

Einführung in die Informationsverarbeitung

Øyvind Eide

Woche 10 Visualisierung

oeide@uni-koeln.de
<http://idh.uni-koeln.de>



Test

- Studienleistung
- Themen: Woche 1–10
- Wiederholung: spät Januar
- Selbstevaluation
- In Hauptgebäude Hörsaal VIII



Visualisierung



Grundgedanke

- (1) Ein Satz von Variablen gibt einen bestimmten Zustand wieder.
- (2) Dieser Zustand wird regelbasiert verändert, wobei eine bestimmte Anzahl von Veränderungen einen „Schritt“, also ein Stadium in der Entwicklung eines Systems wiedergibt.
- (3) Dieser Zustand kann in regelmäßigen Abständen visualisiert werden.



Beispiel: Towers of Hanoi

stack turm1, turm2, turm3;

Ausgangszustand: Alle 3 Scheiben auf turm1;

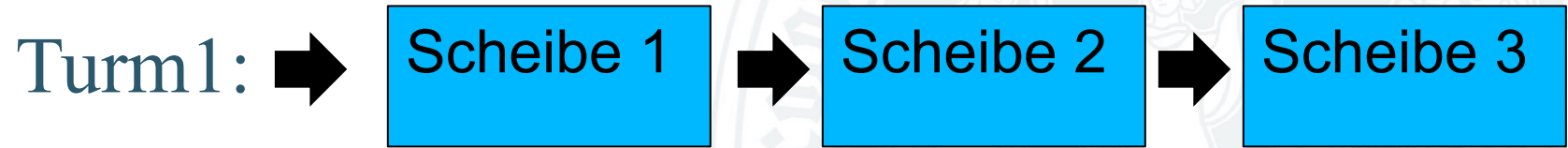
Dann: transport(3, turm1, turm2, turm3);

```
function transport( int n, stack spindel1, stack spindel2, stack spindel3) {  
    if (n >1) transport(n-1,spindel1,spindel3,spindel2);  
    schritt(spindel1,spindel);  
    if (n>1) transport(n-1,spindel3,turm2,spindel1);  
}
```

```
function schritt( stack spindel1, stack spindel2) {  
    spindel2.push(spindel1.pop());  
    visualisiere();  
}
```



Ausgangszustand der Variablen

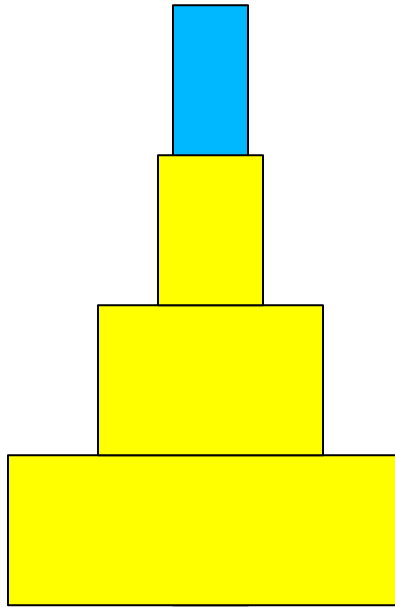


Turm2:

Turm3:



visualisiere();



