описанието е генерирано автоматично

Ако **дължината** е **различна** **от 1**, означава че **играча е въвел повече** или **по-малко от 1** **символ**, затова трябва да **нарисувате** на **конзолата** **статуса** на **играта**, използвайки нашия **метод** и да **продължим** със **следващата итерация**.

Подайте **true** като **първи параметър** на **метода DrawCurrentGameState()**. Ако се подаде **true**, **методът** ще **засече**, че **играчът** е **въвел неправилни символи** и **ще изведе съобщение** на **конзолата**.



Ако проверката мине, конвертирайте входа на играча в **char** и **добавете буквата в списъка**.



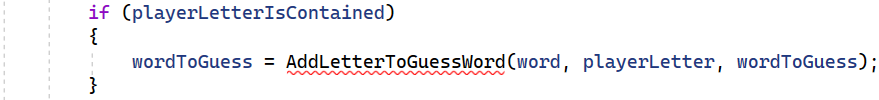
Следващата **проверка** е да **проверим** дали **символът на играча** съществува в **думата**. За тази цел създайте булев **метод** CheckIfSymbolIsContained() и **присвоете** резултата на **булевата** **променлива**. **Метода** ще **приема** **думата** и **символа на играча**.



\*Забележка: Отново ще имплементираме метода в края.

Ако **буквата съществува**, трябва да създадем още един метод AddLetterToGuessWord(), който ще ни помогне за добавяне на буква в думата за отгатване.

Методът ще приема **думата**, **буквата на играча** и **скритата дума**. Накрая ще **присвоим** резултата на **променливата** wordToGuess.



В другия случай, когато играча не познае буквата, просто трябва да **увеличим** **броя грешни опити** с **едно**. Знаете как да направите това:

A picture containing text

Description automatically generated

След това отново трябва да извикаме **метода** за **отпечатването** на **статусите** на **играта** със съответните **параметри**, но този път **подаваме** **false** като **първи** параметър. С подаването на false методът ще знае, че **играчът** е подал **невалидни данни.**



Остава да проверим да ли играча е **загубил** или **победил**. Нека започнем със случая, когато **играча** **победи**. Ще създадем метод CheckIfPlayerWins(), който ще ни помогне да **разберем** дали **играча** е **победил** в **играта**. **Методът** ще **получава** **думата** за **отгатване** и ще върне **true**, ако играча е **победил**. След това трябва да присвоим **отговора** на **булева** **променлива**.



Ако играчът е **победил**, **изчистете конзолата** и напишете на **конзолата** **константата за победа** и **съобщение**, което съдържа **познатата дума**. Накрая **излезте** от **цикъла**.

Text

Description automatically generated with low confidence

Сега направете същото със случая, когато **играчът загуби**. Създайте метода CheckIfPlayerLose(), който ще ни **помогне** да разберем дали **играчът** е **загубил в игра**. **Методът** ще **получава броя грешни опити** и ще **връща** **true**, ако играчът е **загубил**. След това присвоете резултата в **променлива**.



If the **player** **loses,** first set the **cursor** **position** to (0, 0), we do this to **escape** the **console** **blinking** while we **render** the **death** **animation**.

Ако **играчът** **загуби**, сложете **курсора** на **позиция (0, 0)**. Причината затова е да **избегнем** **мигането** на **конзолата**, докато **рендираме** **анимацията на смърт**. **// console blinking**

Text

Description automatically generated

След това ще **създадем** метод с име DrawDeathAnimation(), който ще **реднира масива** **анимацията** за **смърт**. Ще приема масива като параметър.



След като е приключил **метода**, трябва да **изчистим** **конзолата** и да **отпечатаме константата** за **загуба** и **съобщение**, което ще съдържа **познатата дума.** Накрая трябва да **излезем** от **цикъла**.

Graphical user interface, text, application

Description automatically generated

Това е всичко, от което се нуждаем за нашия **метод** да **работи**. Кода трябва да изглежда така:

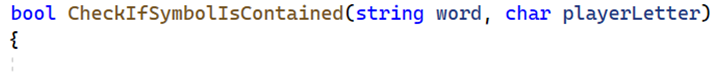
Graphical user interface, text, application, email

Description automatically generated

Сега трябва само да **инициализираме** **методите**, които са му необходими. Имайте предвид, че трябва да **коментирате горния код**, за да тествате реализираните методи един по един.

