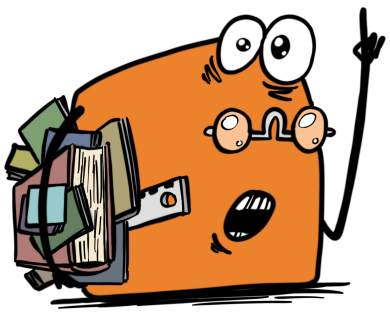
AgentCubes online

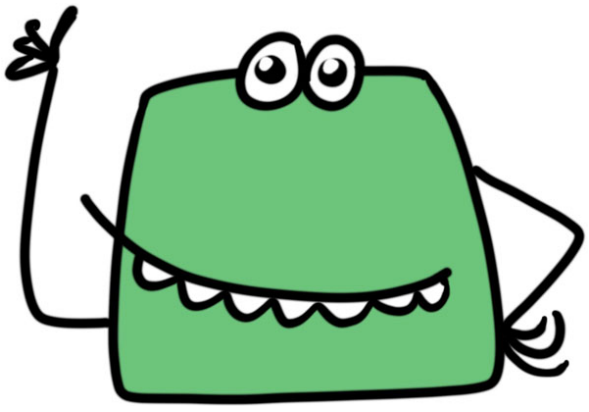
Handbuch



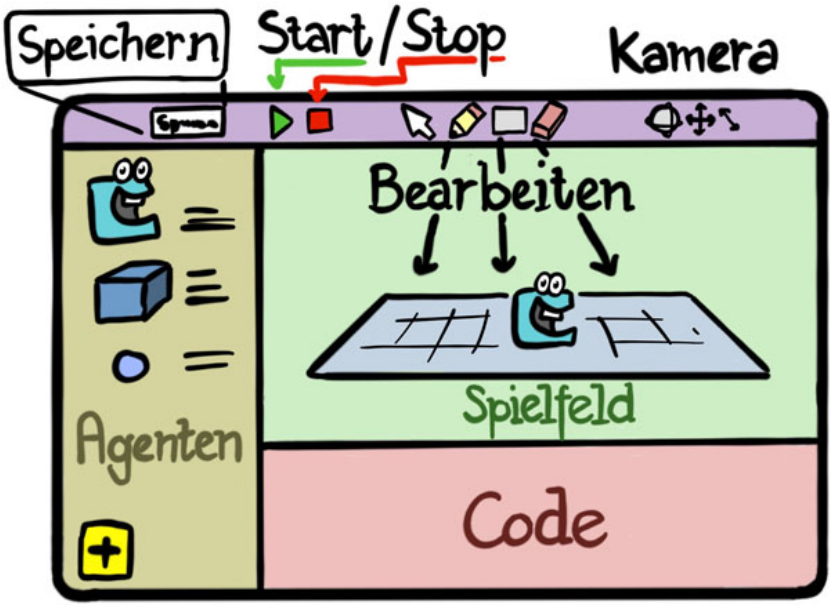
Michael Mittag / PH FHNW

Inhalt

Die Grundlagen



Wie man mit AgentCubes arbeitet

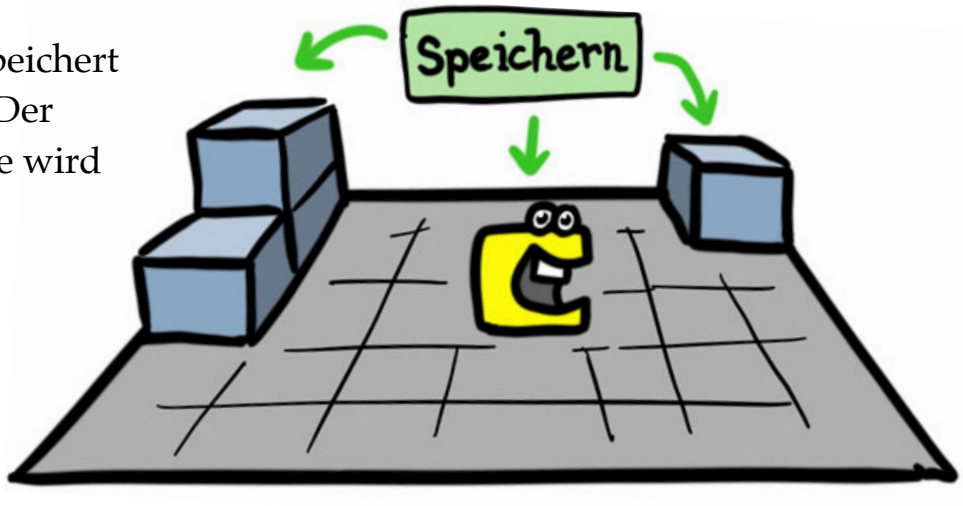


AgentCubes hat drei grosse Bereiche: Links die Agenten, in der Mitte das Spielfeld und unten der Codebereich, in dem das Verhalten der Agenten auf dem Spielfeld definiert wird.

Agenten sind der Kern von AgentCubes. Bevor man irgend etwas machen kann, muss man Agenten erschaffen.

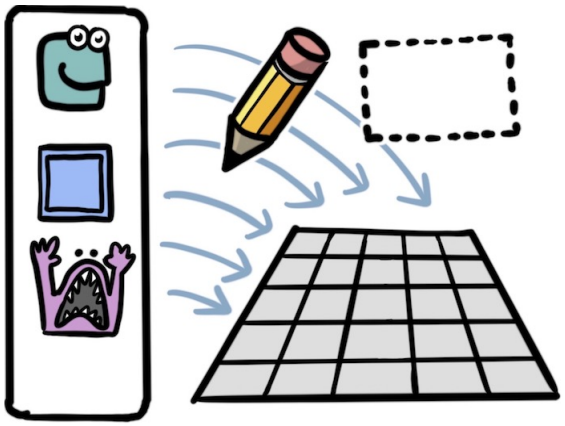
Programmieren erfolgt durch „Drag-and-Drop“ von Befehlen im Codebereich.

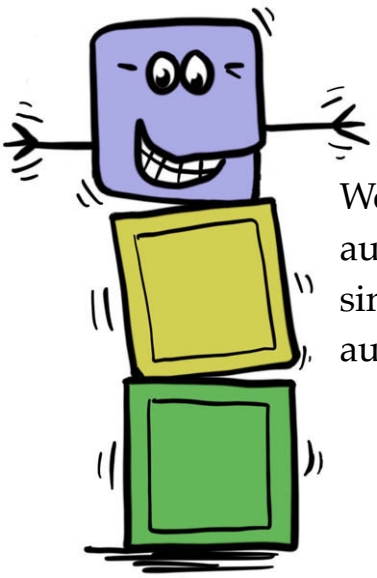
„Speichern“ speichert die Spielwelt. Der Programmcode wird automatisch gespeichert.



Agenten auf dem Spielfeld platzieren

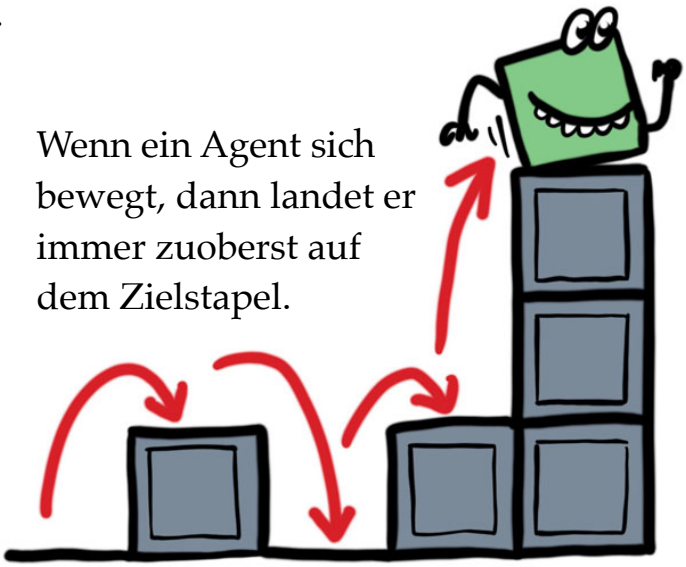
AgentCubes funktioniert, indem Agenten auf einem Schachbrett-artigen Spielfeld platziert werden. Dafür kannst du den Bleistift oder das Flächenwerkzeug verwenden.