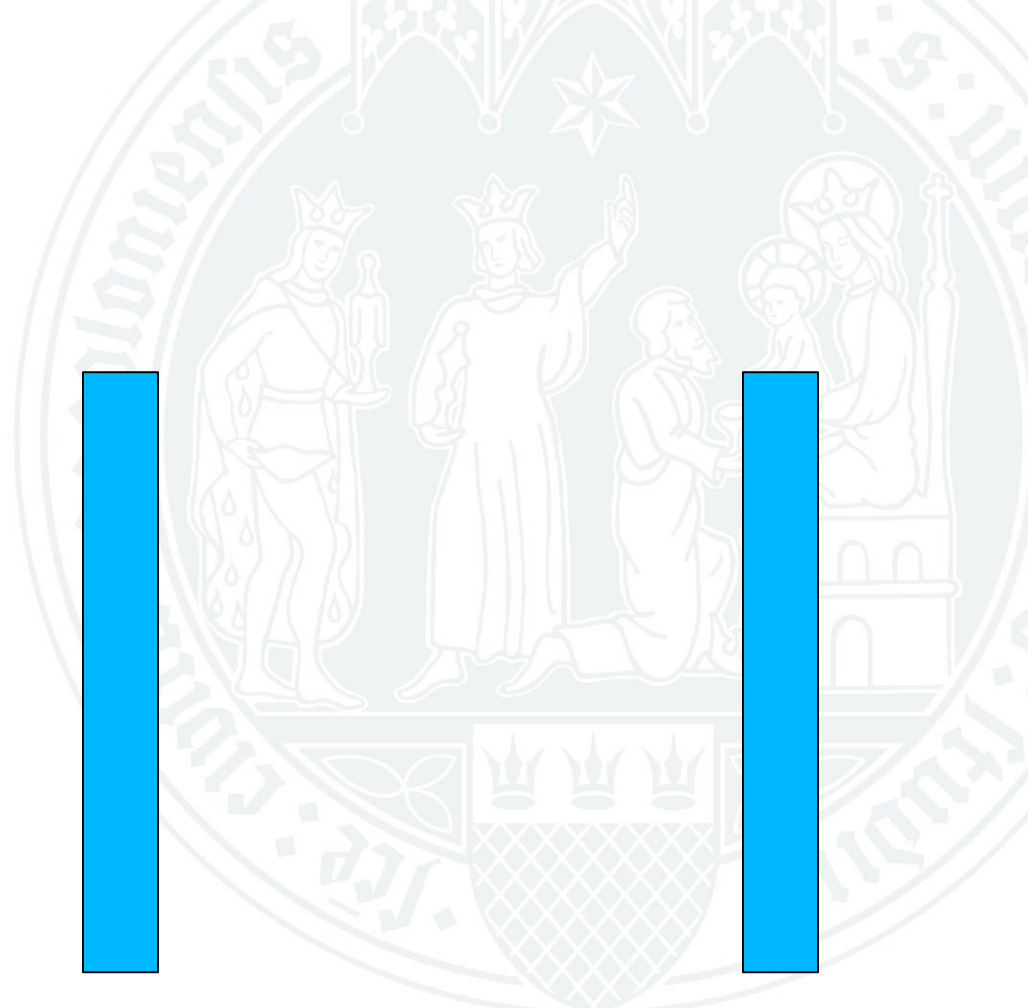
S1



S2



S3



Ausgangszustand der Variablen

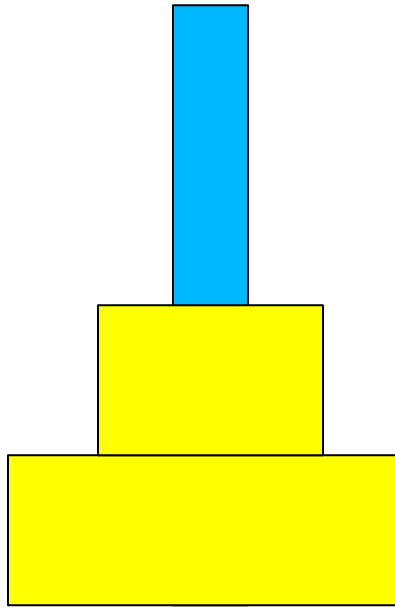
Turm1: → Scheibe 1 → Scheibe 2

Turm2: → Scheibe 3

Turm3:



visualisiere();



S1



S2

S3



Bewegung und Wachstum

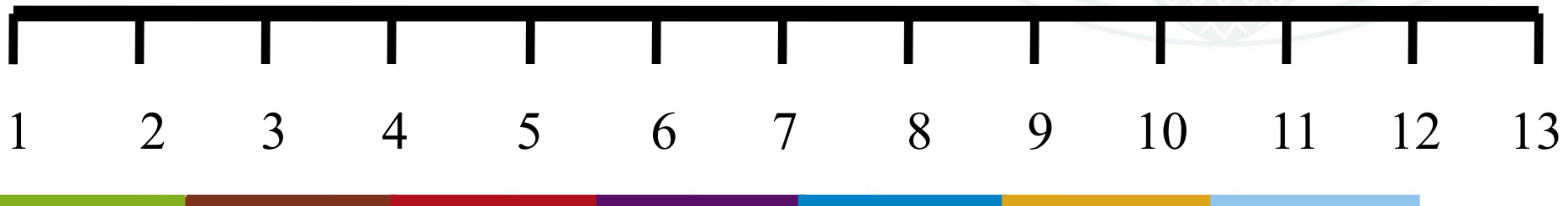
Ausgangszustand:

Position =2; Durchmesser=1;

Schritt:

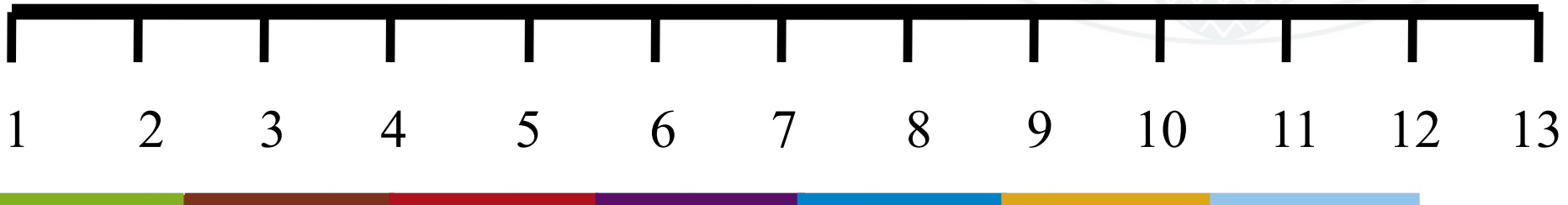
Position=Position+1:

Durchmesser=Durchmesser+Durchmesser *0.20;



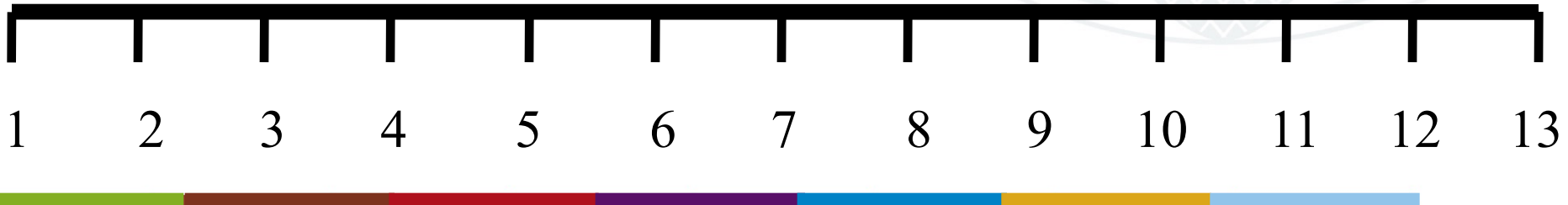
Bewegung und Wachstum

Position = 2; Durchmesser = 1;



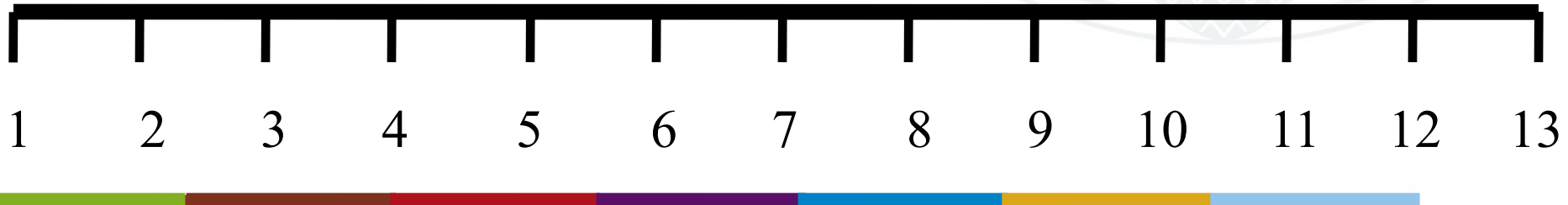
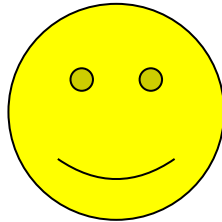
Bewegung und Wachstum

