# Практически проект: Змия



Днес ще създадем конзолната игра "[**Змия**](https://en.wikipedia.org/wiki/Snake_(video_game_genre))". Това е игра, в която потребителят **контролира фигура с формата на змия**, която се движи в рамките на зададено поле, взимайки **храна** по пътя.

Text

Description automatically generated A picture containing text

Description automatically generated

## Писане на кода на играта

Нека да създадем играта "**Змия**".

### Създаване на Visual Studio проект

Първо, **стартирайте** **Visual Studio** и създайте ново C# конзолно приложение. **Важно** е да **преименувате главния клас** от **Program.cs** на нещо **по-смислено** – например **SnakeGame.cs**.

След като създадете проекта, трябва да изглежда така:

Graphical user interface, application, Word

Description automatically generated

## Създаване на GitHub репо

**Изберете** един от начините за **създаване на репо**, които сме ви показвали – от **GitHub сайта** или директно от **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, application

Description automatically generated with medium confidence

### Стъпки за имплементация на играта

Ще започнем, като имплементираме **два класа** и **една** **енумерация** (enum).

Първият **клас Position** ще репрезентира всеки елемент от змията. Класът **Position** ще има:

* **Конструктор**, който приема:
  + **Ред** (row) на текущия елемент
  + **Колона** (column) на текущия елемент
* **Свойства**:
  + int Row – реда на текущия елемент
  + int Row – колоната на текущия елемент

Вторият **клас** **Food** ще репрезентира **храна**, която змията **трябва да изяде**, за да **порасне** и да се **движи по-бързо**. Класът **Food** ще има:

* **Полета**:
  + **char[] foodSymbols –** съдържа различни **символи за храната**, които ще визуализираме на конзолата
* **Конструктор**, който приема:
  + **Височина** (height) на конзолния прозорец
  + **Ширина** (width) на конзолния прозорец
* **Свойства**:
  + int XCoordinate – хоризонталната (x) координата на храната
  + int YCoordinate – вертикалната (y) координата на храната
  + char Symbol – визуализация на храната чрез символ

Третият и последен **клас Snake** ще репрезентира змията, която ще се движи в рамките на конзолния прозорец. Класът **Snake** ще има:

* **Константи**:
  + **int StartingElementsCount** – началните елементи на змията
  + **int MaximumSnakeSpeed** – максималната скорост на змията
* **Конструктор**, който приема:
  + int bordersOffset – отстояние на границата (border)
  + int boardHeight – височина на полето (конзолния прозорец)
  + int boardWidth – ширина на полето
* Свойства:
  + **int** **StartingXPosition** – начална X-координата на змията
  + **int** **StartingYPosition** – начална Y-координата на змията
  + Queue<Position> Elements – съдържа **елементите на змията**
  + Position NewHeadPosition – елемент с **нова позиция** на **главата на змията**
* **Методи**:
  + Queue<Position> InitialCreate() – **създава началната змия**
  + bool IsOutOfBouders(int boardHeight, int boardWidth, int bordersOffset) – проверява дали **змията** е **извън границите** на прозореца
  + bool IsBitten() – проверява дали **змията** се е **захапала**
  + int GetReversedSpeed() – взима обратната скорост на змията, за да визуализираме **правилното число** на конзолата.

**Енумерацията** Direction съдържа константи за **всяка посока**, в която **змията** може да се **движи**:

* **Right**
* **Left**
* **Down**
* **Up**

След като сме **готови** с **класовете** и **енумерацията**, ще имплементираме **главната логика** на играта. За тази цел се нуждаем от **седемнадесет метода**, които ще използваме, за да може играта да **работи правилно**.

