# Практически проект: Игра на война

Днес ще направим конзолната игра "Game of War":

A picture containing text, indoor, different

Description automatically generated

[Игра на война](https://en.wikipedia.org/wiki/War_(card_game)) е проста **игра с карти**, която обикновено се играе от **двама играчи**, използващи стандартно тесте **карти за игра**. Има много вариации. Това са правилата, които ще следваме при изпълнението:

* На всеки от **двамата играчи** се раздава **една половина** от **разбъркано тесте карти.**
* Всеки ход **всеки играч тегли по една карта** от тестето си.
* Играчът, **изтеглил картата с по-висока стойност**, получава и **двете карти**.
  + **И двете** (**или повече**) карти се връщат в **тестето** на **победителя**.
* Ако има **равенство**, двамата играчи поставят **следващите три карти с лицето надолу** и след това **още една карта с лицето нагоре**. Притежателят на **по-високата карта с лицето нагоре** **получава всички карти** на масата.
* Играчът, който **събере всички карти**, **печели**.

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

Graphical user interface

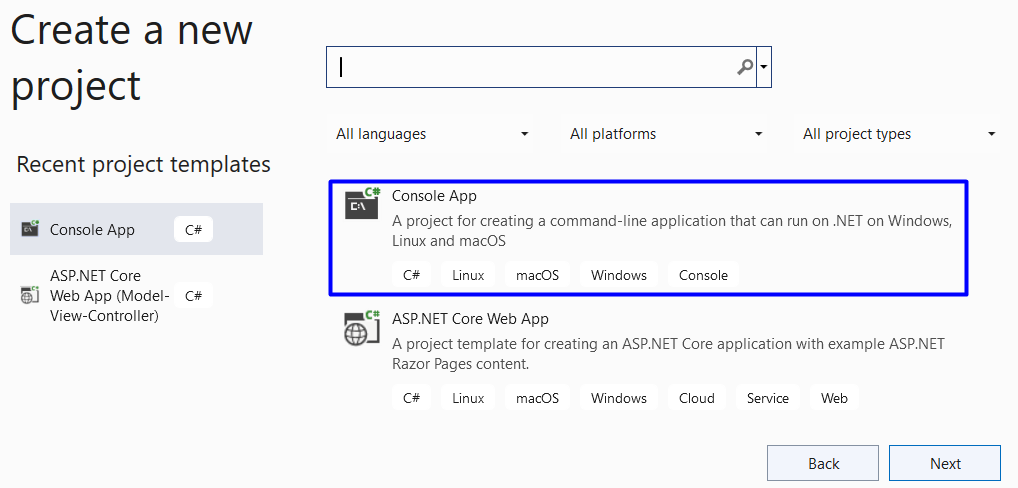
Description automatically generated

## 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**:



След това, **изберете подходящо име** и **място, където да съхраните проекта**. Трябва също да **проверите** [Place solution and project in the same directory] полето, за да нямаме допълнителна папка за нашите файлове. След това натиснете [Next]. На следващия прозорец изберете **[.NET 6 (Long-term support)]** и натиснете **[Create]**:

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

Graphical user interface, text, application

Description automatically generated

Преди да продължим, нека променим името на нашия **main клас** – Program.cs към нещо **по-смислено**. Направете го така:

Graphical user interface, text, application, email

Description automatically generated 🡪 Graphical user interface, text, application

Description automatically generated

## Създайте GitHub репо от Visual Studio

Този път ще създадем **репо** директно от **Visual Studio**. **Visual Studio** улеснява **създаването на репо** директноот **IDE**. Създаване на **репо** от **Visual Studio** е **оптимизирано** за **GitHub**, но можете да **работите отделно** с **Git provider по ваш избор**. Here's how.

|  |  |
| --- | --- |
| 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. |

Имаме създаден **проект**, така че отидете **горе вляво** и натиснете [Git] в **менюто,** след което натиснете[Create Git Repository].

Graphical user interface, application, Teams

Description automatically generated 🡪

Graphical user interface, text, application, email

Description automatically generated

Първо изберете **локален път**, където ще бъде **запазено** **репото на проекта**. Добавете **.gitignore** **за Visual Studio**, **променете лиценза** на "MIT" и **добавете README.md** файл.

Graphical user interface, application

Description automatically generated

Следващото нещо е да **влезете в акаунта си** в **Github** от **Visual Studio**. Натиснете [Sign in…] и [GitHub account].

Graphical user interface, text, application

Description automatically generated

Ще бъдете **пренасочени** към вашия **уеб браузър**, за да **разрешите** вашия **GitHub акаунт**. Кликнете върху [Authorize github].

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

Description automatically generated

След това **въведете паролата си** и натиснете [Confirm password].

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

Description automatically generated

Ако всичко е **минало правилно**, когато се **върнем** към нашия **Visual Studio** **проект**, трябва да сме **влезли** в нашия **GitHub акаунт**.

Graphical user interface, application

Description automatically generated

Остава да изберем **смислено име** за нашето **репо**, напр. "**GameOfWarByPeter**", добавете **кратко описание** и **премахнете отметката** от **квадратчето** (**Private repository**), за да направите вашето репо **публично**. Накрая натиснете [Create and Push].

Graphical user interface, text, application

Description automatically generated

Сега вашето **репо** е **създадено**. Отидете във вашия **GitHub** **профил** и вижте как изглежда. Трябва да е нещо подобно:

Graphical user interface, text, application, email

Description automatically generated

Чрез **създаването** на **репото** от **Visual** **Studio** **получаваме** файла "**.gitattributes**". Файлът "**.gitattributes**" е **прост** **текстов** **файл**, който дава **атрибути** на **имената** **на** **пътищата**. Тоест **модел**, последван от **списък** **с** **атрибути**, разделени с **интервали**. **Празните** **интервали в началото** и **в** **края** **се игнорират**.

Повече за това ще научите тук: <https://git-scm.com/docs/gitattributes>.

В **папката**, в която е **създаден проектът**, трябва да се **появи** **зелен кръг** с **отметка**. Освен това **файловете в** папката **трябва** да са в **зелен кръг с отметка**. Това **показва**, че **файловете на вашия проект** са **commit-нати** и **psuh-нати** към **GitHub**.

