# Codebreaker

Das erste etwas längere Programm soll Ihnen einige Elemente der Programmierung spielerisch näherbringen.   
Es geht darum, eine „Geheimzahl“ zu raten, also einen Code zu „knacken“.

Jede Variante sollten Sie zunächst versuchen selbstständig zu lösen. Wenn Sie keine Lösungsansätze finden, suchen Sie sich einen Tischnachbarn und versuchen es gemeinsam. Wenn Sie dabei weitere Hilfe brauchen, lesen sie im online zur Verfügung gestellten Buch „C# – Grundlagen der Programmierung“ 🡪 Dateien „VCSPNET\_2019 - … .PDF“. Insbesondere die Kapitel „Sprachgrundlagen und Kontrollstrukturen“ sind hier hilfreich.   
Zusätzlich sind an mehreren Stellen im Dokument Links mit Hinweisen zu einzelnen Themen zu finden.

Legen Sie bitte zunächst eine Projektmappe mit dem Titel „Codebreaker“ an. In der Projektmappe legen Sie bitte für jede Variante ein neues Projekt an, so dass jede Variante in einer eigenen Main-Methode steht.  
Als Projektname können Sie z.B. „Variante1“, „Variante2“, etc. wählen.

Durch dieses Verfahren kann man auch später noch ihren Fortschritt nachvollziehen. Wenn alle Stricke reißen, dann rufen Sie mit allen gemeinsam den Lehrer!

<https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide/classes-and-structs/methods> 🡪 Methoden in C#

##### Variante 1

Für die erste Variante müssen Sie sich eine Geheimzahl ausdenken und über die Tastatur eingeben. Der Mitspieler soll nun eine Zahl eingeben und bekommt von ihrem Programm gesagt „Die Zahl ist richtig“ oder „Die Zahl ist falsch“.

(Sie üben dabei: Eingabe/Ausgabe, sowie einfache bedingte Anweisungen und die Benutzung von Variablen.

<http://www.holzers-familie.de/schule/csharp/c_sharp_Datentypen.html> 🡪 Variablen und Datentypen

<https://docs.microsoft.com/de-de/dotnet/api/system.console.readline?view=netcore-3.1> 🡪 Einlesen von Tastatureingaben

<https://docs.microsoft.com/de-de/dotnet/api/system.convert?view=netcore-3.1> 🡪 Konvertieren von Datentypen

<http://openbook.rheinwerk-verlag.de/visual_csharp_2010/visual_csharp_2010_02_004.htm> 🡪 Vergleichsoperatoren

<https://docs.microsoft.com/de-de/dotnet/csharp/tutorials/intro-to-csharp/branches-and-loops-local> 🡪 Verzweigungen

##### Variante 2

Nun erweitern Sie ihre Ausgabe um die Bemerkung „Die eingegebene Zahl ist zu groß“ oder „Die eingegebene Zahl ist zu klein“ oder „Die eingegebene Zahl ist richtig“

(Sie üben dabei zusätzlich zur Variante 1 komplexere bedingte Anweisungen, Vergleiche mit > und < )

<http://openbook.rheinwerk-verlag.de/visual_csharp_2010/visual_csharp_2010_02_004.htm> 🡪 Vergleichsoperatoren

##### Variante 3

Für die nächste Variante erweitern Sie das Programm um eine Schleife, und geben so lange neue Zahlenversuche ein, bis die Zahl gefunden wurde.

(Sie üben dabei zusätzlich zur Variante 2 den Einsatz einer wiederholten Anweisung (while-Schleife))

<https://docs.microsoft.com/de-de/dotnet/csharp/tutorials/intro-to-csharp/branches-and-loops-local> 🡪 Schleifen

##### Variante 4

Für die nächste Variante erweitern Sie das Programm so, dass Sie nachher dem Mitspieler angeben können, wie viele Versuche er gebraucht hat.