* void SetWindowProperties() – задава **ширината** и **височината** на **конзолния прозорец**
* void DrawBorders() – създава **границите** и ги чертае на конзолата
* void DrawNewFood() – **премахва** **старата храна** от конзолата и визуализира **нова**
* void GenerateNewFood() – създава нова инстанция на **храна** със **случайни** **символ** и **координати**
* bool FoodIsInSnake() – проверява дали координатите на **генерираната храна** съвпадат с някой от **елементите на змията**
* void PlayGame() – **стартира играта**
* void DrawSnake() – визуализира **всеки елемент от змията** на конзолата
* void ChangeSnakeDirection() – **сменя посоката на движение** на змията в зависимост от **натиснатия** **клавиш** на клавиатурата
* void GenerateSnakeNewHead() – генерира **нова глава** за **змията** с **нови координати**, в зависимост от **натиснатия клавиш**
* string GetEndGameMessage() – връща съобщение за **край на играта**, когато змията се е **захапала** или е излязла **извън границите**
* void WriteEndGameMessage(string outcomeMessage) – показва **съобщение** в **панела с резултати**
* bool SnakeEatsFood() – проверява дали **храната** и **главата на змията** са на **едни и същи координати**
* void ModifyPlayerAndSnake() – **добавя точки** към резултата на потребителя и **увеличава скоростта на змията** с **1**
* void MoveSnake() – извършва **движението на змията** чрез **премахване** на **първия елемент** от опашката
* void DrawSnakeHead – показва **генерираната глава на змията** на конзолата
* string GetSnakeHeadElementSymbol() – определя **символа** за **глава на змията** в зависимост от **посоката**
* void DrawScorePanel() – чертае **панел с резултати** на конзолата

Това е **всичко**, от което се нуждаем, за да може **играта да работи**. Нека да продължим с **имплементацията**.

## Дефиниране на класовете

### Клас Position

Нека да създадем първия клас и да го наименуваме Position. Вече трябва да знаете как да го направите:

A screenshot of a computer

Description automatically generated

Първото нещо, което трябва да направите, е да създадете **свойствата на класа**. Ще използваме **две свойства** за **реда** и **колоната**. Те ще бъдат от тип **int** и ще ни помогнат да следим **всеки елемент от змията**.

A picture containing text

Description automatically generated

Готови сме със **свойствата**. Следващото нещо, което трябва да **добавите**, е **конструктора**. Създайте конструктора и го направете така, че да **приема** **ред** и **колона**, които ще подадете при **създаването на инстанция** на класа.

Text

Description automatically generated with medium confidence

Накрая, задайте **реда** и **колоната**, подадени през **конструктора**, на свойствата **Row** и **Col**. Трябва да изглежда така:

Application

Description automatically generated with low confidence

Класът трябва да изглежда така:

Картина, която съдържа текст

Описанието е генерирано автоматично

Това е всичко, което ни трябва в **този клас**. Следващата стъпка е да имплементирате класа **Food**.

### Клас Food

Създайте **нов клас** и го наименувайте **Food**. Вече трябва да знаете как да го направите:

Text

Description automatically generated with low confidence

Първо създайте **трите свойства** на класа. **Първите две** ще бъдат от тип **int** и ще представляват **X** и **Y координатите** на **храната**.

Text

Description automatically generated with low confidence

**Третото** свойство ще бъде от тип **char** и ще представлява **символа на храната**.



Сега, **преди да създадете конструктора**, ще се нуждаете от **частно поле**. То ще бъде от тип char[] и ще съдържа **всички различни символи за храна** (начина, по който ще изглежда храната, когато си покаже на конзолата).



**\*Бележка** - Можете да **копирате** тези или да **създадете ваши собствени**, за да може проектът да изглежда **по-различно** от този на вашите съученици.

|  |
| --- |
| '#', '@', '%', '\*', '&' |

Това е всичко, от което се нуждаем за символите. Остава да **създадем** **конструктора** на класа и ще е **готов**.

Създайте конструктора, така че да **приема височината** и **ширината** на **прозореца**.

A picture containing text

Description automatically generated

В **конструктора** първо се нуждаем от **променлива** от тип **random**. Тази променлива ще ни помогне да **генерираме случайни координати** и **символ** за храната, когато я създаваме.