Graphical user interface, application

Description automatically generated 🡪 Graphical user interface, application

Description automatically generated

Ако направите **промени** в някой от **файловете**, върху **модифицираните**, ще се **появи** **червен** **кръг** с **удивителен знак**. Това **означава**, че сте направили промени, които трябва да обмислите дали да **commit-нете** и **push-нете** към GitHub.

Graphical user interface, text, application

Description automatically generated

Сега нека започнем да работим върху нашия код.

### Стъпки за имплементация на играта

Ще започнем с **имплементиране** на **клас Card**, който ще **представлява** **карта**. Вътре ще имаме:

* **Две пропъртита**. **Едно** за **лицето** на **картата** и **едно** за **боята** на **картата**.
* **Пренаписан** ToString() **метод**.

След **класа** **Card** ще **имплементираме две енумерации**.

* **Първата** ще **държи** **лицето** **на** **картата** и **номера**.
* **Втората** ще **съдържа** **боите** **на** **картите**, **представени** с **Unicode** **символи**.

Следва **визуализиране** на **менюто** **на** **играта** на **конзолата**.

* Ние ще ви предоставим **текста за менюто**.

След това ще **инициализираме** **играта** и ще **помислим** за **методите**, които **ще са** ни **необходими** за **приложението**.

Накрая ще **имплементираме** следните **методи**:

1. List<Card> GenerateDeck() - **методът** ще ни помогне да **генерираме тесте** от (52) **карти**.
2. void ShuffleDeck(List<Card> deck) - **методът** ще ни помогне да **разбъркаме** **тестето**.
3. void DealCardsToPlayers() - **методът** ще ни помогне да **раздадем** **картите** на **играчите**.
4. bool GameHasWinner() - **методът** ще ни помогне да **проверим** дали **играта** има **победител**.
5. void DrawPlayersCards() - **методът** ще ни помогне да **изтеглим** **карта** от **тестето** **на** **играчите** и да **покажем** **картите** на **конзолата**.
6. void ProcessWar(Queue<Card> pool) - **методът** ще ни помогне да **обработим** **войната**, ако **играчите** **имат** **карти** с **равни** **стойности**. В този **метод** ще имаме още **два** **метода**. Те ще ни помогнат да:
   1. void AddCardsToWinnerDeck(Queue<Card> loserDeck, Queue<Card> winnerDeck) **- добавим карти** към **тестето на победителя.**
   2. void AddWarCardsToPool(Queue<Card> pool) **- добавим карти към набора** **от карти**.
7. void DetermineRoundWinner(Queue<Card> pool) - **методът** ще ни **помогне** да **определим** **победителя** в **рунда**.

Накрая ще **комбинираме** **всичко**, за да може нашата **игра** да **работи** **правилно**.

## Дефиниране на класа и енумерациите

### Типове енумерации

**Енумерацията** е **конструкция**, която **наподобява** **клас**, но с тази **разлика**, че в **тялото** на **енумерацията** можем да **декларираме** **само** **константи**. **Типове** енумерации могат да приемат **стойности** само **измежду** **константите**, изброени в **типа**.

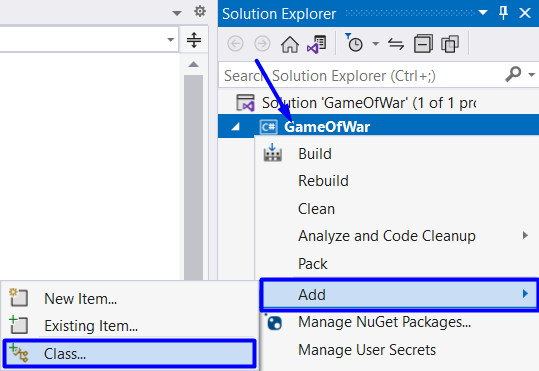
**Енумерацията** правят **кода** **по-ясен** и **по-четим**, **особено** когато се използват **смислени имена**. **Ползите** от **използването** на **енумерациите** включват **намаляване на грешките**, причинени от **транспониране** или **грешно** **въвеждане на числа**. Те също така **улесняват** **промяната** **на** **стойностите** в **бъдеще**.

Можете да научите повече тук: <w3schools.com/cs/cs_enums.php>.

#### CardFace енумерация

Ще започнем с **дефиниране на енумерация**, която ще представлява **лицата на картите и техния номер**.

За да добавите енумерация, отидете на [SolutionExplorer], **натиснете** върху **проекта** и след това **[**Add**]** 🡪 [Class...]:



Натиснете [Class] и изберете **смислено** име, напр. **"**CardFace**"** и го създайте**.**

Graphical user interface, application

Description automatically generated

След като го създадете, трябва да имаме **нормален клас**:

Graphical user interface, text, application

Description automatically generated

За да го направим от тип **enum**, просто трябва да **променим** ключовата дума "**internal**" на "**public**" и ключовата дума "**class**" на "**enum**". Накрая трябва да изглежда така:

Ключовите думи "**internal** " и "**public**"са **модификатори за достъп**. **Модификаторите за достъп** определят **достъпността** на **обект** и всички негови **членове** в C**#** **проекта**. Можете да научите повече тук: <https://code-maze.com/csharp-access-modifiers/>.

В нашия случай ще направим нашата **енумерация** **публична**, така че да е **достъпна** от други класове:

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

Description automatically generated

Сега нека **създадем** **лицата** **на** **всички** **карти**. За да направим това, трябва да знаем, че **едно** **тесте** **карти** **има** **13 различни карти**, които **започват** от **две** (2) до **асо** (A). Трябва да **добавим** **всички** **карти** със **съответния им номер в нашата енумерация**. Това ще ни трябва **по-късно**, за да получим **номера** на карта **според** нейното **име**. Трябва да изглежда така:

A picture containing graphical user interface

Description automatically generated

По подразбиране **свързаните** **постоянни** **стойности** на **пропъртитата** **на** **енумерацията** са от **тип** **int**. Те **започват** **от** **нула и се увеличават с единица**, следвайки реда на **текста на дефиницията**. **Можете** **изрично** да посочите всяко друго **число**. В нашия случай ще **започнем с 2** и ще **увеличим с едно до 14**, така че да имаме **правилните номера на картите**.

