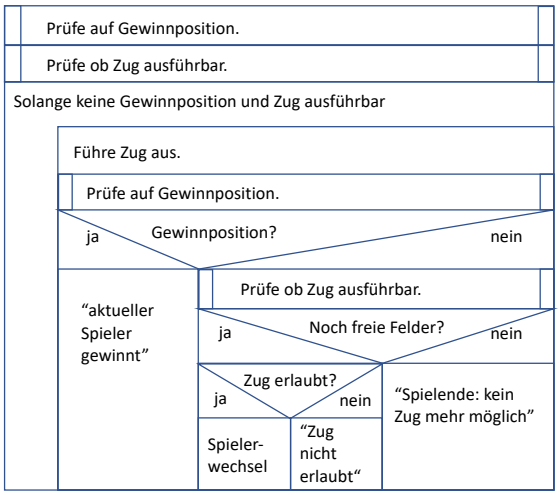
Algorithmen

Ein Algorithmus ist eine **eindeutige Handlungsvorschrift zur Lösung eines Problems**. Er besteht aus endlich vielen Einzelschritten. Algorithmen kann man z.B. mit natürlicher Sprache, bestimmten graphischen Mitteln (z.B. Struktogrammen) oder Programmiersprachen beschreiben.

Das Spiel „Vier gewinnt“ in einzelnen Algorithmen

Bei dem Spiel lassen zwei Spieler abwechselnd Steine von oben auf das Spielfeld fallen. Das Spielfeld besteht aus sieben Spalten und sechs Zeilen. Der Spieler, der zuerst vier Steine zusammenhängend in einer Zeile, Spalte oder Diagonale hat, gewinnt.
Das Spiel kann man als **Algorithmus** darstellen, **dessen Schritte wiederum aus Algorithmen bestehen**:

Abbildung: Algorithmus für "Vier gewinnt"



Wichtige weitere Algorithmen sind die **Prüfung auf eine Gewinnposition** und die **Prüfung, ob ein Zug möglich ist**. Damit können bereits zwei Spieler gegeneinander spielen. Falls der Computer selbst Züge machen soll, sind im Schritt „führe Zug aus“ weitere Algorithmen erforderlich.

Spieltheorie

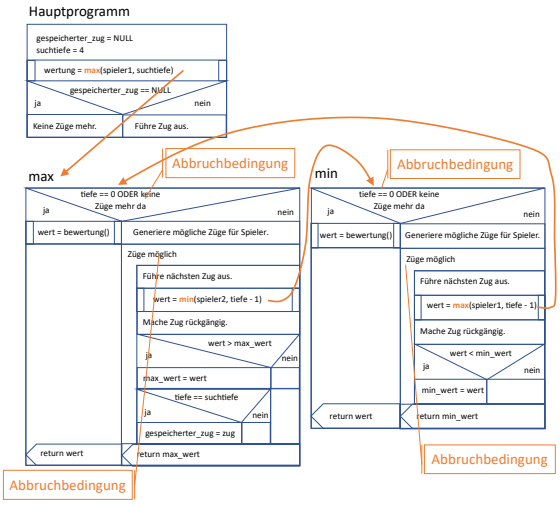
Die Spieltheorie bezeichnet ein Spiel wie “Vier gewinnt” als **Zwei-Personen-Nullsummenspiel**. Nach der Spieltheorie muss es dafür eine **berechenbare Gewinnstrategie** geben. Es wurde bewiesen, dass “Vier gewinnt” **vollständig gelöst** ist und es **für den beginnenden Spieler immer möglich** ist **zu gewinnen**.

Es gibt etwa 4,5 Billionen verschiedene Stellungen – ihre **Vorberechnung würde zu viel Zeit und Speicherplatz kosten**.

Der MinMax-Algorithmus

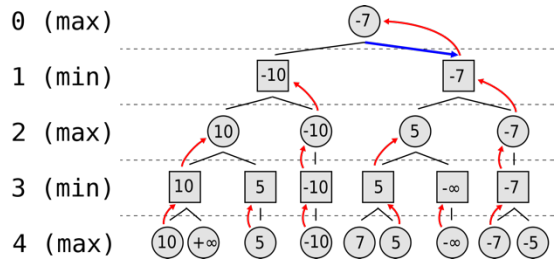
Der MinMax-Algorithmus ist das S**tandardlösungsverfahren für Zwei-Personen-Nullsummenspiele**. Er eignet sich zur Untersuchung von Spielbäumen.
Zwei Spieler – Max und sein Gegner Min – ziehen abwechselnd. Die daraus entstehende Spielposition wird jeweils bewertet. Der beste Zug für jeden Spieler soll genommen werden. Da man nicht sinnvoll alle Stellungen untersuchen kann, wird die Suchtiefe begrenzt.