(Sie üben dabei zusätzlich zur Variante 3 den Einsatz eines Zählers)

[http://openbook.rheinwerk-verlag.de/visual\_csharp\_2010/visual\_csharp\_2010\_02\_004.htm](http://openbook.rheinwerk-verlag.de/visual_csharp_2010/visual_csharp_2010_02_004.htm#mj56406ff2bd03fdc1d1139ec50d378566) 🡪 Arithmetische Operatoren

##### Variante 5

Nun kommt der Zufall ins Spiel. Dies ist ein sehr großer Schritt, gerade jetzt, zu Beginn ihrer Programmierer-Laufbahn. Aber es erhöht deutlich den Reiz dieses Programms, wenn man selbst auch nicht weiß welche Zahl geraten werden muss.   
Zunächst benötigt man ein neues Element, welches eine solche Zufallszahl erzeugt. Wie eine solche Funktion in C# heißt und wie sie benutzt wird, finden Sie im unten aufgeführten Link. Bauen Sie diesen Programmcode ein, und versuchen Sie die die zufällige Zahl herauszufinden.   
Aus welchen Zahlenbereich kommt die Zufallszahl, wenn sie keine weiteren Angaben machen?

(Sie üben dabei zusätzlich zur Variante 4 den Einsatz eines Zufallszahlengenerators)

<https://docs.microsoft.com/de-de/dotnet/api/system.random.next?view=netcore-3.1> 🡪 Zufallszahlengenerator

##### Variante 6

Da der Zahlenraum aus der Variante 5 doch ziemlich groß ist, möchten wir ihn etwas einschränken. Im nächsten Schritt soll die Geheimzahl immer zwischen 1 und 999 liegen.

(Sie üben dabei zusätzlich zur Variante 4 die Übergabe von Parametern bei dem Einsatz eines Zufallszahlengenerators)

##### Variante 7

Als nächstes soll der Benutzer selbst entscheiden, zwischen welchen Grenzen der Zahlenraum für die Zufallszahl liegen soll. Lesen Sie die linke und die rechte Grenze ein. Überprüfen Sie, ob die rechte Grenze wirklich eine größere Zahl ist als die linke Grenze. Wenn das der Fall ist erstellen Sie eine entsprechende Geheimzahl mit Random. Ansonsten lassen Sie den Benutzer solange Grenzen eingeben, bis er es richtig macht.

(Hier gibt es außer der Do-While-Schleife nicht viel Neues.)

<https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/do> 🡪 Do-While Schleife

##### Variante 8 (Zusatzaufgabe)

Haben Sie schon gemerkt, dass es gute und schlechte Strategien gibt, um dieses Spiel zu spielen? Überlegen Sie sich nun, wie Sie herausfinden, welche Anzahl von Versuchen „gut“ und welche Anzahl tatsächlich eine schlechte Anzahl von Versuchen darstellt. Formulieren Sie die perfekte Spielstrategie! Rufen Sie nun den Lehrer und erläutern Sie ihm in einem kurzen Fachgespräch ihren Ansatz.

(Sie lernen selbst einen Lösungsalgorithmus zu entwickeln und sie lernen, wie ein Fachgespräch zur Notenfindung bei den späteren, größeren Programmen aussehen wird.)

Nachdem Sie nun wissen, wie man das Spiel am besten spielt, setzen sie ihren Lösungsansatz im Programm um. Dabei sollen die einzelnen Schritte der KI nachvollziehbar auf dem Bildschirm angezeigt werden. Sie können dabei iterativ[[1]](#footnote-1) oder rekursiv[[2]](#footnote-2) vorgehen.

(Für die Implementierung müssen Sie nun den Algorithmus, den Sie zur Suche des richtigen Wertes im Kopf haben wieder in einen Programmierten Algorithmus umsetzen).

1. sich schrittweise in wiederholten Rechengängen der exakten Lösung annähernd [↑](#footnote-ref-1)
2. Bei der rekursiven Programmierung ruft sich eine Prozedur, Funktion oder Methode in einem Computerprogramm selbst wieder auf [↑](#footnote-ref-2)