Следващата **стъпка** е да зададем **случайни числа** като **координати**. Ще го направим чрез нашата **random променлива**.

The next **step** is to **assign** a **random** **number** to our **coordinate** **properties**. We will do this by **using** our **random** **variable**.



Тук започваме от **2**, защото иначе има вероятност да **генерираме храната** върху **границите** (при което змията няма да може да я достигне).

Накрая трябва да **генерираме случаен символ за храната**. Опитайте се да го направите сами:



Това е всичко, от което се нуждаем за нашия **клас Food**.

Картина, която съдържа текст

Описанието е генерирано автоматично

Внимавайте с **подредбата** на свойствата, полетата, конструкторите и т.н. в класа, за да е **верен**!

Нека да **продължим** със **следващия**.

### Клас Snake

Създайте **нов клас** и го **наименувайте Snake**.

Graphical user interface

Description automatically generated with low confidence

Първо трябва да създадете **две константи** за **броя** на **началните елементи** на змията и за **максималната скорост** на змията.



Следващото нещо, от което се нуждаем, са **пет свойства**. Първите две са за началните **X** и **Y позиции**.

Text

Description automatically generated

Следващото свойство е за **скоростта на змията**.

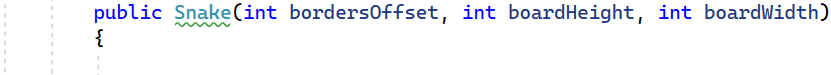


**Последните две свойства** са за **елементите на змията** и за **новата глава** на змията. Свойството за елементите ще бъде от тип Queue<Position>, а свойството за **главата на змията** ще бъде от тип Position (позицията ще бъде създадена по-късно).

Text

Description automatically generated with medium confidence

**Следващата стъпка** е да зададем свойствата на **конструктора**. Създайте конструктора, така че да **приема отстояние на границата**, **височина** и **ширина** на границата.



В конструктора създайте **променлива** от тип Random. Те ще ни помогнат да зададем **случайни координати** като **началните позиции** на змията всеки път, когато **започне нова игра**.



**X** **позицията** ще бъде случайно число между **отстоянието** на границата и **височината минус отстоянието**. Тук ще извадим отстоянието на границата, за да предотвратим възможността **змията** да се **генерира** **върху границите**.



**Y** **позицията** ще бъде случайно число между **отстоянието** на границата и **ширината**, **делено** на **две**. Тук ще разделим на две, за да предотвратим възможността **змията** да се **генерира близо до лявата граница**.



След това задаваме **скоростта**, която ще е **равна** на **константата** за **максималната скорост**.



**Последното нещо** е да **създадем** елементите на змията. Ще ги създадем, като извикаме метода InitialCreate(). Тъй като този метод **още не е добавен**, задайте резултата като **стойност на** Elements и го **закоментирайте**.

A close up of a logo

Description automatically generated

Сега **конструкторът е готов** и всичко има правилните стойности.

Следващата стъпка е да създадем **методите за змията**.

#### Имплементация на метода InitialCreate()

Методът ще бъде от тип Queue<Position> и ще създава и добавя в опашката всеки **елемент на змията** като променлива. Накрая ще **върне създадените елементи**, за да ги зададем на **свойството** Elements в конструктора.

Създайте метода и го направете **частен**, защото ще го използваме **само в класа Snake**.

Text

Description automatically generated

Първото нещо, което трябва да направим в метода, е дали да проверим **началната позиция Y** е **четно число** и ако **не е**, трябва да **добавим 1**, за да го **направим четно**. Ако не направим това, змията **няма да може** да **достигне лявата граница** (змията ще **умре**, преди да достигне границата).