Wenn mehrere Agenten auf dem gleichen Feld sind, dann werden sie aufeinander gestapelt.

Wenn ein Agent sich bewegt, dann landet er immer zuoberst auf dem Zielstapel.



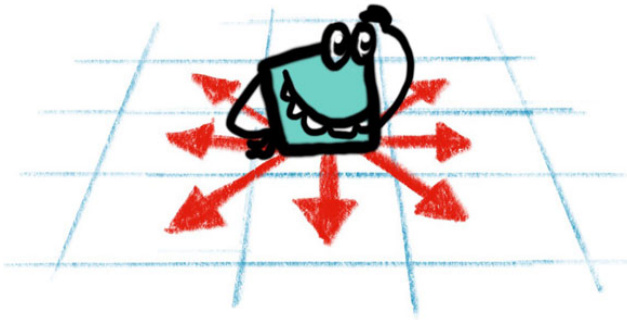
Informatik-Fakten

In den Anfangszeiten spielten fast alle Spiele auf einem Schachbrett-Muster, da die Computer nicht genug Speicherplatz hatten, um ein Bild als einzelne Punkte abzuspeichern. Darüber hinaus konnte man einige wenige Spielfiguren, so genannte „Sprites“, frei bewegen.

Die Tabelle zeigt, welche Systeme wie viele Sprites unterstützen. Kannst du dir denken, weshalb Donkey Kong so viel mehr Sprites unterstützt und der Game Boy weniger als andere Systeme zur gleichen Zeit? (Falls du nicht weisst, wie Donkey Kong oder eine Atari 2600 Konsole aussehen, kannst du sie auf dem Internet suchen).

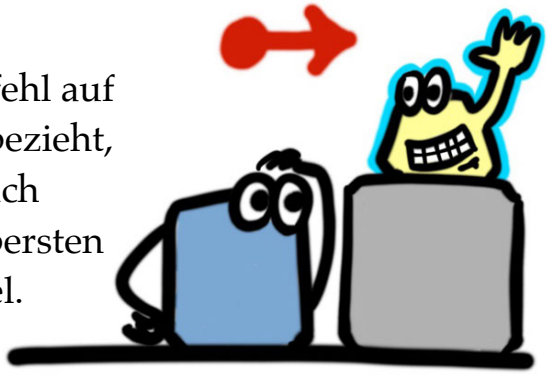
Konsole / System	Jahr	Sprites
Atari 2600 (Konsole)	1977	5
Donkey Kong (Spielhalle)	1979	128
Pac Man (Spielhalle)	1980	8
Commodore 64 (Computer)	1982	8
Nintendo NES (Konsole)	1983	64
Game Boy (Handheld)	1989	40
Playstation (Konsole)	1994	4000

Adressierung

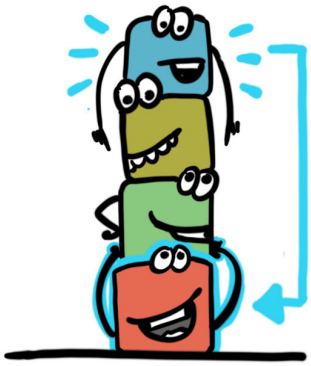


Die Adressierung geschieht in AgentCubes meist über eine Pfeilauswahl. Agenten können damit ein Feld weit in eine beliebige Richtung sehen oder handeln.

Wenn sich ein Befehl auf ein Nachbarfeld bezieht, dann bezieht er sich immer auf den obersten Agenten im Stapel.



Wenn ein Befehl sich auf das aktuelle Feld bezieht, dann bezieht er sich immer auf den Agenten selbst.

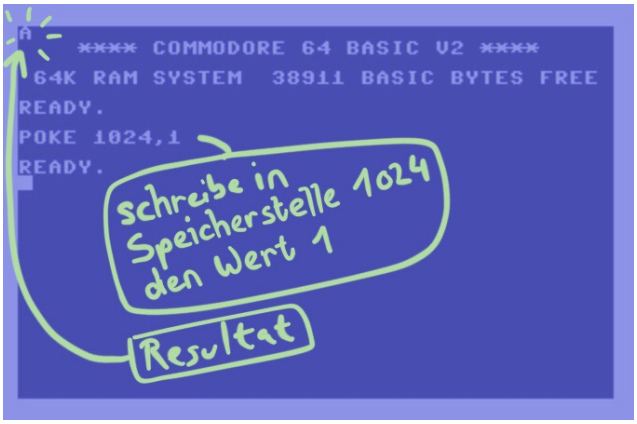


Mit speziellen Befehlen kann man andere Agenten adressieren, zum Beispiel alle Agenten eines Typs oder den untersten Agenten im Stapel.

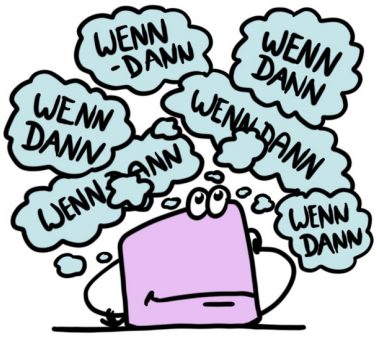
Informatik-Fakten

In den Anfangszeiten adressierte man Computerspeicher direkt. Wenn man beim Commodore 64 Computer den Wert „1“ an Stelle 1024 im Speicher schrieb, dann wurde links oben auf dem Bildschirm ein „A“ angezeigt. Das passierte, weil der Video-Prozessor sich die Informationen zur Bilddarstellung immer von Speicherstelle 1024 bis 2023 holte.

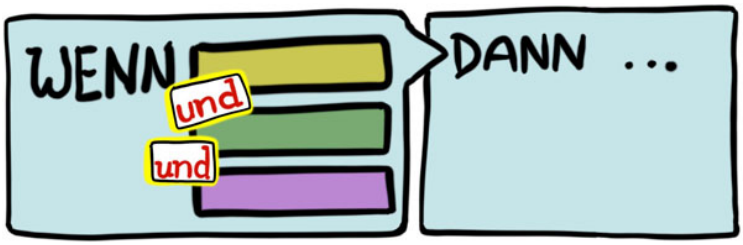
Kannst du dir denken, weshalb das auf modernen PCs so nicht mehr funktioniert?



Wenn-Dann Regeln

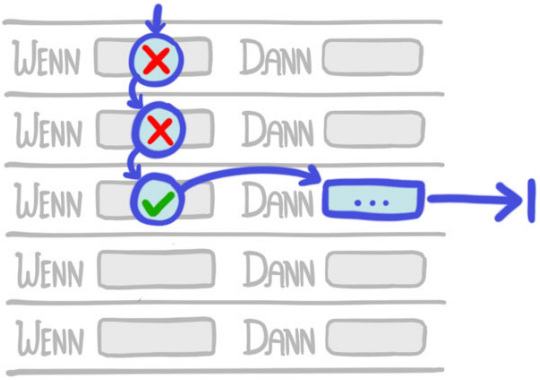


Das Verhalten von Agenten wird durch „Wenn-Dann“ Regeln bestimmt. Die Programmierung erfolgt, indem man im Code-Bereich Wenn-Dann-Regeln für den Agenten zusammenfügt.