Position = 3; Durchmesser = 1.20;



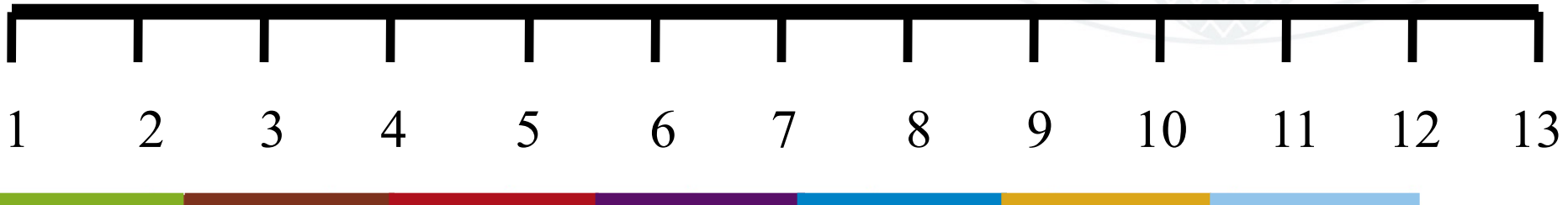
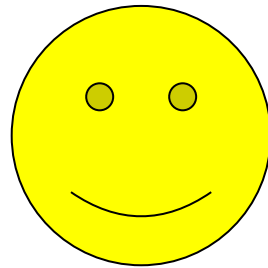
Bewegung und Wachstum

Position = 4; Durchmesser = 1.44;



Bewegung und Wachstum

Position = 5; Durchmesser = 1.728;



Exkurs: Logik der Computergraphik

Trennung zwischen *Graphic Primitives* und *Graphic Context*.

Graphic Primitives:

Draw line.

Draw circle.

Draw text.

Fill area.

[3D equivalents.]



Exkurs: Logik der Computergraphik

Trennung zwischen *Graphic Primitives* und *Graphic Context*.

Graphic Context:

Set line thickness.

Set line type.

Set line color.

Set fill color.

Set text font.

.....



Exkurs: Logik der Computergraphik

Trennung zwischen *Graphic Primitives* und *Graphic Context*.

Die Ausführung identischer Zeicheninstruktionen in geändertem Kontext führt zu unterschiedlichem Aussehen.

Aufrufen derselben Primitives mit unterschiedlichen Parametern wie Position und Durchmesser führt zu unterschiedlicher Zeichnung.



Zufall

Algorithmen sind deterministisch.

Das macht Spiele vorhersehbar und langweilig; Simulationen zu mechanistisch.

Lösung: „Zufallsgenerator“.



Zufall

Definition: Ein mathematisches Verfahren, dass bei n hintereinander erfolgenden Aufrufen n Zahlen in einem Bereich (Min , Max) so errechnet, dass sie zwischen Max und Min mit der Häufigkeit einer bestimmten statistischen Verteilung auftreten.

Oft:

Minimum = 0; Maximum = $2^{32} - 1$

Verteilung: Gleichverteilung



Zufall

Paradoxon: Der die „Zufalls“ zahlen generierende Algorithmus ist selbst deterministisch.

Lösung: Variabler Startwert, der von einer mit dem Zweck des Programms unabhängigen und nicht vorhersehbaren Umweltbedingung abhängt.



Zufall

Ein Bogenschütze soll mit 33 % seiner Schüsse treffen.

Vor jedem Schuss wird eine Zufallszahl von 1 bis 100 gewählt.

Ist sie 33 oder kleiner, trifft der Bogenschütze; sonst geht der Schuss vorbei.



Zweck von Simulationen

Simulationen *müssen* nicht unmittelbar mit Visualisierungen verbunden werden.

Sie können auch dazu dienen, die Performanz von Systemen mit bestimmten Eigenschaften zu testen.



Simulationen → Emulationen

Simulationen von Rechnersystemen auf Rechnersystemen: „Emulationen“.