#### Имплементиране на метода CheckIfSymbolIsContained()

Създайте **метод** с име CheckIfSymbolIsContained(). Той ще получава като параметри **думата** **и буквата, въведена от играча**. Проверете дали буквата, която играчът е въвел, съществува в думата и върнете **true** или **false** .



В **метода** **проверете** дали **буквата** **не съществува** в **думата**. Използвайте **метода** Contains() затова. Ако проверката **мине**, **върнете** **false**. В противен случай, **върнете** **true**.

Graphical user interface, text, application

Description automatically generated with medium confidence

Тествайте метода сами:

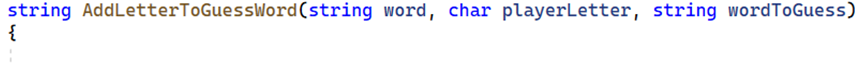




Това беше логиката на **метода**. Нека продължим към следващия.

#### Имплементиране на метода AddLetterToGuessWord()

Създайте **метод** с **име** AddLetterToGuessWord(). Той трябва да получава като параметри **думата**, **буквата,** въведена от играча, и **думата**, която играча трябва да познае. Методът ще **заменя** **буквата** на **индекса**, в който се **съдържа** и ще **връща** новосъздадената **дума**.



Първо създайте **масив** от **символи** и му **присвоете** **скритата дума**. Използвайте **метода** String.ToCharArray(), за да **конвертирате** **думата** в **масив от символи**.



Сега трябва да минем през **всички букви** на **думата** чрез **for**-**цикъл**, за да проверим къде се съдържа буквата.

A picture containing text

Description automatically generated

Когато **буквата**, **въведена от потребителя**, е еднаква с **буквата** на **думата** на **сегашния индекс**, присвояваме буквата, въведена от играча, в **променливата** "wordToGuessCharArr". Кодът трябва да изглежда така:

Text

Description automatically generated with low confidence

Накрая връщаме **нов низ** от **масива** със **символи** "wordToGuessCharArr":



Тествайте метода сами:

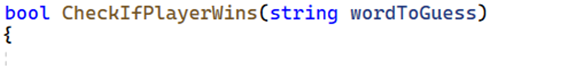




Това е всичко, от което се нуждаем, за да работи **метода**.

#### Имплементиране на метода CheckIfPlayerWins()

Създайте **метод** с **име** CheckIfPlayerWins(). Той ще получава думата, която играча трябва да познае като параметър. Логиката на кода ще бъде дали играчът е познал думата. Знаем че играча е победил играта, когато думата не съдържа долни тирета.



В **метода** трябва да **проверим** дали **думата** **съдържа** **долни** **тирета**.

Ако думата **все** още **съдържа** **долни тирета**, **играчът** все още **не е познал думата** и трябва да **върнем** **false**.

Text

Description automatically generated with low confidence

В противен случай, **върнете true**.



Тествайте метода сами:

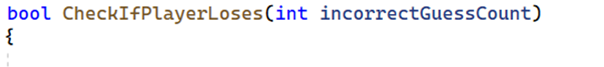




Това беше логиката на **метода**. Нека продължим към следващия.

#### Имплементирайте метода CheckIfPlayerLoses()

Създайте метод с **име** CheckIfPlayerLoses(). Той трябва да **получава** **число**, което представлява **броя на грешните предположения на играча** като параметри. Методът ще ни помогне да проверим дали **играча е изгубил играта**.



Първо трябва да **създадем** **константа**, която е **равна** на **максималния брой грешни опити**, които са равни на дължината на масива wrongGuessesFrames.



Забележете, че **константата** трябва да бъде **създадена в правилното място** в кода. Не в метода.

След като сте създали **константата** се върнете отново в **метода**. В него **проверете** дали **броя грешни опити на играча** е **равен** на **максималния брой грешни опити**. Ако е така, **върнете true**. В противен случай **върнете false**.

Text

Description automatically generated with medium confidence

Тествайте метода сами:

A close-up of a logo

Description automatically generated

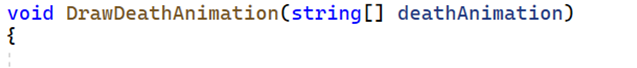
A close up of a box

Description automatically generated

След като свършим с този **метод** може да **продължим** към **последната част** от **нашето** **приложение**.