Ако не **специфицираме** **стойностите**, **енумерацията** ще **изглежда** така:

A picture containing text

Description automatically generated

В нашия случай трябва да **специфицираме** **стойностите**, **защото** ще ни трябват **по-късно**, когато ги **показваме** на **конзолата**. И накрая, нашата **енумерация** трябва да изглежда така:

A picture containing table

Description automatically generated

Това е всичко, от което се нуждаем за тази **енумерация**, така че нека продължим с **втората**.

#### CardSuit енумерация

Създайте втората **енумерация** и я кръстете **CardSuit**. Трябва да знаете как да направите това.

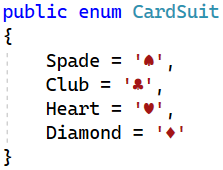
Graphical user interface, text

Description automatically generated with medium confidence

Тази **енумерация** ще съдържа **боите** на **картите**. Всяко **тесте** **карти** има **четири** **бои** (пика, спатия, купа, каро). Ще ги **представим** с техните **Unicode** **символи**. Можете да използвате тези:

|  |
| --- |
| (Пика) ♠, (Спатия) ♣, (Купа) ♥, (Каро) ♦ |

За да създадем enum, ще използваме **името** на **боята** и като негова **стойност** ще **присвоим** **съответния** **символ**. **C#** ни позволява да използваме **символи вместо цели числа** за **стойността** на **енумерациите**. Накрая трябва да изглежда така:



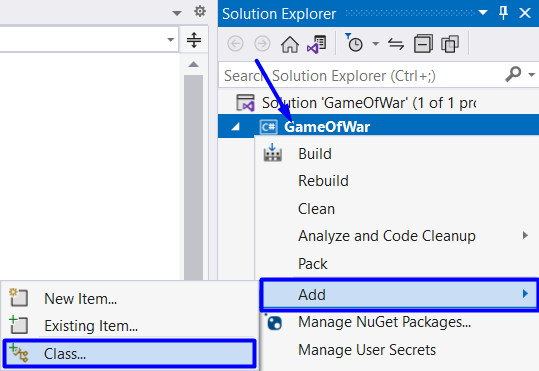
Това е всичко, от което се нуждаем, за да работи **втората ни енумерация**. Нека продължим с **класовете** на нашата игра.

### Клас Card

**Класът** в **ООП** е **описание** (спецификация) на **клас** от **обекти** в **реалността**. **Класът** е **шаблон**, който **описва** **типовете** **състояния** и **поведението** на **конкретни** **обекти** (инстанции), които са **създадени** от този **клас** (шаблон).

Нека **създадем** **клас** за нашата **карта**

За да добавите **клас**, отидете на [SolutionExplorer], **натиснете с** **десния** **бутон** върху **проекта** и след това **[**Add**]** 🡪 [Class...]:



Изберете [Class] и изберете **смислено** име, напр. **"Card"** и го създайте.

**Graphical user interface, application

Description automatically generated**

След като го създадете, трябва да изглежда така:

Graphical user interface, text, application

Description automatically generated

Преди да **продължим** с **дефиницията**, трябва да **променим** **ключовата** **дума** internal на public. Трябва да изглежда така:

Graphical user interface, text, application

Description automatically generated 🡪 Text

Description automatically generated

Сега първо, в **класа**, трябва да **създадем** **две** **пропъртита**. **Пропъртито** в **C#** е **член** на **клас**. **Вътрешно** C# **свойствата** са специални **методи**, наречени **инструменти за достъп**. Повече информация тук: <https://www.c-sharpcorner.com/article/understanding-properties-in-C-Sharp/>.

Когато **създаваме** **клас** или **енумерация**, ние **създаваме** **дефиниран** **от потребителя тип данни**. Това **означава**, че можем да **създаваме** **променливи** или **свойства** на този **дефиниран от потребителя тип данни**.

Първото **свойство** ще бъде от **тип** "CardFace":

Text

Description automatically generated with medium confidence

The second one will be of type "CardSuit":



**Последното нещо**, от което се нуждаем, за да **работи** този **клас**, е да **пренапишем** метода ToString(). **Пренаписан** **метод** **осигурява** нова **имплементация** на **метода**, **наследен** от **базовия клас**. **Всеки обект** в C# получава **метода** ToString(), който **връща** **текстово** **представяне** на този **обект**.

За да пренапишем **метода** ToString(), трябва да създадем **обикновен** **метод** с име ToString() и **преди** **типа** **данни**, който **връща**, просто добавете **ключовата** **дума** **override**. Трябва да изглежда така:

A picture containing graphical user interface

Description automatically generated

Това, което остава, е да добавим **логика** вътре в **метода**, така че той да **връща** **правилните текстови данни**. Искаме да **върнем** **лицето на картата** и **боята** в ред (2♠, 3♠, 4♠,…, A♠, 2♣,… 2♥,…, 2♦,… A♦). По този начин изглежда **по-добре**, когато го **отпечатате** на **конзолата**. За да направим това първо, трябва да вземем **стойностите** на **лицето на картата** от нашата **енумерация**.

За да получим стойността, ще използваме **метода** Enum.Parse(). Като **аргументи на метода** предайте **типа** **на нашата енумерация** и **лицето** на **текущата карта** чрез достъп до нейното **пропърти** Face. Накрая го **парснете** към int, за да получите **стойността на картата**. Трябва да изглежда така:



**Ключовата дума** (this) в **C#** дава **достъп** до **референцията** към **текущия обект**, когато се използва от **метод** в **клас**. Това е **обектът**, чийто **метод** или **конструктор** се **извиква**.

Следва да вземете **боята** на **картата**. Не трябва да **парсваме** този път, защото имаме **char** като **стойност**. Просто трябва да създадем **променлива** **char**, да й присвоим **пропъртито Suite** и да я **кастнем** към **char**.



**Идеята** тук е да получите **лицето** напр. (2, 3, 4, …, A) и **боята**, например (♠, ♣,♥, ♦) на **текущата карта**, **конкатенирайте** ги и ги **върнете** в **метода** ToString(). Така че **всеки път**, когато **методът** ToString() **бъде** **извикан** от някоя от **картите** в **тестето**, той ще **отпечата** тяхното **лице** и **боя** по **начина**, по който **желаем**.