Abbildung: Struktogramm für MinMax-Algorithmus



Die Suche beginnt bei den unteren Blättern des Suchbaumes und geht zur Wurzel. Damit ist dieser Algorithmus eine sogenannte Tiefensuche im Baum, da immer erst die Nachfolger untersucht werden, bevor die Blätter auf der gleichen Ebene betrachtet werden.

Abbildung: Veranschaulichung von MinMax im Suchbaum



Die sogenannte **Alpha-Beta-Suche** verbessert den Algorithmus und klammert dabei die Knoten aus, die das Endergebnis nicht beeinflussen.

Rekursive Programmierung

Der MinMax-Algorithmus benutzt rekursive Programmierung. Dabei ruft sich eine Funktion **immer wieder selbst** auf bzw. (wie hier) rufen sich Funktionen wechselseitig immer wieder gegenseitig auf. Die **Abbruchbedingung** sorgt dafür, dass sich die Funktionen nicht unendlich oft aufrufen.

Verwendung von Bibliotheken

Bibliotheken bieten bei der Programmierung vorgefertigte Teile, die man wieder benutzen kann und nicht selbst zu entwickeln braucht.
Zur Spieleprogrammierung einschließlich einer graphischen Ausgabe kann die Bibliothek **Pygame** benutzt werden.

Abbildung: Beispiel für graphische Ausgabe mit Pygame

Die Bibliothek **NumPy** stellt die Unterstützung für zweidimensionale Arrays bereit (und noch viele weitere Funktionen für Vektoren, Matrizen, Arrays und die effiziente numerische Berechnung).

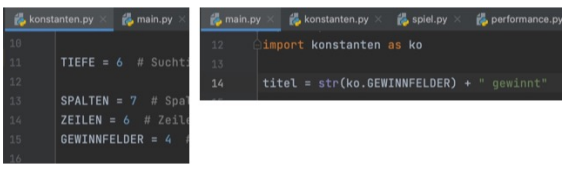
Abbildung: Spielfeld als zweidimensionales Array

	0	1	2	3	4	5	6
0	0.00000	0.00000	0.00000	2.00000	1.00000	0.00000	0.00000
1	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000
2	0.00000	0.00000	0.00000	2.00000	0.00000	0.00000	0.00000
3	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000
4	0.00000	0.00000	0.00000	2.00000	0.00000	0.00000	0.00000
5	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

Struktur eines Projektes

Umfangreicher **Programmcode** sollte in mehrere Python-Dateien **aufgeteilt** werden, um die Übersichtlichkeit zu erhalten. Über die `import`-Anweisung können Inhalte aus anderen Dateien für ein Programm nutzbar gemacht werden.

Abbildung: Import in Python



Weitere Beispiele für **nützliche Vorgehensweisen** bei größeren Projekten sind: Nutzung sprechender Namen für Variablen und Methoden, ausführliche Kommentare, Verwendung von DocStrings, Beachtung von Python-Styleguides, Nutzung von zentral definierten Konstanten anstelle von Literalen.

Listen

Listen sind ein Beispiel für **zusammengesetzte Datentypen**. Sie sind sehr nützlich dafür, Sammlungen von Daten zu behandeln. Eine Liste besteht aus **Elementen**, die durch **Kommas getrennt** und von **eckigen Klammern** umschlossen werden.

Abbildung: Beispiele für Listen in Python

```
namen = ["Arthur", "Bertha", "Christian", "Doris", "Eduard"]
ergebnisse = [23, 43, 66, 125, 87, 99]
leere_liste = []
```

Ihre Elemente können über einen **Index** angesprochen und auch verändert werden (z.B. im Gegensatz zu Tupeln). Der Index **beginnt mit 0**.
Für Listen sind viele nützliche Methoden verfügbar, z.B. zur Behandlung von Elementen, zur Sortierung, zum Zählen usw.

Iteration über Listen

Es ist oft sehr nützlich, **alle Elemente** einer Liste **nacheinander zu behandeln**. Das ist in Python sehr einfach mit einer `for`-Schleife möglich.

Abbildung: Iteration über Listen

```
namen = ["Arthur", "Bertha", "Christian", "Doris", "Eduard"]
for name in namen:
    print(name)
```

Ausgewählte Quellen

Adorf, J. (02. Dezember 2009). *Adversiale Suche für optimales Spiel: Der Minimax-Algorithmus und die Alpha-Beta-Suche.*

Allis, V. (Oktober 1988). *A Knowledge-based Approach of Connect-Four*.

Hilbig, T. (04. November 2014). *Optimierung und Laufzeitanalyse einer künstlichen Intelligenz für das Spiel Vier gewinnt*.

Krusenotto, P. (2016). *Funktionale Programmierung und Metaprogrammierung.* Wiesbaden: Springer Fachmedien.

Pratzner, A. (07. Februar 2022). *Pygame Tutorial*.