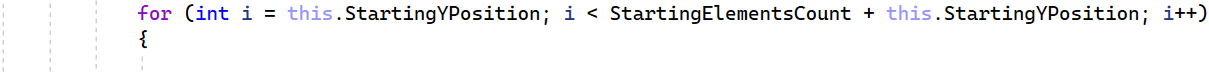
Text

Description automatically generated

След това **създаваме променлива**, която ще съхранява **новосъздадените елементи**.



Сега се нуждаем от **for-цикъл**, който започва от **началната позиция Y**, и итерира докато стойността е **по-малка** от **броя на началните елементи плюс** **началната позиция Y**.



В цикъла добавяме към елементите нов **Position** с **началната позиция X** и **i** като параметри.



Накрая **върнете елементите**.



Това е всичко, което ни трябва за **този метод**. Сега можем да се **върнем в конструктора** и да **откоментираме тази част**.

Text

Description automatically generated

Нека да продължим със **следващия метод**.

#### Имплементация на метода IsOutOfBorders(int boardHeight, int boardWidth, int bordersOffset)

Методът ще бъде от тил **bool** и ще проверява дали генерираната **глава на змията** е **извън границите**. За този метод ще използваме **ламбда израз**.



Сега трябва да проверим дали свойството **row** на главата е **по-малко от едно** или дали е **по-голямо** от височината на полето минус отстоянието на границата.



По този начин проверяваме **горната** и **долната граница**. Обмислете начин да проверите **лявата** и **дясната**.



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

#### Имплементация на метода IsBitten()

Методът ще бъде от тип bool и ще проверява дали **главата на змията** е на **същите координати** като някой от елементите в **опашката**. Отново ще използваме **ламбда израз**, но **комбиниран** с **LINQ**.



Сега ще проверим дали някой от елементите има **column** и **row**, които са същите като тези на **главата на змията**. Вече знаете как да го направите.



Това е всичко за този метод. Нека да създадем **последния метод** за този клас.

#### Имплементация на метода GetReversedSpeed()

Последният метод на класа ще върне int, обратната скорост на змията.

Обмислете начин да върнете **правилната скорост**. Спомнете си, че имаме **константа** за **максималната скорост** – можете да работите с нея.



Това е **всичко за клас**. Целият клас Snake трябва да изглежда така:

Картина, която съдържа текст

Описанието е генерирано автоматично

Нека да продължим с **енумерацията**.

### Енумерация Direction

Енумерацията ще съдържа **всяка посока**, в която змията може да се движи. Вече знаете как се създава енумерация.

A computer screen shot of a code

Description automatically generated

Остава да създадем **константи** за **всички възможни посоки** (right, left, down, up) и да им **зададем индекси**.

A picture containing table

Description automatically generated

Това е **всичко, от което се нуждаем**. След като финализирахме тази част, сме **готови с подготовката на класовете и енумерацията**. Нека да продължим с **инициализацията на играта**.

## Инициализация на играта

Първото нещо, което трябва да направим, е да отидем в нашия **стартов клас** и да инициализираме играта. В нашия случай той е наименуван SnakeGame:

A screenshot of a computer

Description automatically generated

Първо задаваме **изходното кодиране** да е **Unicode**, защото използваме **Unicode** **символи** за **змията**.



Нуждаем се от **четири константи**, които ще използваме, за да подготвим **конзолния прозорец** и да инициализираме **змията** и **храната**.

Text

Description automatically generated

След константите ще използваме нашия първи метод SetWindowProperties(). Тъй като методът **още не е имплементиран**, го **извикайте** и **засега го закоментирайте**.



След това трябва да създадем **масив** от тип Position. Този масив ще съдържа **всички възможни посоки**, в които **змията** може да се **премести**.

Text

Description automatically generated

След като имаме посоките, ще се нуждаем от **две** int **променливи**. Първата ще съдържа **точките на потребителя**, а втората ще съдържа **началната посока** на **змията** (ще използваме енумерацията за това).