Сега имаме **лицето** и **боята** на **картата**. Но засега не можем да го **върнем**. Първо, трябва да проверим дали **стойноста** на **лицето** на картата е **по-голяма** от **десет**.

A picture containing text

Description automatically generated

Тази **проверка** е **необходима**, защото след **лицето на картата десет** (10) имаме (Вале, Дама, Поп, Асо) и искаме да **отпечатаме** (J, Q, K, A), а не (11, 12, 13, 14). За да направим това, трябва да получим текста от **лицето на картата** и след това **първия символ** на **низа**.



След това **върнете** **символа** **на лицето** и **боята** **свързани**.



Ако проверката е **пропусната**, просто **върнете** **лицето** и **боята**, **свързани**

A picture containing text

Description automatically generated

Това е всичко, от което се нуждаем за нашия **клас**. Нека продължим с **инициализацията** на **играта**.

## Инициализация на играта

Първо **натиснете** върху файла "GameOfWar.cs" в [SolutionExplorer]. В този файл ще **инициализираме** **играта**.

Graphical user interface, text, application

Description automatically generated

**Преди** да **започнем** първо с **менюто на играта**, трябва да настроим **изходното кодиране** на **конзолата** на **UTF8**. Това е така, **защото** ще използваме **Unicode символи**, за да **визуализираме** **бои на** **карти**. Можете да направите това, както следва:



### Показване на менюто на играта

Нека започнем, като **напишем менюто** на играта на **конзолата**. Можете да **копирате** и **поставите** **предоставения** **текст** от менюто или да опитате да го създадете сами.

|  |
| --- |
| @"  ================================================================================  || Welcome to the Game of War! ||  || ||  || HOW TO PLAY: ||  || + Each of the two players are dealt one half of a shuffled deck of cards. ||  || + Each turn, each player draws one card from their deck. ||  || + The player that drew the card with higher value gets both cards. ||  || + Both cards return to the winner's deck. ||  || + If there is a draw, both players place the next three cards face down ||  || and then another card face-up. The owner of the higher face-up ||  || card gets all the cards on the table. ||  || ||  || HOW TO WIN: ||  || + The player who collects all the cards wins. ||  || ||  || CONTROLS: ||  || + Press [Enter] to draw a new card until we have a winner. ||  || ||  || Have fun! ||  ================================================================================" |

Трябва да изглежда така:

Text

Description automatically generated

### Работен процес на играта

**Последователността** на играта ще бъде **следният**:

* Първо ще **генерираме** **тесте** от **петдесет** **и две** (52) **карти** и ще **разбъркаме** **генерираното тесте**
* Следва да **създадем** **две** **опашки**, които ще **представляват** тестетата на **първия** и **втория** **играч**, и да **раздадем** **картите** на **играчите**
* След това трябва да създадем **цикъл**, който ще **изпълнява логиката на играта**. В **цикъла**:
  + Ще **изтеглим** карти от **тестетата на играчите** и ще ги **покажем** на **конзолата**
  + След това ще **сравним** **картите**, за да видим **кой** от **играчите** има **карта** с **най-висока стойност** и ще **определим** **победителя** в **рунда**
  + Имаме **още един случай**, при който **картите** са с **еднакви** **стойности**. Когато **картите** имат **еднаква стойност**, имаме **война**
    - Когато имаме **война**, ще **изтеглим** **три** **карти** от **тестето** **на** **всеки** **играч** и ще ги добавим към **тестето** от **карти** (тестето от карти представлява масата, на която са поставени картите в реалната игра)
    - След това ще **изтеглим по една карта** от **всеки играч** и ще **сравним** **картите** **отново**, за да видим **кой** е **победителят** в **текущия кръг**. **Победителят** в **рунда** **получава** всички карти от **тестето** и **ги добавя** към **тестето си**.

Ще **започнем** с **основната логика на играта**. Докато преминаваме **през** нея, ще помислим за **методите**, от които ще се нуждаем. След като **приключим** с **основната логика на играта**, ще **коментираме** **кода** и ще го **разкоментираме** **бавно** с **инициализацията** на **методите**.

Първо, трябва да **генерираме тесте карти**. Това е мястото, където нашият **метод** GenerateDeck() ще дойде на мястото си. Създайте **списък** с **карти** и му задайте метода GenerateDeck(). Знаем, че **списъкът** е **тип** **данни**, който **приема** други **типове данни** и **прави колекция** от **тях**. Както **знаем**, нашият **клас** е **тип данни**, **дефиниран от потребителя**. Така че можем да **направим** **колекция** от нашия **клас**. Трябва да изглежда така:



**\*Забележка:** Можете да създадете метода празен сега или да го закоментирате, за да не дава грешка.

Следващото нещо е да **разбъркаме** **тестето карти**, преди да ги **раздадем** на **играчите**. Ще **извикаме** метода ShuffleDeck() и ще му подадем **генерираното тесте** карти.



Сега трябва да създадем две **опашки** от **карти**, които ще **представляват** **тестетата** на **двамата играчи**. Тук използваме **опашки**, **защото** **искаме** да **теглим** **картите** от **горната част на тестето**.



Следва **раздаване** на **картите** на **играчите**. Ще използваме **метода** DealCardsToPlayers(), за да ни помогне с това.



Преди да **започне играта**, трябва да създадем **три допълнителни променливи**, които ще ни помогнат **по-късно** в **кода**. **Две** **променливи** от тип **Card** ще **представляват** **картите**, **изтеглени** от всеки от играчите, а една **променлива** int ще бъде **броячът** на **общия брой ходове**, направени от **играчите**.

Text

Description automatically generated

**Подготвили** сме **всичко** за **играта**. Следва да създадем while **цикъл**. Този **цикъл** ще **изпълнява** **играта**, **докато** има **победител**. Тук ще **създадем** друг **метод** GameHasWinner(), който ще проверява вместо нас **дали** има **победител**. И ако има **победител**, **цикълът** ще **спре** и ще **покаже** на **конзолата** **съобщение**. Трябва да изглежда така:

A picture containing text

Description automatically generated

В **цикъла** първо трябва да **прочетем** **един** ред от **конзолата**. Това ще ни помогне да **спрем** **цикъла**, **докато** не бъде **натиснат** някой **бутон**, за да може да **продължи** със **следващата** **итерация**.