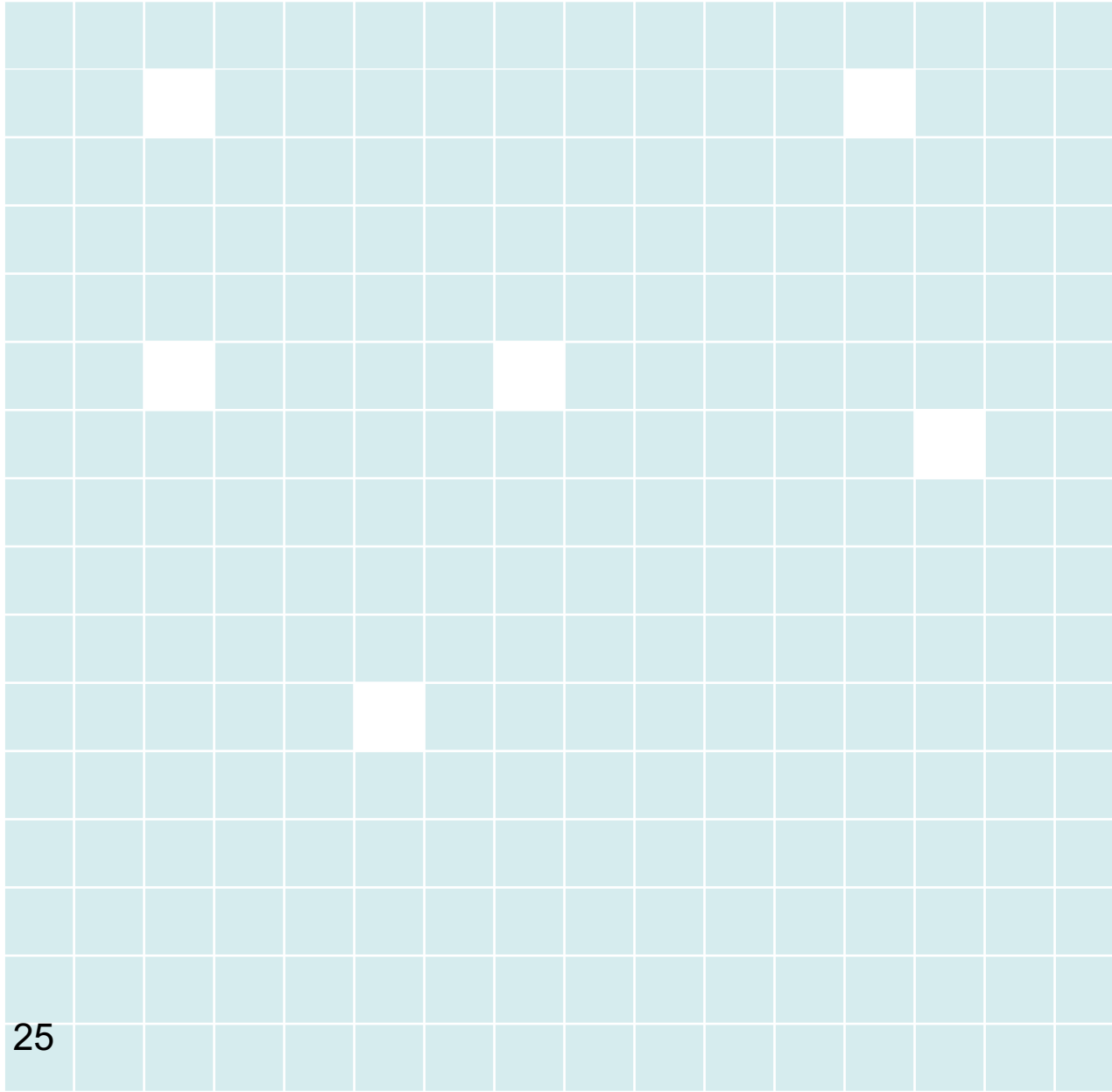
- Smartphone Programmierung
- Langzeitarchivierung



Beispiel: Aufzucht von Trollen

Annahme:
Ein Troll lebt
in einer $n \times n$
Welt.