#### Имплементиране на метода DrawDeathAnimation()

Създайте метод с име "DrawDeathAnimation()". Той трябва да рендира **масива за анимацията на смъртта на играча**. **Методът** трябва да получава **масива за анимацията за смърт** като **параметър**. Кода трябва да изглежда така:



Първо трябва да създадем **for-цикъл**, чрез който ще **итерираме** до **дължината** на **масива за анимацията на смъртта**.

Text

Description automatically generated

В **for-цикъла** отпечатайте на **конзолата масива с** **анимацията за смърт** със сегашния индекс. Знаете как да направите това:



Сега трябва да използваме **метода** "Thread.Sleep()" от **namespace-а** "System.Threading". Този метод ще ни помогне със спирането на **for-цикъла** за **определен период**. В нашия случай **200 милисекунди**. Така ще **визуализираме бавно анимациите** от **масива**. В противен случай, **нашите анимации** ще бъдат **рендирани** **много бързо** и няма да **ги видим**. Трябва да изглежда така:



Накрая ще **сложим курсора** на **позиция** **(0, 0)**. Това ще ни помогне с **нарисуването** на анимацията от **началато на конзолата** и ще избегнем мигането на конзолата. ?? **console blinking**

Text

Description automatically generated with medium confidence

Сега ако всичко е **добре,** не трябва да **получаваме грешки**. Може да се върнем отново в **кода на играта** и да **премахнем коментара** в **метода** на PlayGame(), за да видим дали **работи правилно**.

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated Text

Description automatically generated

Може да видим, че ако сме **загубили играта**, ще се появи **анимацията на смъртта** и играта ще се **рестартира**. И ако **победим**, **играта** ще се рестартира отново. Това е така **защото** **не сме разкоментирали** още един код. Затова се върнете в инициализацията на играта и премахнете коментара на **последнта част от кода**.

A computer screen shot of a computer code

Description automatically generated

**След премахване** на **кода** трябва да може да **победим/загубим** в **края** на **играта**.

Text

Description automatically generated Text

Description automatically generated

**Всичко** е готово за **нашата игра**. Сега да **качим** проекта в **GitHub**.

## Качване на проекта в Github

Вече знаете как да клонирате repository-то с **Git** **Bash** или **TortoiseGit**.

### Използване на TortoiseGit (Опция 1)

Използвайте **Git** **clone**, за да клонирате с TortoiseGit. Отидете на желаната директория, натиснете **десен бутон** на празното пространство в папката и натиснете [Git Clone**]**.

Graphical user interface, application

Description automatically generated

Сега може да отидете на ново създаденото **repository-и** Air да копирате **URL-ла** на **repository-то** – трябва да знаете как да направите това. Последната стъпка е да отворите TortoiseGit, да поставите **URL-ла** и да натиснете [OK].

Graphical user interface, application, email

Description automatically generated

Вашите файлове от **GitHub** ще бъдат свалени в **подпапка** с името на вашия проект в GitHub. Примерно в нашия случай името е "**HangmanByPeter**".

Graphical user interface, application

Description automatically generated

Следващата стъпка е да **добавите** **проекта** в **клонираното repository-и**. Може да преместите **вашия код** от **старата папка на проекта** в **новата папка за** **repository-то**. Може да го направите това с "**Cut & Paste**". Папката трябва да изглежда така.

Graphical user interface, text, application

Description automatically generated

Сега **качете** промените от папката на клонираното **repository-и** в **GitHub**.

Може да използвате [GitCommit…] на TortoiseGit. Отидете на папката на проекта, натиснете **десен бутон** на празното място в папката и натиснете [Git Commit -> "main"…].

Graphical user interface, application

Description automatically generated

Добавете **подходящо** съобщение и натиснете [Add], за да не пропуснете някой липсващ файл и накрая натиснете [Commit].

Graphical user interface, application

Description automatically generated

След това натиснете [Push] и [OK]:

Graphical user interface, text, application

Description automatically generated 🡪

Graphical user interface, application

Description automatically generated

Това е всичко, от което се нуждаете **да качите вашия код** в вашето **repository-и** в **GitHub**, използвайки TortoiseGit.

### Използвайте Git Bash (Опция 2)

**Алтернативно** може да използвате **Git Bash**, за да **къмитнете** и **качите** вашите локални промени в **repository-то**.

Отидете на желаната **директория**, натиснете десен бутон на празното пространство в папката и изберете **[Git Bash Here]**, за да отворите командната конзолата на Git. Ако нямате опция **[Git Bash Here]** в менюто, трябва да инсталирате **Git**.