Сега трябва да **изтеглим** **карти** от **тестетата** на **играчите** и да ги **покажем** на **конзолата**. Ще създадем **метод** DrawPlayersCards() за това.



След това трябва да създадем **набор** **от карти**. В този случай **набор** **от карти** наричаме **масата**, на която са **поставени** **картите** в **реална игра**. Отново ще бъде **опашка от карти**. Ще използваме **опашка**, **защото** след като добавим картите към **набора** **от карти**, можем **лесно** да ги **премахнем** от **опашката** и да ги **добавим** към **тестето на победителя** в **същия** **ред**.



След като **набора** **от карти** е **създаден**, трябва да **добавим** **изтеглените карти на играчите** в него. Използвайте **първото** и **второто** **пропърти** на играча, които **създадохме**. Трябва да знаете как да направите това:

A picture containing graphical user interface

Description automatically generated

Виждаме, че **картите** на играчите са **маркирани** в **червено**. Това е така, защото нямаме нова **инстанция** на **променливата** **Card**. Ще създадем тази инстанция в DrawPlayersCards().

Имаме още **няколко** неща за вършене и **основната ни логика е готова**. Следващото нещо е да **сравним** **картите на играчите**. Първо, можем да имаме **война**. Имаме **война**, когато **картите** **на двамата играчи** са с **равни стойности**. Ще създадем **метод** ProcessWar(), който ще ни помогне с **процеса**. **Методът** ще **приеме** **набор от карти**.



След **обработката** на **войната** трябва да **определим** **победителя** в **текущия кръг**. Ще **създадем** **метод** DetermineRoundWinner(), който ще **приеме** **набор** **от карти**. Този **метод** ще **сравни** **отново** **картите** и ще **покаже** **съобщение** за това кой е **победителят** в **текущия кръг**.



И накрая, трябва да **покажем** на **конзолата** колко **карти** има всеки **играч** и да **увеличим** **общата** **променлива** на **ходовете** с **едно**. Опитайте това сами. Ето един пример:

Graphical user interface, text, application

Description automatically generated with medium confidence

Това е всичко, от което се нуждаем за нашата **основна логика**. Вашият код трябва да изглежда така:

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

Description automatically generated

\***Забележка**: Можете да **закоментирате** **кода** и да го **разкоментирате** **бавно** с **инициализирането** на **методите** или да **инициализирате** **методите** **празни**. Зависи от вас.

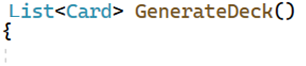
Нека продължим с прилагането на методите.

### Имплементиране на методите

Нека **имплементираме** **методите**, които ще ни трябват за този проект.

#### Имплементиране на GenerateDeck() метода

Нека започнем с **имплементирането** на нашия метод GenerateDeck(), който ще създаде **колекция с нашите карти** (всяка с различно лице и цвят). Методът ще върне **списък** с **карти** (тесте карти). Трябва да изглежда така:



Първото нещо, което трябва да направим в **метода**, е да създадем List<Cards>, който ще **представлява тестето карти**:



Сега за интересната част от този **метод**, трябва да получим **всички имена** на **карти** и **бои** от нашата **енумерация**.

За да направим това, трябва да използваме **метода** Enum.GetValue(), който **приема** **типа** CardFace (името на нашата енумерация). След това трябва да **кастнем** **върнатата** **колекция** от метода към **масив** **от** CardFace и да я присвоим на **променлива**. Направете го така:



Трябва да направим същото с боите на картите.



Сега, когато имаме **картите и боите**, можем да **създадем** **тестето**. За да направим това, имаме нужда от **два** вложени for **цикъла**. **Първият** ще **итерира** през **боите**, а **вторият** ще **итерира** през **лицата**. Трябва да изглежда така:

A picture containing text

Description automatically generated

Следващото нещо е да поставим **лицето** на **картата** в **текущия** **индекс** и **боята** на **картата в текущия индекс**. Трябва да изглежда така:



Сега, когато имаме **текущото** **лице** и **боя** на **картата**, трябва да **създадем** нова инстанция на Card и да я **добавим** към **списъка** с **карти**. Всеки път, когато **декларирате** **обект** от **определен клас**, вие **създавате** **нова** **инстанция** от този **клас**. **Създава** се **нова** **инстанция** с **ключовата дума** new.

Когато създадем **инстанция** на **класа в нашия случай** (class Card), ние имаме **достъп** до **пропъртитата** на **този** **клас**, **в нашия случай** (Face и Suite). Когато имаме **достъп** до **пропъртитата**, можем да им **присвоим** **стойности**. Единственото **изискване** за **стойностите** е те **да са** от един и същи **тип данни** в нашия случай (CardFace и CardSuite). Можете да направите това по следния начин:

A picture containing text

Description automatically generated

Накрая върнете **тестето** с картите в него.



Сега нека **извикаме** и да **дебъгнем** **метода**, за да видим дали **работи правилно**. Можете да направите това сами:

Chart

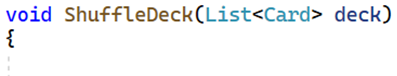
Description automatically generated with medium confidence

Ако всичко работи **правилно**, трябва да има **52** **различни** **карти** в списъка. Това е всичко, от което се нуждаем, за да създадем този **метод**. Нека продължим с метода ShuffleDeck().

#### Имплементиране на ShuffleDeck() метода

В **метода** ShuffleDeck() ще използваме **прост алгоритъм** за **разбъркване на картите** в **тестето**. **Алгоритъмът** ще **вземе** **карта** с **произволен индекс** от **тестето** и ще я **размени** с **друга произволна карта**.

Методът ще приеме **нашето генерирано тесте карти**. Трябва да изглежда така:



Ще ни трябва **променлива от типа Random**, за да **работи** **алгоритъмът**. Знаете как да направите това:



След това трябва да създадем for **цикъл**, който ще **обхожда** нашата колода.

Chart

Description automatically generated

Следващото нещо е да получим **случаен индекс** за **първата карта**, която ще преместим. Използвайте променливата Random, за да направите това:



След това създайте **временна променлива**, която ще **държи картата на произволен индекс**, който ще **разменим с друг**:



Сега, когато имаме **временната карта** и **случаен индекс**, можем да **разменим** **карта** от тестето на **произволния** **индекс** с **карта** **от** **тестето** с индекс "i". Трябва да изглежда така:

Text

Description automatically generated with medium confidence

Това е всичко, от което се нуждаем за **метода** **за разбъркване на тестето ни**. Нека го **извикаме** и **дебъгнем**. Направете това сами.