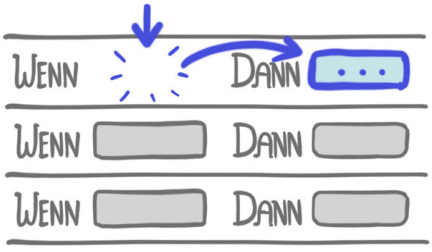


Eine Regel wird ausgeführt, wenn **alle** „Wenn“-Bedingungen zutreffen.

Die Regeln werden von oben nach unten geprüft, bis **die erste** Regel ausgeführt werden kann. Regeln weiter unten werden nicht geprüft oder ausgeführt.



Neue Regeln fügst du mit „+ Regel“ hinzu (unten an der Box).



Wenn eine Regel keine Bedingungen hat, wird sie immer ausgeführt. Achtung: Regeln, die unten dieser Regel stehen, werden nie ausgeführt.

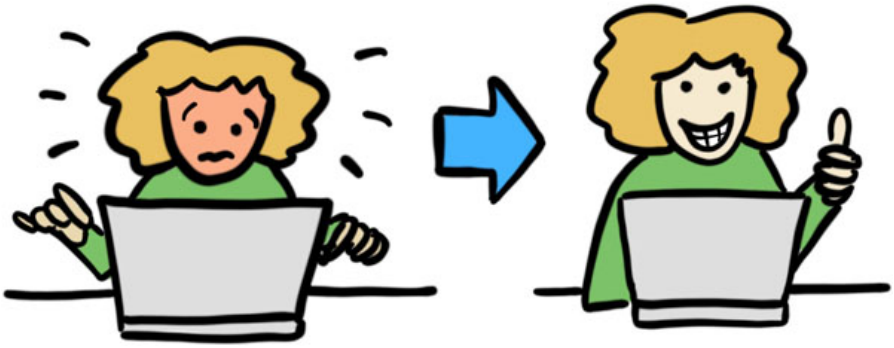
Informatik-Fakten

Wenn-Dann-Regeln werden in fast allen Programmiersprachen mit IF-THEN (das ist englisch für „Wenn-Dann“) geschrieben. Meist gibt es zusätzlich ein ELSE (deutsch: sonst), das ausgeführt wird, wenn die Bedingung nicht zutrifft.

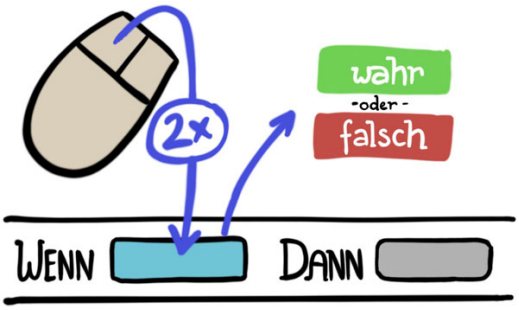
Eine Befehlsfolge wie bei AgentCubes, bei der nur der erste Befehl ausgeführt wird, welcher zutrifft, würde man wie folgt schreiben:

```
if a=0 then
        print „a ist null“
else if a=1 then
        print „a ist eins“
else
        print „a ist weder null noch eins“
```

Fehlersuche

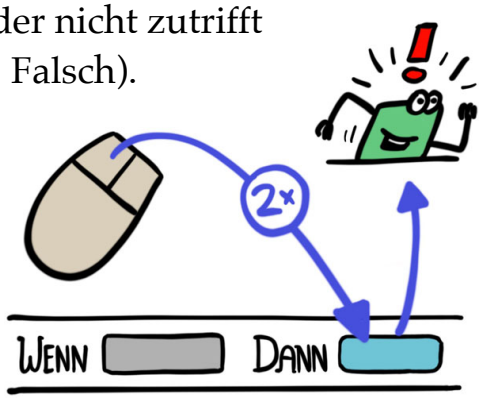


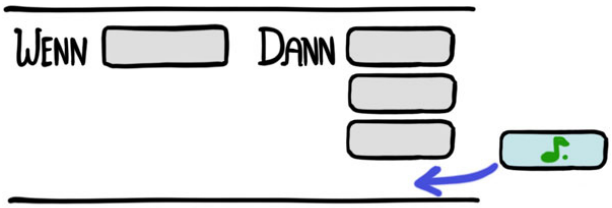
AgentCubes enthält viele Methoden, um Fehler zu finden. Du wirst sehr viel besser werden im Programmieren, wenn du lernst, wie du Fehler finden kannst.



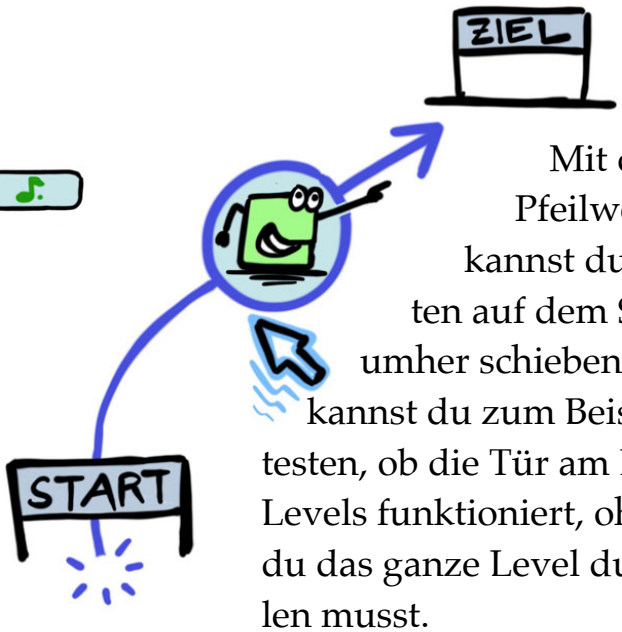
Mit einem Doppelklick auf eine Bedingung sagt dir AgentCubes, ob die Bedingung zutrifft („True“, deutsch: Wahr) oder nicht zutrifft („False“, deutsch: Falsch).

Mit einem Doppelklick auf eine Aktion führt AgentCube die Aktion aus, unabhängig davon, ob die Bedingungen dafür erfüllt sind.





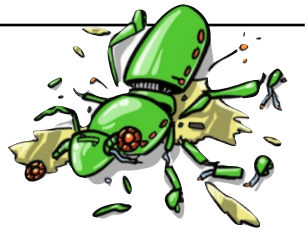
Wenn du einen Tonbefehl einfügst, dann hörst du, wann eine Regel ausgeführt wird. Am besten eignet sich ein kurzer Ton.