Graphical user interface, application

Description automatically generated

Напишете командата **"**gitclone**"**, следвана от линк за вашето **repository-и**:

|  |
| --- |
| git clone {URL към Repository-то} |

Тази команда е за клониране чрез **Git Bash**.

Text

Description automatically generated

Файловете от вашето **GitHub** repository-и ще бъдат свалени в **подпапка** с името на вашия **GitHub** проект. В нашия случай името е "**HangmanByPeter**".

Graphical user interface, application

Description automatically generated

Следващото нещо е да **качите** **файловете на проекта** във **вашата папката за клонираното** **repository-и**. Папката трябва да изглежда така:

Graphical user interface, text, application

Description automatically generated

Вече сме готови да качим нашите промени от **Git Bash clone**. Отидете в желаната папка, натиснете десен бутон на празното пространство в **папката**, изберете "GitBashHere" и изпълнете следните **команди**.

Напишете следните команди:

|  |
| --- |
| git status |

Командата **git status** ще отпечата статуса на работещата директория и **индексната област**.

A screenshot of a computer program

Description automatically generated

Сега напишете:

|  |
| --- |
| git add . |

Командата ще **добави** всички модифицирани файлове.

След това:

|  |
| --- |
| git commit -m "Вашето съобщение" |

Тази команда **къмитва** вашите промени. Също така трябва да **добавим** и подходящо **съобщение**.

Second to the last type.??

|  |
| --- |
| git pull |

Командата ще свали всички промени на вашето локално **repository-и**.

Сега остана да **добавим** нашите промени чрез командата:

|  |
| --- |
| git push |

С тази команда промените се прехвърлят в нашето локалното **repository-и**.

A screenshot of a computer

Description automatically generated with medium confidence

Това е всичко, от което се нуждаете, за да качите **промените** на вашето **repository-и** с Git Bash.

## \* Модифициране на кода. Писане на ваши допълнителни функции

Време е да си **поиграете с кода** и да го **модифицирате**.

|  |  |
| --- | --- |
| Icon  Description automatically generated | Това е вашия проект. Бъдете уникални. Не преписвайте!   * Имплементирайте сами **вашите функции**. * **Напишете кода си сами** като използвате вашия стил на код, формат, коментари и т.н. * Направете вашия проект по-интересен. Учете се като си играете с кода и добавяте ваши промени. |

По-долу са представени няколко **идеи**, които може да имплементирате или да модифицирате като допълнения към вашия код.

### Познайте цялата дума

Помислете за **начин**, с който играча да **познае цялата дума**.

### Добавете анимация за победа чрез редниране

Може да **добавете визуализация** за анимация при **победа** в играта.

### Помощ за играча

Имплементирайте функция, която да **помага на играча**, ако иска. Например да **разкриете буква** от думата за играча.

### Създайте GUI приложение

Може да създадете **GUI приложение** с **WinForms** за вашата игра.

### Допълнителни идеи

* Може да направите по-интересен начин за **рендирането на анимации в играта**.
* Може да добавите каквото искате във вашия код, базирано на ваша идея.

### Къмитване в GitHub

Сега може да **къмитнете** и да **добавите** **вашите** **промени** във вашето GitHub repository-и!

|  |  |
| --- | --- |
| Icon  Description automatically generated | Много е важно да **къмитвате вашия код често** в GitHub. Това е начин да имате **богата история на къмитване** за вашите проекти и да принесете растеж във вашата статистика в GitHub:  A picture containing chart  Description automatically generated |

## Писане на README.md файл

Препоръчително е да осигурите **документация за вашия проект** **в GitHub**. Целта е опишете какво прави вашия проект. Затова създайте такъв файл за този **проект**. Нека започнем с редактиране на файла README.md в нашето repo в GitHub:

Graphical user interface, text, application, email

Description automatically generated

### Секция за документация

Добавете информация за проекта във вашия README.md файл: цел на проекта, използване на технология за направата, снимки, демо и т.н.