Beispiel: Aufzucht von Trollen

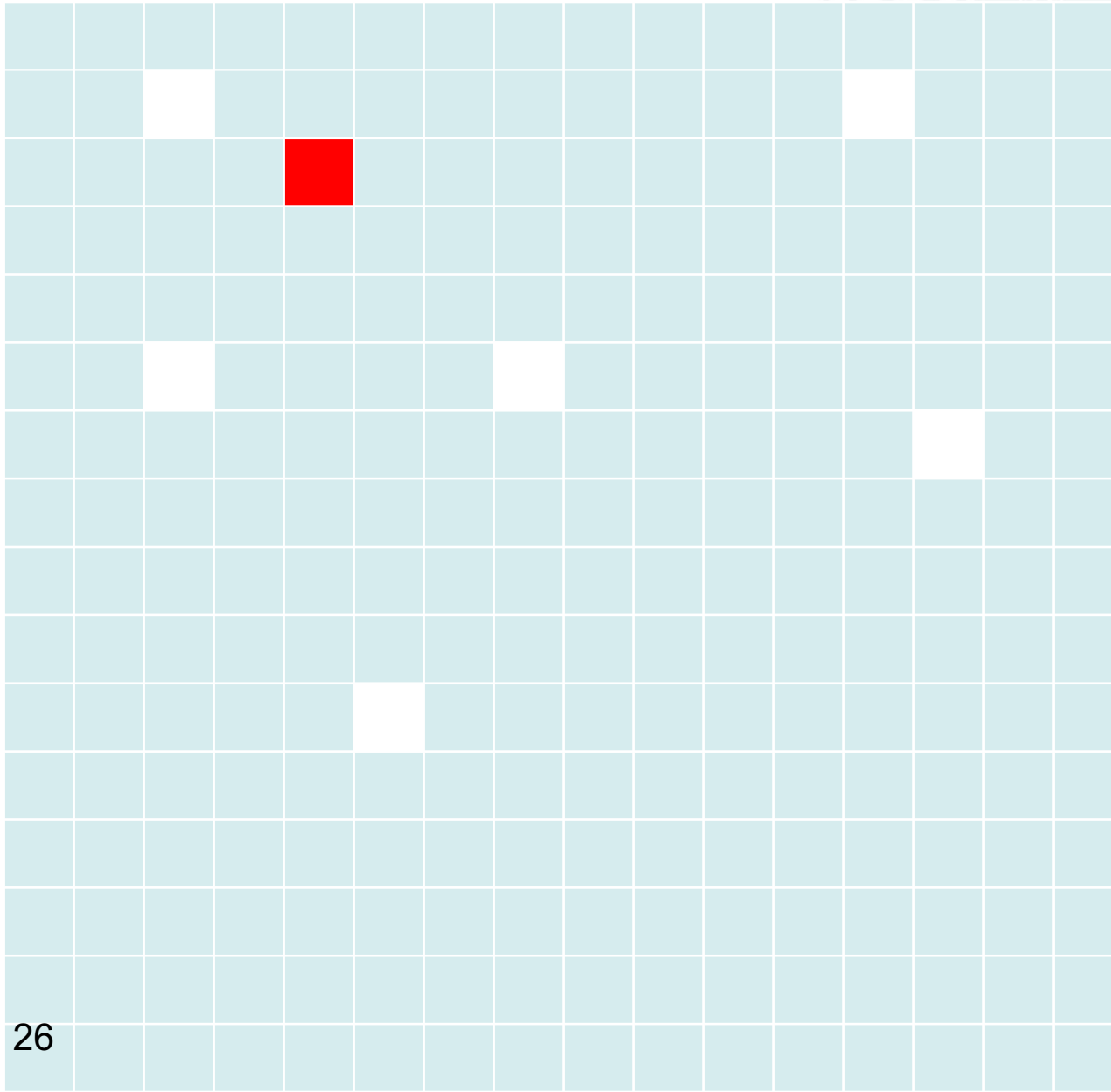


Diese enthält:

■ Schafe.

Schmackhaft
und für das
Überleben
des Trolls
notwendig.