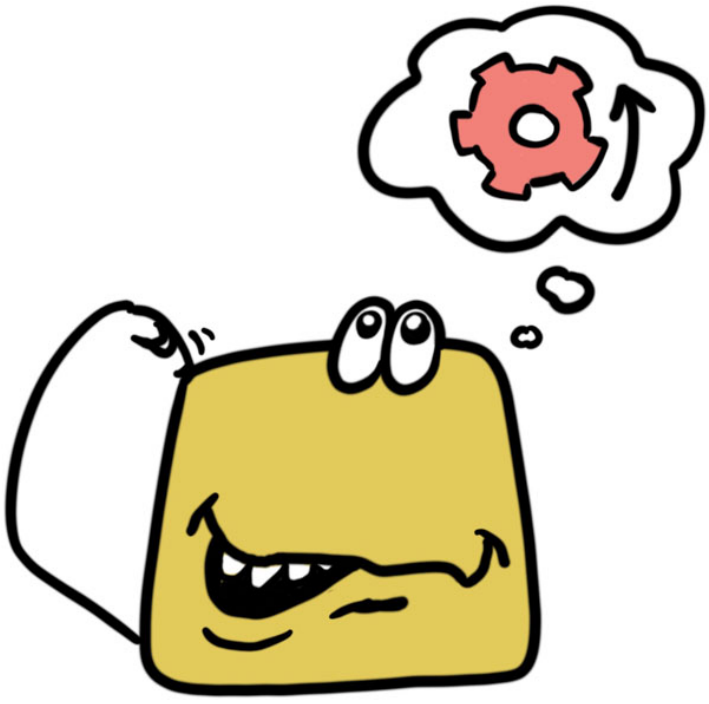
Mit dem Pfeilwerkzeug kannst du Agenten auf dem Spielfeld umher schieben. So kannst du zum Beispiel testen, ob die Tür am Ende des Levels funktioniert, ohne dass du das ganze Level durchspielen musst.

Informatik-Fakten

Fehlersuche heisst auf Englisch „Debugging“, also „Ent-Käfern“. Die Idee ist, dass der Code von schädlichen Käfern heingesucht wird, welche man einen nach dem anderen finden und entfernen muss. Das Wort ist auch auf deutsch üblich.

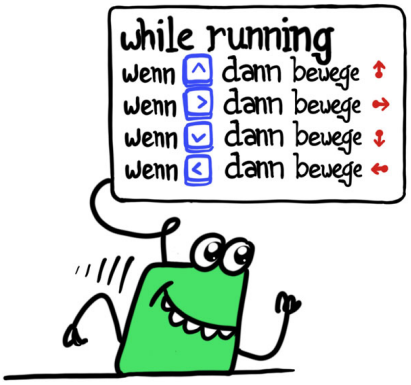


Mittleres Niveau



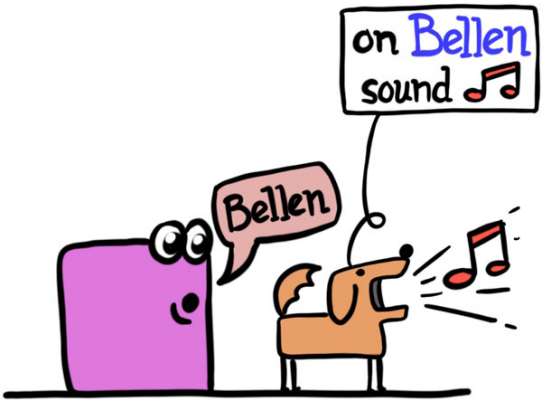
Verhalten gliedern mit Methoden

Methoden gliedern das Verhalten von Agenten. Methoden fassen Regeln zusammen, die unter bestimmten Bedingungen ausgeführt werden.



Die häufigste Methode ist „while-running“. Sie wird automatisch ausgeführt, wenn das Spiel läuft. Hier werden die grundlegenden Verhaltensweise wie die Fortbewegung programmiert.

Mit der „on“-Methode können Methoden selber benannt werden. Diese Methoden können dann vom Programm aufgerufen werden.



Informatik-Fakten

„while running“ ist eine so genannter **Schleife**. Schleifen sind eines der grundlegenden Ideen beim Programmieren: Eine Anzahl Anweisungen wird immer wieder ausgeführt. Dafür gibt es eine Reihe von Anweisungen. Kannst du erraten, was die folgenden Schleifen machen?

```
while read(Zeichen) {
        Text = Text + Zeichen
}
print Text
```

Übersetzung:
while: während
read: lies
print: drucke/schreibe

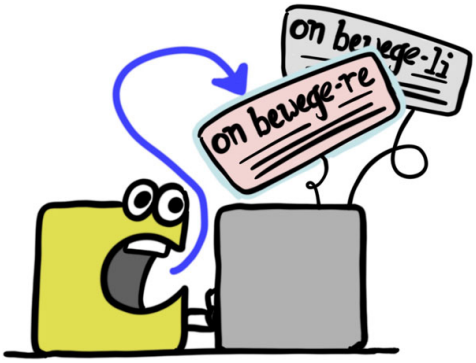
```
while t>5 {
        t = t / 2
}
print t
```

Übersetzung:
while: während
print: drucke/schreibe

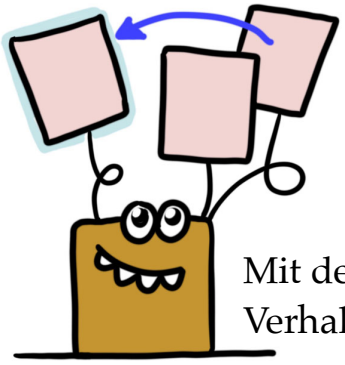
```
for t=1 to 10
        print „hallo!“
next
```

Übersetzung:
for: für
print: drucke/schreibe
next: nächste(s)

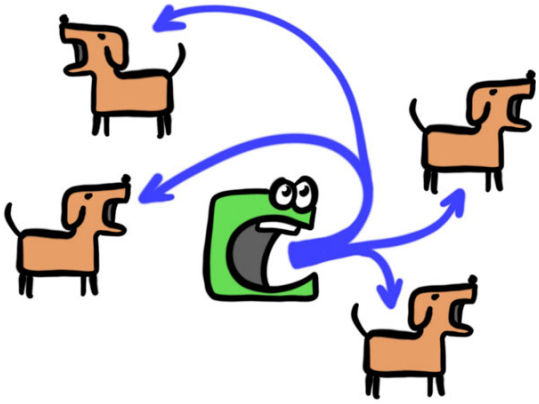
Verhalten auslösen mit Nachrichten



Der „Nachricht“ Befehl ruft ein bestimmtes Verhalten in einem benachbarten Agenten auf.



Mit dem „Nachricht“ Befehl kann auch ein anderes Verhalten im gleichen Agenten aufgerufen werden.



Der „übertrage an alle“ Befehl ruft ein bestimmtes Verhalten in allen Agenten eines bestimmten Typs auf.

Informatik-Fakten

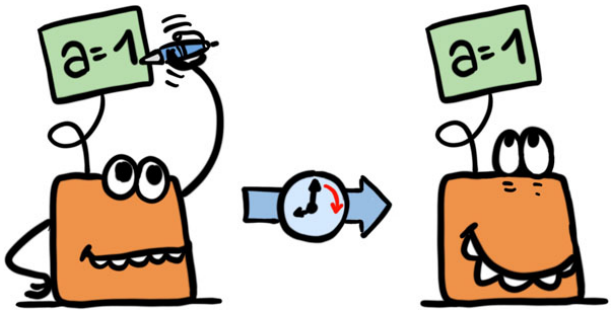
In den meisten Programmiersprachen spricht man von „Funktionen“, welche aufgerufen werden. Eine Funktion kann von überall aus dem Programm heraus aufgerufen werden:

```
b = verdopple(2)
print b

function verdopple(a)
        a = a * 2
        return a
```

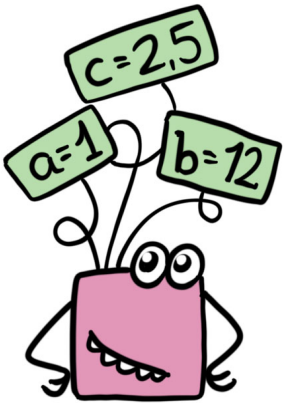
Wenn man viele Funktionen hat, wird das unübersichtlich. Moderne Sprachen wie Java verwenden deshalb so genannte Klassen, damit man die Übersicht behält (rechts, das Programm ist etwas vereinfacht):

```
b = Mathe.verdopple(2)
print b

class Mathe {

        verdopple(a) {
                a = a * 2;
                return a;
        }

        verdreifache(a) {
                a = a*3;
                return(a)
        }
}
```