Обикновено трябва да имате следните **секции**:

* **Име на проекта** (трябва да отговаря на въпроса "Какво има в проекта")
* **Цел на проекта** (какъв проблем ще решим с него. Примерно ще имплементираме определена игра)
* **Решение** (трябва да добавим описание как сме решили проблема 🡪 алгоритми, технологии, библиотеки, framework-ове, инструменти и т. н.)
* **Линк към кода** (трябва да дадете директен линк към вашия код)
* **Снимки** (добавете снимки на вашия проект в различни ситуации по време на използване)
* **Демо** (добавете демонстрация на вашия код на живо с едно кликване)

### Използвайте Markdown

Забележете че файла README.md в GitHub е написан с **езика** **Markdown**. Markdown комбинира текста и специални форматиращи тагове, за да опише форматирани текстови документи.

### Цел на проекта

Започнете с описанието на вашата документация с **целите на проекта**. Какъв проблем решава вашия проект?

### Примерна документация

Това е **пример** как може да изглежда вашата документация. **Не преписвайте**!

Graphical user interface, text, application, email

Description automatically generated

|  |  |
| --- | --- |
| Icon  Description automatically generated | **Напишете документацията сами.** Не преписвайте!  Това е вашия **уникален GitHub профил** с вашия уникален проект. **Бъдете различни** от другите. |

Намерете подходяща снимка и я добавете. Може да добавите снимка с кода:



Може да добавите информация за **входа** и **изхода** на проекта:

Graphical user interface

Description automatically generated with medium confidence

### Вашето решение

Опишете по-долу как сте решили проблема: алгоритми, технологии, библиотеки, framework-ове, инструменти и т.н.

### Добавете линк към кода

Добавете **линк към кода както** по-долу:



### Снимки

Добавете **снимки** на вашия проект:

1. **Направете скрийншот** с вашия любим инструмент(примерно [Snipping Tool](https://support.microsoft.com/en-us/windows/open-snipping-tool-and-take-a-screenshot-a35ac9ff-4a58-24c9-3253-f12bac9f9d44) в Windows).
2. **Поставете** скрийншота в редактора на GitHub Markdown, използвайки [Ctrl+V]:

Примерна снимка на играта " Hangman":

Graphical user interface

Description automatically generated

## Качване на прокта в Replit

Нека добавим нашия **проект** в Replit, за да може да го споделим с нашите **приятели** и да го добавим в нашия **GitHub** профил. Вече знаете как да направите това.

Имаме **две** **опции** за **избиране**. Първата е да използваме **Mono C#**. Технологията все още не актуализирана и работи само за **.NET 5.0**. Това **означава** ще трябва да **направим** нашия проект на **.NET 5.0**, за да може да **работим** с **Mono C#**. Втория начин е да създадем **replit** **проект** на **C#**. Проекта е на **.NET 6.0** и може лесно да **копираме** **нашите** **файлове** в него. Единствената разлика е **Mono C#**, че работи **малко по-бързо** от **Replit проекта** на C#.

### Опция 1 – Mono C#

Не можете да **копирате файловете** във вашия код в **Mono C#** **проект**, защото нашия проект е с **версия** .**NET 6.0**, затова можете да **създадете** **локален** проект на **C#** с версия **.NET 5.0** в **VS** и след това да **копирате** файлове от **стария проект** в **новия**. Ще излязат **няколко грешки**. Трябва да можете да се **оправите с това**. Ако сте се **справели**, може да копирате **кода** с .NET 5.0 в **Mono C# проекта**.

### Опция 2 – C#

Вашата втора опция е да създадете **C# replit** **проект**. С този **начин** трябва **само** да **копирате файловете** от вашия проект в **Replit** **проекта**. Имайте в **предвид**,че проектите, които са **C# replit** са по-бавни от **Mono C#** проектите.

Която и да е опция, трябва да направите **само една промяна**, за да работи **правилно кода**. Намерете променливата string[]words и **променете** **директорията**, която даваме на метода "File.ReadAllLines()". Това е достатъчно на **Replit** да намери файла. Кода трябва да изглежда така:

A black text on a white background

Description automatically generated

Изберете опция и създайте вашия **Replit** **проект**, за да го споделите с вашите приятели.

## Добавяне на Replit линк в файла README.md

Сега може да добавите **демонстрация на проекта на живо** във вашата документация на проекта в **GitHub**. Направете го така както е показано по-долу:

Graphical user interface

Description automatically generated with low confidence

Може да направите **скрийншот** от Replit.com и да го поставите в редактора на GitHub документацията с **[Ctrl+V]**.

Така трябва да изглежда след промяната във вашата документация в README.md файла:

Graphical user interface, application

Description automatically generated

Вече приключихме нашата конзолна игра и имате нов проект във вашето GitHub портфолио.