Text

Description automatically generated Text

Description automatically generated

Ако всичко **работи правилно**, когато дебъгвате **приложението**, **тестето** трябва да има **карти** **в** **различен ред.** Сега нека създадем следващия.

#### Имплементиране на DealCardsToPlayers() метода

Нека създадем метода DealCardsToPlayers(). **Методът** ще вземе **две карти** от **тестето**, докато **няма карти** в **тестето** и ще ги **раздаде** на **играчите**. И накрая, всеки **играч** **трябва да има** **двадесет и шест** (26) **карти**.

Създайте while цикъл, който се **повтаря**, докато все още имаме карти в тестето. Трябва да знаете как да направите това:

Graphical user interface, text

Description automatically generated

Сега трябва да вземем **две карти от тестето** и да ги **раздадем** на **играчите**. За да направим това, можем да използваме метода от масив Take() от LINQ. Този **метод** връща **определен брой последователни елементи** от **началото** на **поредица**. След това преобразувайте взетите **карти** **в** **масив**.



След като **вземем картите**, ние също трябва да ги **премахнем от тестето**, тъй като ще бъдат **дадени** на **играчите**. Можем да използваме **метода** RemoveRange(), за да **премахнем** **първите два елемента**. Трябва да изглежда така:



Накрая **поставете картата** с **индекс 0** в **тестето на първия играч** и **картата** с **индекс 1** в **тестето** **на втория играч**. Трябва да знаете как да направите това:

A picture containing text

Description automatically generated

Това е всичко, от което се нуждаем, за да работи нашият **метод**. Извикайте и дебъгнете метода, за да видите дали работи правилно.





Ако всичко **работи правилно**, всеки играч трябва да има **двадесет и шест** (26) карти в **тестето** си. Това е всичко, от което се нуждаем, за да работи този метод, нека продължим със следващия.

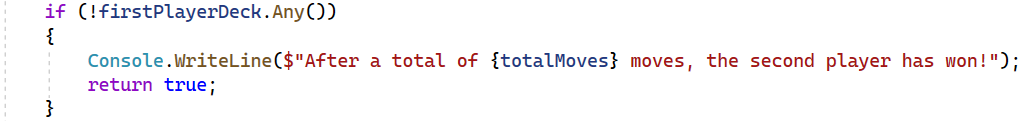
#### Имплементиране на GameHasWinner() метода

Създайте метода GameHasWinner(). Този **метод** ще **провери** дали някой от **играчите** **няма** карти в **тестето**. Ако **играчът** няма **карти** в **тестето**, това означава, че **другият играч** е **спечелил** **играта**. **Методът** ще **върне** true или false в зависимост от това дали **имаме** **победител** или **не**. Ако имаме победител (true), ако не (false).

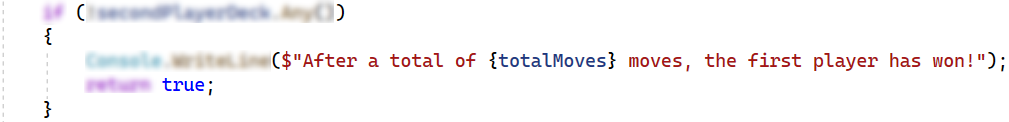
A picture containing text

Description automatically generated

Първо проверете дали първият играч **има** **карти**. За да направите това, можете да използвате **метода** Any(), за да проверите дали има **карти в тестето**. Ако **първият** **играч** **остане** **без** **карти**, това означава, че **вторият** **играч** **печели** **играта**. Покажете **съобщение** за **обявяване** на **победителя** и **върнете** на true.



Направете същото за другия играч.



Ако сме **пропуснали и двете проверки**, върнете false. Това **означава**, че **все още нямаме победител**.



Това е всичко, от което се нуждаем, за да работи този метод. Намерете начин да го **дебъгнете** сами и вижте дали **работи правилно**.





Ако някой от **играчите** няма **карти** в **тестето** си, **методът** трябва да **върне** true (имаме победител), **в противен** **случай** трябва да **върне** false (все още нямаме победител).

Това е всичко, от което се нуждаем, за да работи този метод. Да продължим със следващия.

#### Имплементиране на DrawPlayersCards() метода

Text

Description automatically generated with medium confidenceСъздайте метода DrawPlayersCards(). Този метод ще **изтегли** **карта** от **тестетата** **на играчите** и ще я **покаже** на **конзолата**.

Сега извадете карта от **тестето** на **първия** **играч** и **покажете** **съобщение** на **конзолата**. Трябва да знаете как да направите това:



Направете **същото** с **втория играч**.



Това е всичко, от което се нуждаем за този **метод**. На този **етап** можем да **разкоментираме** **по-голямата** **част** от **кода** и да **тестваме** как **работи** **играта**.

Graphical user interface, text, application

Description automatically generated

Само **методите** ProcessWar() и DetermineRoundWinner() трябва да бъдат **закоментирани**, защото все още не са **инициализирани**. Поради тази причина **все още нямаме победители в кръга**, само **картите** се **показват** на **конзолата**.

Стартирайте приложението за игра и вижте дали работи правилно.

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Можете да видите **показаното меню на играта** и ако натиснете [Enter], **картите** ще бъдат **изтеглени** от **тестетата на играчите** и ще се **покажат** на **конзолата**. Сега нека продължим със следващия метод.

#### Имплементиране на ProcessWar() метода

Създайте **метода** ProcessWar(), който ще **приеме** **набор** **от карти** от **карти**. **Методът** ще вземе **картите на играчите** и ще **сравни** техните **стойности**, за да види дали са **равни**. Ако **стойностите на картите** са **равни**, имаме **война**. Когато **войната се случи**, ще вземем **още три карти** от **тестето на всеки играч** и ще ги **добавим** към **набора** **от карти**. Накрая **теглим** **още една карта** от **всеки играч**, за да **определим** **победителя** в **рунда**. **Победителят** в **кръга** получава **всички** **карти** от **набора** **от карти**.