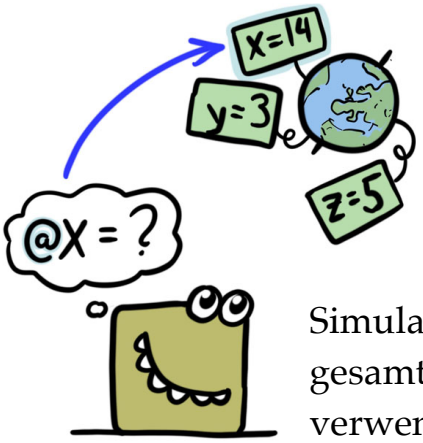
Attribute und Simulationseigenschaften



Attribute und Simulationseigenschaften speichern Werte, auf die später zurückgegriffen werden kann.



Attribute speichern Werte im Agenten, zum Beispiel die aktuelle Laufrichtung oder eine Anzahl Lebenspunkte.



Simulationseigenschaften speichern Werte, die für die gesamte Spielwelt gelten. Wenn man auf sie zugreift, verwendet man das @ Zeichen.

Informatik-Fakten

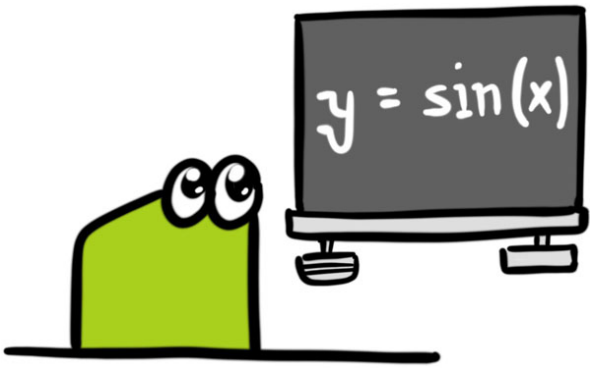
Die Dinge, die Werte speichern, nennt man in der Informatik **Variablen**. Eine Simulationseigenschaft wäre eine so genannte **globale Variable**, da sie von überall her gesetzt und gelesen werden kann (Globus bedeutet „Welt“). Ein Attribut wäre eine so genannte **lokale Variable**, da sie nur an einem Ort gilt (lokal bedeutet örtlich).

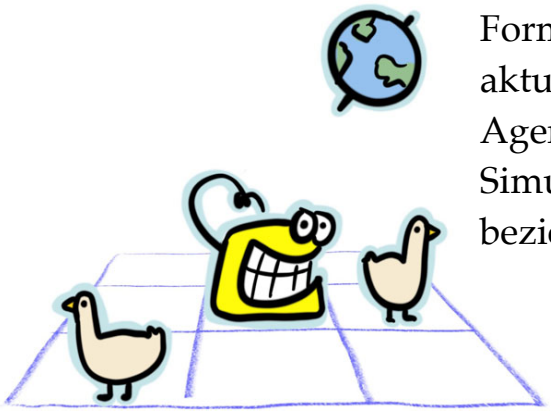
In vielen Programmiersprachen muss man angeben, welche Arten von Werten man in einer bestimmten Variable speichern will. Wir geben einige Beispiele in der Programmiersprache Java.

```
int a = 0;                  //ganze Zahl
float b = 2.5;              //Zahl mit Kommastellen
char c = „x“;               //ein Zeichen
String s = „Hallo Welt!“;   //Wort oder Zeichenfolge
boolean b = true;           //entweder wahr (true) oder falsch (false)
```

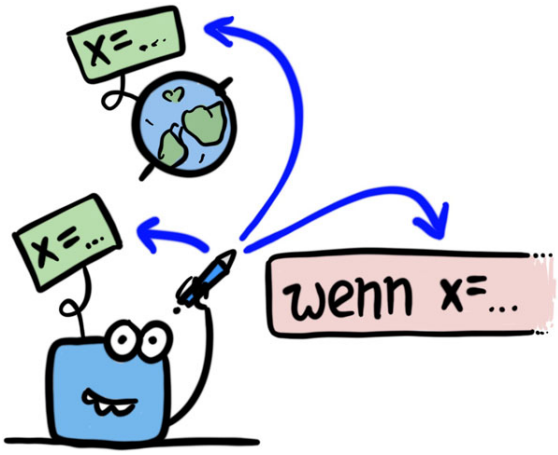
Formeln

Mit Formeln kann ein Verhalten aufgrund von komplexen Bedingungen und mathematischen Funktionen ausgeführt werden.





Formeln können sich auf den aktuellen Agenten, umliegende Agenten und auf Simulationseigenschaften beziehen.