A black and blue letters and symbols

Description automatically generated with medium confidence

Следващото нещо, което трябва да направим, е да извикаме метода DrawBorders(). Той ще **създаде** и **визуализира границите** на конзолата, но тъй като **още не е имплементиран**, го **извикайте** и го **закоментирайте**.



Сега трябва да **инициализираме храната**. Първоначално тя ще бъде **null**, защото ще създадем **нова инстанция** за нея в метода DrawNewFood().



След храната, **инициализираме** **змията**. Създаваме нова инстанция на змията и подаваме константите BorderOffset, BorderHeight and BorderWidth.



Накрая трябва да извикаме методите DrawNewFood() and PlayGame(). Методите **още не са имплементирани**, така че ги **извикайте** и ги **закоментирайте**.

A picture containing text

Description automatically generated

Това е всичко, което ни трябва за играта. Сега само остава а имплементираме **методите**, за да може играта да **работи правилно**. Нека да продължим с тях.

### Имплементация на методите

#### Имплементация на метода SetWindowProperties()

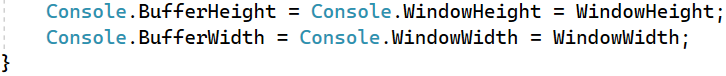
Този метод ще задава **свойствата**, които ни трябват за **конзолата**. Височината и ширината на конзолата, както и **видимостта на курсора**, ще имат стойност false. Методът ще направи настойките **преди да започне играта**.

Създайте **метод** и направете видимостта на курсора **false**, за да **не се вижда по време на играта**.

A picture containing text

Description automatically generated

Сега трябва да зададем **конзолния буфер** и **размера на прозореца**. Ще го направим, като достъпим свойствата за **ширината** и **височината** на конзолата и ги направим равни на **константите** за **ширина** и **височина**. Можем да го направим, използвайки следния синтаксис:



Сега можем да **откоментираме метода** и да тестваме **дали работи**.

🡪

Text

Description automatically generated

След като изпълним програмата, конзолата трябва да се **събере до зададените размери** от константите.

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

#### Имплементация на метода DrawBorders()

Този метод ще **рисува вертикалната** и **хоризонталната** граница на **полето за игра**. Това ще покаже границите и ще ни помогне да **виждаме "стените"**.

Създайте метода и изолзвайте **StringBuilder**, за да добавите границите:

Картина, която съдържа текст

Описанието е генерирано автоматично

Класът StringBuilder може да бъде използван, за да се **модифицира стринг**, **без** да се създава **нов обект**.

**Бележка**: Можете да **научите повече** за StringBuilder тук: <https://learn.microsoft.com/en-us/dotnet/api/system.text.stringbuilder?view=net-7.0>.

Следващата стъпка е да **изберете цвят** за **границите**. Можете да го направите със свойството **Console.ForegroundColor**. В този случай ще използваме **цвета cyan**.



След това трябва да зададем **знаците** на **хоризонталната** и **вертикалната граница**, които по-късно ще използваме в нашия **StringBuilder**.



Следващото нещо, което трябва да направим, е да зададем **дължината на хоризонталната граница**. Трябва да направим **нов стринг** със символа (piece), който сме задали. За броя на символите трябва да извадим **отстоянието на границата** от **ширината на полето** и да **добавим 2**.



След това трябва да зададем **празна хоризонтална страна**. Можем да го направим, като създадем **друг стринг** със символа **space** и за **брой** – **отстоянието на границата**, извадено от **ширината**.



След това трябва да **добавим символите** към **StringBuilder**-а. Това ще добави **хоризонтална линия** в играта



За **вертикалната линия** се нуждаем от for-цикъл, който ще създава **редовете**. Цикълът ще **спре**, преди **редът** да стане **равен на височината** на **полето** **минус отстоянието на границата**.