Започнете, като създадете метода:

Logo

Description automatically generated with medium confidence

Създайте while цикъл, за да **итерирате**, докато **двете карти станат равни**. Ако **все още сме в цикъла**, това означава, че **все още имаме война** (можем да имаме няколко войни една след друга). За да направим това, трябва да **имаме достъп** до пропъртито Face на **картата** и да го **конвертираме** към int (за да получим стойността на енумерацията). След това **напишете на конзолата** "WAR". Трябва да изглежда така

A picture containing text

Description automatically generated

Сега трябва да **проверим** дали има **играч**, чието **тесте има по-малко от четири карти**. Ако един играч **има по-малко от четири карти**, той **губи автоматично** (играчът няма достатъчно карти, за да продължи войната) и **другият играч получава всички карти**.

Тук ще ни трябва друг метод AddCardsToWinnerDeck(), който ще вземе **картите** от **тестето на губещия играч** и ще ги **добави** към **победителя**.

Нашата проверка трябва да изглежда така:

Logo, company name

Description automatically generated

Направете същата проверка за другия играч. Вече знаете как да го направите.



Ако и **двете проверки са успешни**, това означава, че **играчите** имат **повече от четири карти** и трябва да **добавим три карти** от **двамата играчи към набора** **от карти**. Тук ще създадем метод AddWarCardsToPool(), който ще ни помогне с това.



След като **добавим картите към набора** **от карти**, трябва да **решим** кой **играч ще спечели** "**войната**" и ще получи всички карти. За да направим това, трябва да **изтеглим** **карта** от **тестетата** на **двамата** **играчи**, за да можем да ги **сравним**, за да видим **кой от играчите** **печели** **войната** и **рунда**. Също така **показва картите на конзолата** и ги **добавя към набора** **от карти**. Можете да направите това сами.

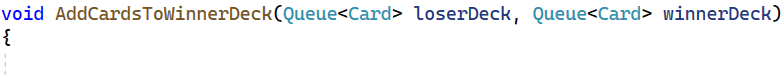
A picture containing text

Description automatically generated

Това е всичко, от което се нуждаем за този метод. Преди да продължим, трябва да създадем двата помощни метода AddCardsToWinnerDeck() и AddWarCardsToPool(), за да бъде нашият метод ProcessWar() завършен и да няма никакви **грешки** в него.

#### Имплементиране на AddCardsToWinnerDeck() метода

Създайте метода AddCardsToWinnerDeck(). **Методът** **приема** тестетата на **губещите** и **печелившите** **играчи**. Ще премине **през** **тестето** **на** губещия, докато няма карти в него и ще ги добави към **тестето на победителя**.



Единственото нещо, което трябва да направим тук, е да **създадем цикъл**, който ще се **повтаря**, докато има **карти** в **губещото тесте** и да **добавя картите** към **печелившото тесте**. Опитайте се да направите това сами.

Graphical user interface, text, chat or text message

Description automatically generated with medium confidence

Това е всичко, от което се нуждаем за този метод. Можете да извикате метода и да го тествате дали работи за вас. Да продължим със следващия.





#### Имплементиране на AddWarCardsToPool() метода

Създайте метода AddWarCardsToPool(). **Методът** ще приеме **набор** от **карти**. Ще вземе **три карти** от **тестетата** на **двамата играчи** и ще **добави** **картите** към **набора** **от карти**.

A picture containing shape

Description automatically generated

Използвайте for **цикъл**, за да добавите **три карти** от **първото** и **второто** **тесте** на **играча** **към** **набора** **от карти**. Можете да направите това сами.

Graphical user interface, text

Description automatically generated

Това е всичко, от което се нуждаем за този метод. Можете да извикате метода и да го тествате дали работи за вас.





Сега **методът** ProcessWar() работи, но трябва да **инициализираме** **последния метод** DetermineRoundWinner(), за да видим дали **всичко** работи **правилно заедно**.

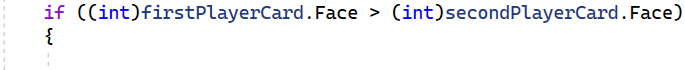
#### Имплементиране на DetermineRoundWinner() метода

Създайте метода DetermineRoundWinner(), който ще **приеме** **набора** от **карти**. **Методът** ще получи **изтеглените карти** **на** **играчите** и ще ги **сравни**, за да види кой **играч** **има** **карта** с **по-висока стойност**. **Играчът** **с карта с по-висока стойност** **печели** **рунда** и **получава** всички **карти** от **набора** **от карти**.

A picture containing text

Description automatically generated

Първото нещо в **метода** е да проверите дали картата на първия играч има по-висока стойност от **картата на втория играч**. За да направим това, трябва да имаме **достъп** до пропъртито Face на **картата** и да го **конвертираме** към int. Трябва да изглежда така.



Ако **първият играч спечели**, напишете на конзолата съобщение, за да го обявите, и **добавете всички карти от набора** **от карти към тестето му**. Трябва да изглежда така.

Text

Description automatically generated

Направете **същото** за **другия играч**.

Graphical user interface, Teams

Description automatically generated with medium confidence

Това е всичко, от което се нуждаем, за да **работи правилно** нашият **последен метод**. Сега нека се върнем към **основната логика**, да **разкоментираме** **последния метод** и да **тестваме играта**, за да видим дали **работи правилно**. Когато **стартирате приложението**, то трябва да изглежда така:

Text, timeline

Description automatically generated

Text

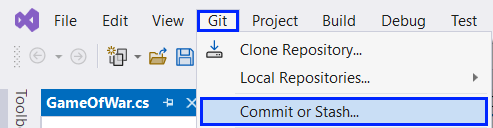
Description automatically generatedText

Description automatically generated

Сега нека качим приложението в GitHub.

## Качете проекта си в Github от Visual Studio

Този път създадохме нашето **GitHub репо** от **Visual Studio**, така че ще **commit-нем** нашия проект от **Visual Studio**. За да направите това, отидете в **горния ляв** ъгъл и натиснете [Git] в **менюто**. След това натиснете [Commit or Stash…].



**Отляво** ще се появи прозорецът [Git Changes]. **Visual Studio** има прозорец [Git Repository] (прозорец "GitChanges**"**), който е **консолидиран изглед** на всички **подробности** във вашето **репо**, включително **локални** и **отдалечени** **разклонения** и **хронология** на **commit-и**.