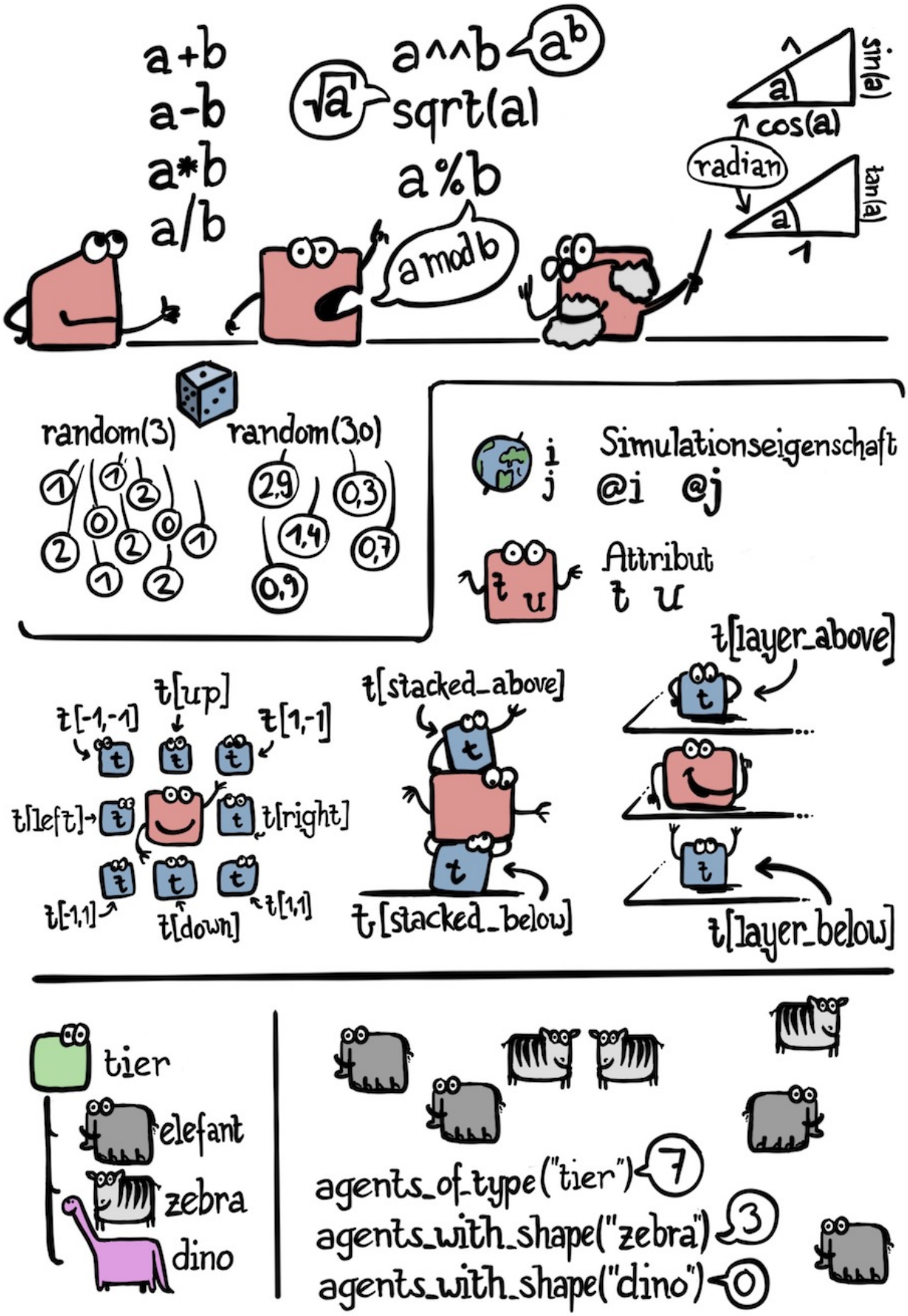


Die Ergebnisse können in Attributen und Simulationseigenschaften gespeichert werden, oder zur Steuerung von Verhalten verwendet werden.

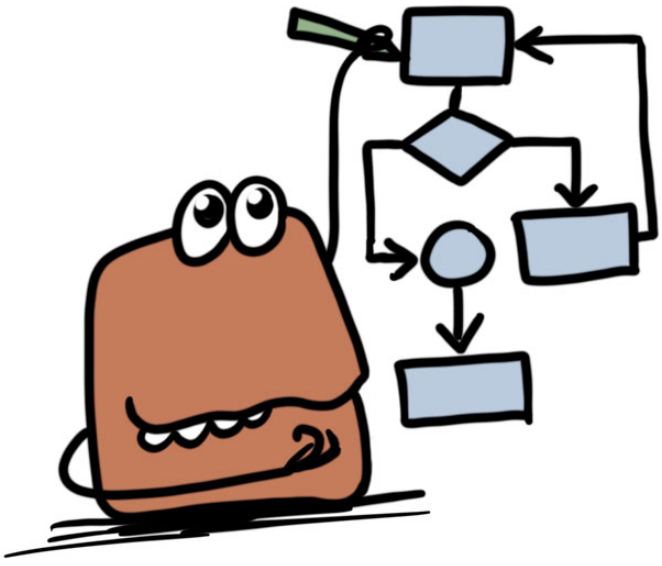
Informatik-Fakten

Das Wort „Computer“ stammt aus dem Englischen und bedeutet „Rechner“. Ursprünglich waren Computer einfach Rechenmaschinen. Sie konnten gar nichts anderes als Formeln ausrechnen. Bevor es heutige Computer gab, bezeichnete man mit „Computer“ oder „Rechner“ auch Menschen, die Rechenaufgaben ausführten. Das war ein eigener Beruf, beispielsweise um Kalenderdaten, Tabellen von mathematischen Werten oder die Flugbahn von Raketen zu berechnen.

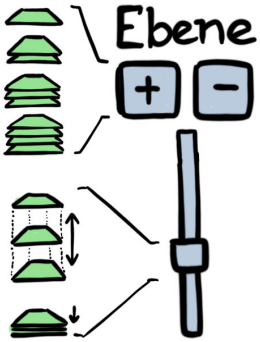
Übersicht: Alles rund um Formeln



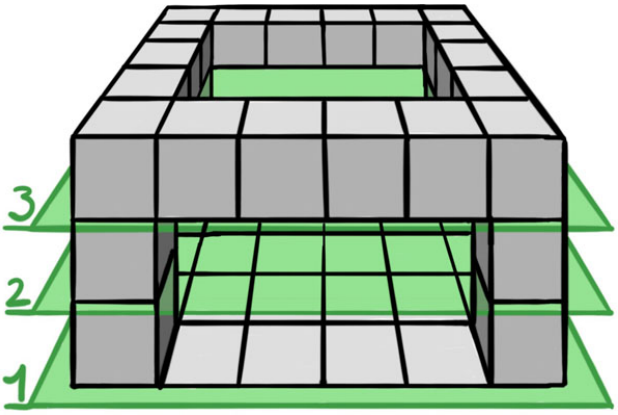
Fortgeschrittene Techniken



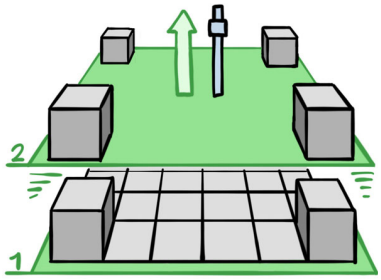
Arbeiten mit Ebenen



Mit dem Ebenenmenü kann man neue Ebenen zur Spielwelt hinzufügen und den Abstand zwischen den Ebenen festlegen.



Mit Ebenen kann man vollständig dreidimensionale Spielwelten erschaffen. Eine Konstruktion wie abgebildet könnte man mit Stapeln nicht umsetzen, da sie frei schwebende Elemente enthält.



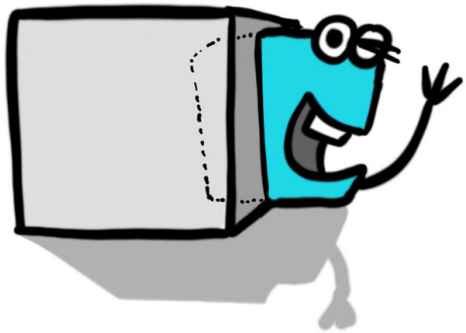
Zum Bearbeiten von Ebenen kannst du die Entfernung zwischen den Ebenen vergrössern. So kannst du die Ebenen getrennt bearbeiten.

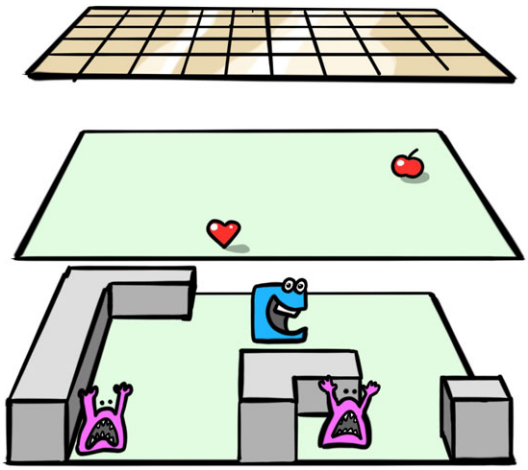
Viele Befehle haben ein Aufklappmenü. Dort kannst du angeben, auf welche Ebene sich der Befehl bezieht. Die Angabe ist immer relativ zur aktuellen Ebene. 0 ist die gleiche Ebene wie der Agent, 1 ist eine Ebene über dem Agenten, und so weiter.

Die Spielwelt gliedern mit Ebenen

Wenn du den Abstand zwischen den Ebenen auf Null setzt, dann überlagern sich die Ebenen. Das hat viele nützliche Effekte.

Wenn zwei Agenten aus verschiedenen Ebenen auf der gleichen Position zu liegen kommen, dann durchdringen sie sich. Das kann witzige Effekte hervorrufen und sehr praktisch sein.