В **цикъла** ще добавим **символите**, за да създадем **вертикалната линия**:



След това трябва да **създадем долната хоризонтална линия** по същия начин, по който направихме **горната**. Можете да направите това сами:



Накрая трябва да направим **нов стринг** с нашия StringBuilder:

Картина, която съдържа текст

Описанието е генерирано автоматично

#### Имплементация на метода DrawScorePanel()

Този метод ще **съхранява точките** на потребителяи **скоростта** на змията по време на играта.

Създайте метода и вашия StringBuilder. В StringBuilder-а ще добавим **точките** и **скоростта**.

A close-up of a logo

Description automatically generated

Сега трябва да вземем **скоростта на змията**. Ще го направим, използвайки **метода**, който направихме в **класа за змията**:



След като вземем **скоростта**, трябва да добавим нея и точките към StringBuilder-а. Също трябва да добавим някакъв **описателен текст**, за да стане ясно какви са тези числа:



След това трябва да **направим нов стринг** с нашия StringBuilder:



Сега **задаваме позицията** на текста и рисуваме **панела за резултата**:



Нека да стартираме приложението и да видим какъв е **резултатът**. Ако всичко работи правилно, **долната част на конзолата** трябва да изглежда така:



**Стартираната игра** трябва да изглежда по подобен начин:

A screen shot of a computer

Description automatically generated

И ето как трябва да изглежда, когато змията се **блъсне в стената**:

Картина, която съдържа текст

Описанието е генерирано автоматично

## Качване на проекта в GitHub

Качете промените в проекта в **GitHub**, както вие прецените – чрез **TortoiseGit**, чрез **Git Bash** или от **Visual Studio.**

A screenshot of a computer

Description automatically generated

## \* Модификации на кода

Сега можете да добавите и **други функционалности** към кода.

|  |  |
| --- | --- |
| Icon  Description automatically generated | Това е ваш собствен проект. **Бъдете уникални**.   * Имплементирайте **свои собствени функционалности**. * Имплементирайте кода **сами**, добавете ваш стил, форматиране, коментари и т.н. |

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

### Добавяне на Unicode символи

Потърсете за **по-интересни символи** за **змията** и за **храната**.

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

### Минаване на змията през стената

Направете играта **по-лесна** за потребителя, като позволите на змията да **минава през стените**. Например, ако докосне **дясната граница**, змията да продължи да се движи от началото (лявата граница), за да може играчът да **не загуби**.

### Създаване на GUI приложение

Може да създадете **GUI приложение** за играта.

### Допълнителни идеи

* Създайте **импровизирана база данни**, която да съхранява **името на потребителя** и разултата, и ги **визуализирайте на конзолата**.
  + Използвайки JSON – <https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/how-to?pivots=dotnet-6-0>.
  + Използвайки XML – <https://learn.microsoft.com/en-us/dotnet/standard/serialization/how-to-serialize-an-object>.
* Може да добавите и **свои идеи**

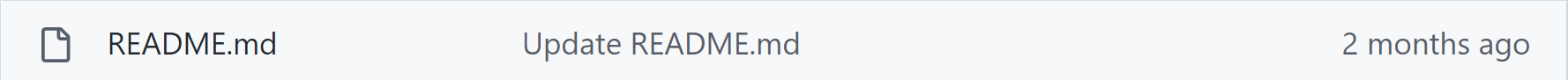
### Commit-ване в GitHub

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 |

## Създаване на README.md файл

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.



### 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:

A green snake with a red tongue

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:



### Контроли

Добавете информация за **контролите на играта**:

A screenshot of a computer

Description automatically generated

### Вашето решение

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

### 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 "Snake":

Картина, която съдържа текст, монитор, екранна снимка

Описанието е генерирано автоматично

## Качване на 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.

**Бележка**: използвайте темплейта Mono C# в Replit за деплой на играта, тъй като C# ще бъде твърде бавен, за да играете нормално. Трябва да имате **всички класове в един файл**, за да може да **работят заедно**.

Поздравления! Успешно завършихте **конзолната игра Snake** и имате още един проект в **GitHub портфолиото ви**.