Graphical user interface, text, application

Description automatically generated

Въведете **подходящо съобщение**.

Graphical user interface, text, application

Description automatically generated

Накрая натиснете **стрелката** до бутона [Commit All] и изберете [Commit All and Push].

Graphical user interface, text, application, email

Description automatically generated

Ще се **появи** **прозорец**, за да ви попита дали искате да **запазите** **следните промени**, натиснете [Save]. Това е всичко, което трябва да **commit-нем** и да **push-нем** нашия проект от **Visual Studio**. Трябва да изглежда нещо подобно във вашия **GitHub**.

Graphical user interface, text, application, email

Description automatically generated

## \* Модификации на кода

Сега е време да си **поиграете с кода** и да го **промените**.

|  |  |
| --- | --- |
| Icon  Description automatically generated | Това е ваш собствен проект. **Бъдете уникални**. Не бъдете copy/paster!   * Имплементирайте свои **собствени функции**. * **Имплементирайте кода сами**, като използвате свой собствен стил на кодиране, форматиране на кода, коментари и т.н. * Направете проекта **по-интересен**. Научете, като играете с кода и добавяте свои собствени промени. |

Ето **няколко идеи** за това какви други функционалности можете да добавите.

### Добавете логика за четирима играчи

Помислете за начин да **добавите опция** за **четирима** играчи.

### Добавете опцията „Война във войната“

Можете да приложите следната **функция на играта**: ако сте във **война** и някоя от **трите карти**, които всеки играч **поставя** **в** **набора** **от карти**, също е с **еднакви сили на картата** (трябва да ги погледнете и да проверите), тогава играчите са **отново във война** и така нататък.

### Създайте GUI приложение

Можете да създадете **GUI приложение** с **WinForms** за текущата игра.

### Допълнителни идеи

* Има много **варианти** на **играта**, можете да ги **разгледате** и да **изберете** **един**, който да **имплементирате**. Ето някои идеи: <https://www.pagat.com/invented/war_vars.html>.
* Може да добавите и **свои идеи**

### Commit-ване в GitHub

Сега е време да **качите промените в GitHub**!

|  |  |
| --- | --- |
| Icon  Description automatically generated | Много е важно да **commit-вате често** вашия код в GitHub. По този начин създавате **богата commit история** за вашия **проект** и **профил**:  A picture containing chart  Description automatically generated |

## Write a README.md File

Силно препоръчително е да напишете **документация на вашия проект** в GitHub, за да опишете какво представлява. Нека да направим **README.md** файл за нашия проект. Можем да го редактираме от GitHub репото.

Graphical user interface, text, application, email

Description automatically generated

### Секции на документацията

Добавете информация за проекта, използвайки **Markdown**: цели на проекта, използвани технологии, скрийншоти, живо демо и т.н. Обикновено файлът съдържа следните **секции**:

* **Име** на проекта (трябва да отговаря на въпроса "Какво има в проекта?")
* **Цели** на проекта (какъв проблем решаваме, напр. определена игра)
* **Решение** (описва как решаваме проблема 🡪 алгоритми, технологии, библиотеки, технологични рамки и т.н.)
* Линк към **source code**
* **Скрийншоти** (скрийншоти от проекта в различни ситуации)
* **Живо демо** (демо, което може да се достъпи и тества с един клик)

### Цели на проекта

Започнете документацията, като опишете **целите на проекта**. Какъв проблем решава?

### Примерна документация

Това е пример как може да изглежда **документацията** на проекта. Не ви съветваме да копирате директно:

Chart, radar chart

Description automatically generated

|  |  |
| --- | --- |
| Icon  Description automatically generated | **Напишете документацията сами!** Не копирайте примерната такава!  Това е **вашият уникален GitHub профил** и вашият проект. **Бъдете различни** от останалите. |

Намерете подходящо **изображение** и го добавете. Можете да го направите по следния начин:



Можете да добавите информация за **входовете** и **изходите** на проекта:

Graphical user interface, text, application, email

Description automatically generated

### Вашето решение

Опишете как сте **решили проблема**: алгоритми, технологии, библиотеки, технологични рамки и др.

### Линк към source code

Добавете **линк** към вашия **source код**.



### Скрийншоти

Добавете **скрийншоти** на проекта:

1. **Направете скрийншот** с предназначен за това инструмент (например [Snipping Tool](https://support.microsoft.com/en-us/windows/open-snipping-tool-and-take-a-screenshot-a35ac9ff-4a58-24c9-3253-f12bac9f9d44) за Windows).
2. **Поставете** скрийншота във файла за документация, използвайки [Ctrl+V]:

Примерни скрийншоти за играта "Игра на война":

Text

Description automatically generated

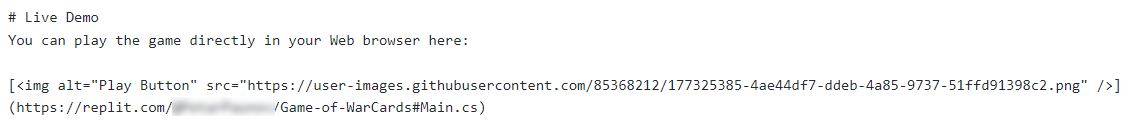
## Качете приложението си в Replit

Нека да добавим **проекта** в **Replit**, за да можем да го **споделяме с приятели** и да го добавим в нашия **GitHub** профил. Вече знаете как да направите това.

Забележка: използвайте **Mono C#** шаблон в **Replit** за **разгръщане на вашата игра** **"Game Of War"**, тъй като със C# ще бъде **твърде бавно**, за да играете нормално. Трябва да имате **всичките си класове в един файл**, за да могат да **работят заедно**.

Сега можете да **споделята** приложението ви с приятели.

## Добавяне на Replit линк към README.md

Сега може да добавите **демо**, което да се тества с **един клик**, към вашето **GitHub README**: 

Може да направите **скрийншот** от **Replit.com** и да го добавите в README файла с [**Ctrl+V**]:

Ето как трябва да изглежда след промените във вашата README.md документация:

Graphical user interface, text

Description automatically generated

Сега завършихме **конзолната игра** и имаме нов **проект** в нашето GitHub портфолио.