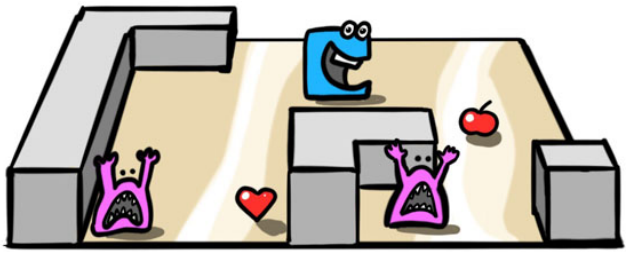


Ein Vorschlag wäre, das Spielfeld wie folgt auf drei Ebenen aufzuteilen:

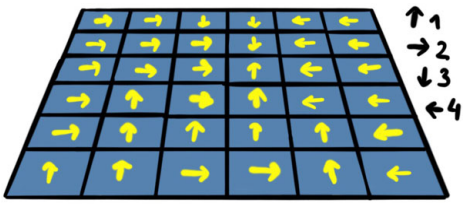
- Ebene 3: Boden
- Ebene 2: Extras wie Bonusfrüchte
- Ebene 1: Spielfeld mit Spieler und Monstern.

Monster und Wände sind auf der untersten Ebene, da diese am sehr einfach bearbeitet werden kann, indem du den Ebeneabstand erhöhst.

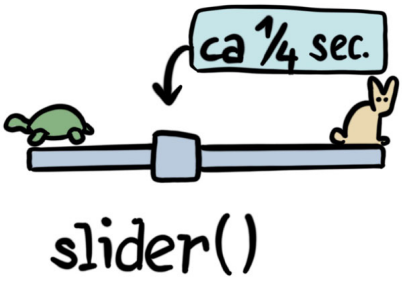
Wenn du jetzt den Abstand zwischen den Ebenen auf 0 setzt, dann hast du ein kompaktes Spielfeld. Die Monster fahren durch die Früchte hindurch statt über sie hinweg, da sie auf anderen Ebenen liegen, und die Programmierung wird einfacher, da innerhalb einer Ebene alles sehr übersichtlich ist.



Wenn du Logik für Bewegung in die Bodenplatten packst, dann am besten so, dass der Boden gleich ausrechnet, in welche Richtung es zur Spielfigur oder zum Ausgang geht. Ein Monster muss dann nur auf der Bodenebene nachschauen, welche Richtung es einschlagen soll. Die Richtungen können mit Zahlen von 1-4 codiert werden.

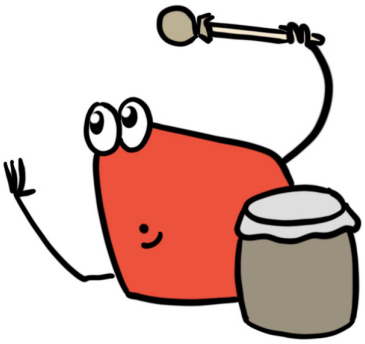


Animationen



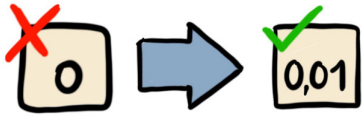
AgentCubes führt die Animationen in einem einheitlichen Rhythmus aus. Die Geschwindigkeit des Rhythmus kann mit dem Schieberegler oben eingestellt werden (Standard: 1/4 Sekunde). Der Wert kann mit der Variable slider() abgefragt werden und ist die Standardeinstellung für alle Animationen.

AgentCubes wartet immer, bis alle Animationen fertig ausgeführt sind, bevor das Programm weiter ausgeführt wird. Du kannst also nicht über die Animationen einen Agenten etwas schneller bewegen als einen anderen. Der schnelle Agent wartet immer, bis der langsame fertig ist mit seiner Bewegung.



Anstatt die Animationen über den Schieberegler einzustellen, kannst du auch eine Simulationseigenschaft verwenden. So kannst du ein Spiel programmieren, das immer schneller wird.

Für Fortbewegung interpretiert AgentCubes den Wert 0 als „slider()“. Wenn du schnelle Bewegungen umsetzen willst, dann wähle statt 0 einfach einen sehr kleinen Wert.

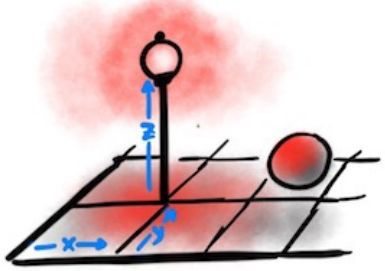


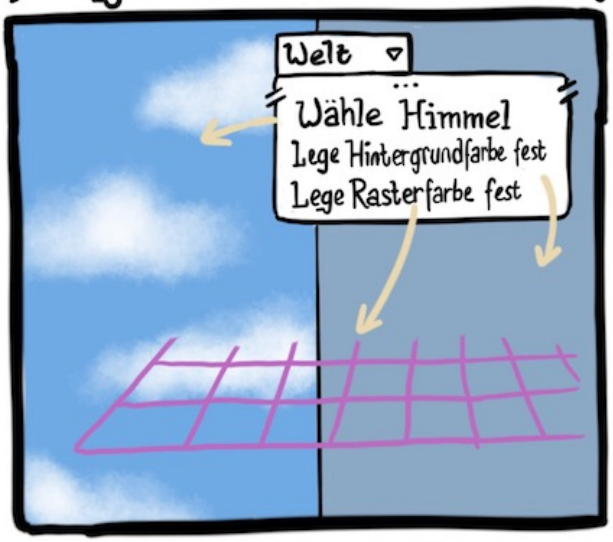
Animation und Verzierungen

von Michael Mittag

zeige Lichtfenster

x	y	z	Rot	Grün	Blau
1	1	2	255	0	0





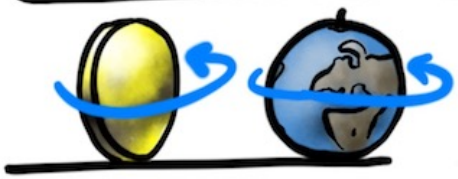
Bewegung

→ Verwende die gleiche Zeitdauer für alle Bewegungen, zB slider()
→ Wenn du für eine Bewegung länger angibst, wartet AgentCubes. Das Programm stockt dann
→ Bewegungen können konstant (gehen, rollen) oder beschleunigt (springen, fallen) ablaufen

bewege →
drehe um 0 0 45
Zeit slider() sek

drehe um 45 0 0
Zeit slider() sek

drehe um 0 -90 0
Zeit slider()



Hin & her:

einmal alle 2*slider() sek → drehe um 0 0 15
einmal alle slider() sek → drehe um 0 0 -15

lösche •
Zeit slider()

Animation

Tim
stehen
springen

sehe • → wechsle •
sehe • → wechsle •

Achtung: Während ein Agent sich bewegt, stoppt die Animation

Startbildschirm

Am besten verwendest du hierfür eigene Agenten

Formen effizient einsetzen

Mit Formen kannst du Agenten mit überschneidenden Verhaltensweisen gruppieren. So musst du die Verhaltensweisen, welche für alle Agenten gelten, nur ein Mal programmieren.

Mit der sehe-Bedingung kannst du definieren, dass bestimmte Befehle nur dann ausgeführt werden, wenn der Agent eine bestimmte Form hat.

Wenn du zusätzlich Nachrichten verwendest, dann kannst du Methoden definieren, die nur bei bestimmten Formen zum Einsatz kommen.

Informatik-Fakten

Die Informatik hat sich lange damit beschäftigt, wie man es vermeidet, dass der gleiche Programmcode an verschiedenen Stellen Verwendung findet. Das moderne Konzept dafür ist die so genannte **Vererbung**. Dabei erbt ein Programmteil den Code eines anderen Teils und kann dann eigenen Code hinzufügen oder auch Teile des geerbten Codes überschreiben.

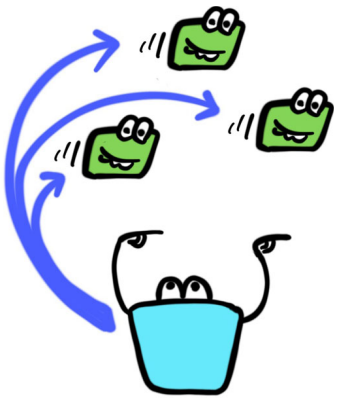
Der (leicht vereinfachte) Code rechts definiert, dass eine Katze jagen kann und miaut. Die Klasse „Löwe“ erbt dieses Verhalten, ändert aber das Miauen in ein Brüllen.

Auf diese Weise kann Programmcode mehrfach genutzt werden. Ich muss also für den Löwen nicht noch einmal neu hinschreiben, wie das Jagen funktioniert.

```
class Katze {

    jagen() {
        //Befehle für das Jagen
    }

    laut() {
        spieleTon(„miau.mp3“)
    }
}

class Löwe extends Cat {

    laut() {
        spieleTon(„roar.mp3“)
    }
}
```

Übersetzung:
class: Klasse
extends: erweitert

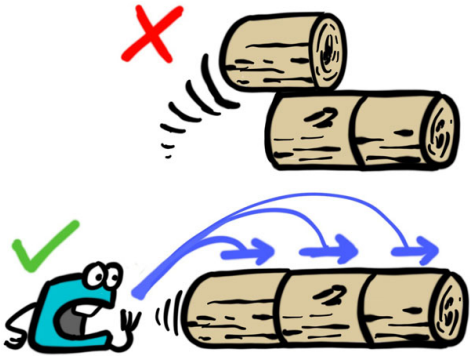
Verhalten mit einem zentralen Controller steuern



Üblicherweise programmiert man das grundlegende Verhalten eines Agenten in die **while running** Methode. Damit wird es von AgentCubes automatisch aufgerufen.

Wenn du mehr Kontrolle brauchst, dann kannst du das Verhalten in eine eigene Methode packen, zum Beispiel **on Bewegung**, und diese von einem zentralen Controller aus aufrufen.

Der Controller ist ein (meist unauffällig auf dem Spielfeld platzierter) Agent, der nichts anderes tut, als das Verhalten von anderen Agenten geordnet aufzurufen. Meist geschieht dies über den **übertragen an alle** Befehl.

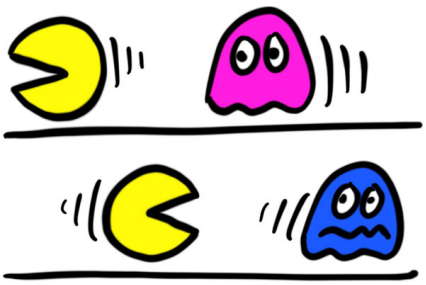


Manchmal brauchst du Controller, wenn du Animationen synchronisieren musst. Wenn du zusammengesetzte Objekte bewegst, dann stapelt sie AgentCubes manchmal aufeinander (oben), wenn es die hinteren Elemente zuerst bewegt. Der Controller kann sicherstellen, dass alle Elemente gleichzeitig bewegt werden.

Controller werden häufig bei Simulationen eingesetzt. Dort werden zunächst bei allen Agenten die Wahrnehmungen ausgelöst, dann in einem zweiten Schritt die Handlungen. So wird sicher gestellt, dass sich die Handlungen auf den Zustand zu Anfang der Runde beziehen und nicht auf die gerade erfolgten Handlungen anderer Agenten.

Methoden verschachteln

Bei Pac Man haben die Geister unterschiedliches Verhalten, je nach dem, ob Pac Man eine Kraftpille gefressen hat oder nicht.



Wenn Verhalten von geschachtelten Bedingungen abhängig ist, dann kann man es mit verschachtelten Methoden abbilden.

Dazu programmiert man zwei Methoden für die unterschiedlichen Verhaltensweisen. Im Hauptprogramm ruft man dann je nach Zustand die eine oder andere Methode auf.

Mit verschachtelten Methoden kann man oft erreichen, dass Code sehr viel weniger kompliziert und besser gegliedert ist, als wenn ich alles in Bedingungen abbilde.

Informatik-Fakten

Das Verschachteln von Bedingungen kommt in den Informatik zwar häufig vor, ist aber nicht gerne gesehen. Unten siehst du ein verschachteltes Programm:

```
if stark=0
        if richtung_zu_pacman = „rechts“
                if leer(rechts)
                        bewege(rechts)
        else if richtung_zu_pacman = „links“
                if leer(links)
                        bewege(links)
else
        if richtung_zu_pacman = „rechts“
                if leer(links)
                        bewege(links)
        else if richtung_zu_pacman = „links“
                if leer(rechts)
                        bewege(rechts)
```

Meist versucht man, das auseinanderzunehmen. Damit landet man bei dem Aufbau, den du in AgentCubes ohnehin machst, da AgentCubes keine verschachtelten Bedingungen unterstützt.

```
if stark=0
        verfolgen()
else
        weglaufen()

function verfolgen()
        if richtung_zu_pacman = „rechts“
                if leer(rechts)
                        bewege(rechts)
        else if richtung_zu_pacman = „links“
                if leer(links)
                        bewege(links)

function weglaufen() { ...
```

Antworten zu den Fragen

s.5: Warum kann Donkey Kong mehr Sprites darstellen als andere Geräte seiner Zeit?

Donkey Kong war ein Spielhallengerät. Diese Geräte wurde speziell gebaut, waren sehr teuer und mussten auch Dinge können, welche deutlich über dem lagen, was man zu Hause hatte. Ein Atari 2600 kostete damals $200, ein Donkey Kong Automat über $4000. Bis zur Playstation (1994) und dem N64 (1996) waren Spielhallenautomaten den Heimkonsolen deutlich überlegen. PCs waren vor allem für Büroanwendungen ausgelegt und konnten weder mit den Konsolen noch mit den Spielhallengeräten mithalten. Der GameBoy sollte klein und preiswert sein, weshalb er nur das Minimum konnte, um Spiele darzustellen (die ersten Modelle hatten einen Schwarz/Weiss Bildschirm).

s.6: Warum adressiert man heute nicht mehr direkt?

Heute müssen auf PCs mehrere Programme gleichzeitig laufen können. Das können sichtbare Programme sein wie Word und Excel, aber auch unsichtbare, wie das Programm, das die Mausbewegungen an den Computer überträgt oder im Hintergrund gelegentlich prüft, ob ein Programm ein Update benötigt. Diese Programme können sich nicht darauf verlassen, dass bestimmter Speicher ausschliesslich von ihnen verwendet wird. Deshalb wird die Speicherzuteilung jeweils vom Betriebssystem vorgenommen und die Programme sind so geschrieben, dass sie an beliebigen Speicherstellen laufen.

s.10: Was machen die drei Schleifen?

a) So lange es Zeichen hat, die eingelesen werden können, werden sie einem Text hinzugefügt. Dann wird der ganze Text auf dem Bildschirm ausgegeben.

Diese Schleife ist typisch für Programme, welche Daten von einem Medium einlesen, wie zum Beispiel der Festplatte. Die Daten werden Stück für Stück von der Festplatte gelesen, an das Programm übermittelt und dort zusammengefügt.

b) Die Zahl, die in der Variable t gespeichert ist, wird so lange halbiert, bis sie kleiner als 5 ist. Dann wird sie ausgegeben.

c) Das Programm schreibt 10 Mal „Hallo Welt!“ auf den Bildschirm.