Beispiel: Aufzucht von Trollen

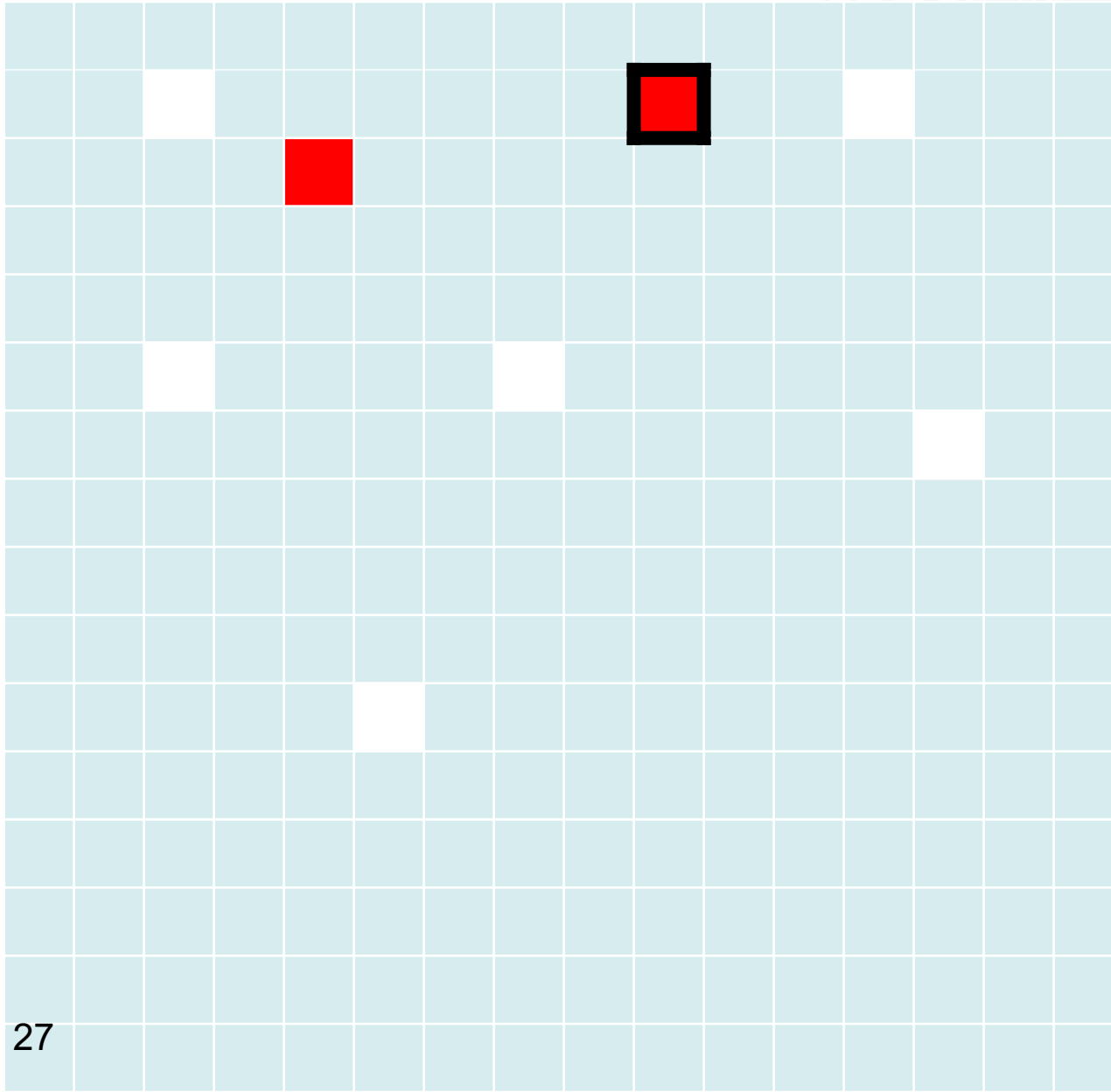


Diese enthält:

 Ritter.

Aggressiv,
gefährlich,
aber sterblich.

Beispiel: Aufzucht von Trollen

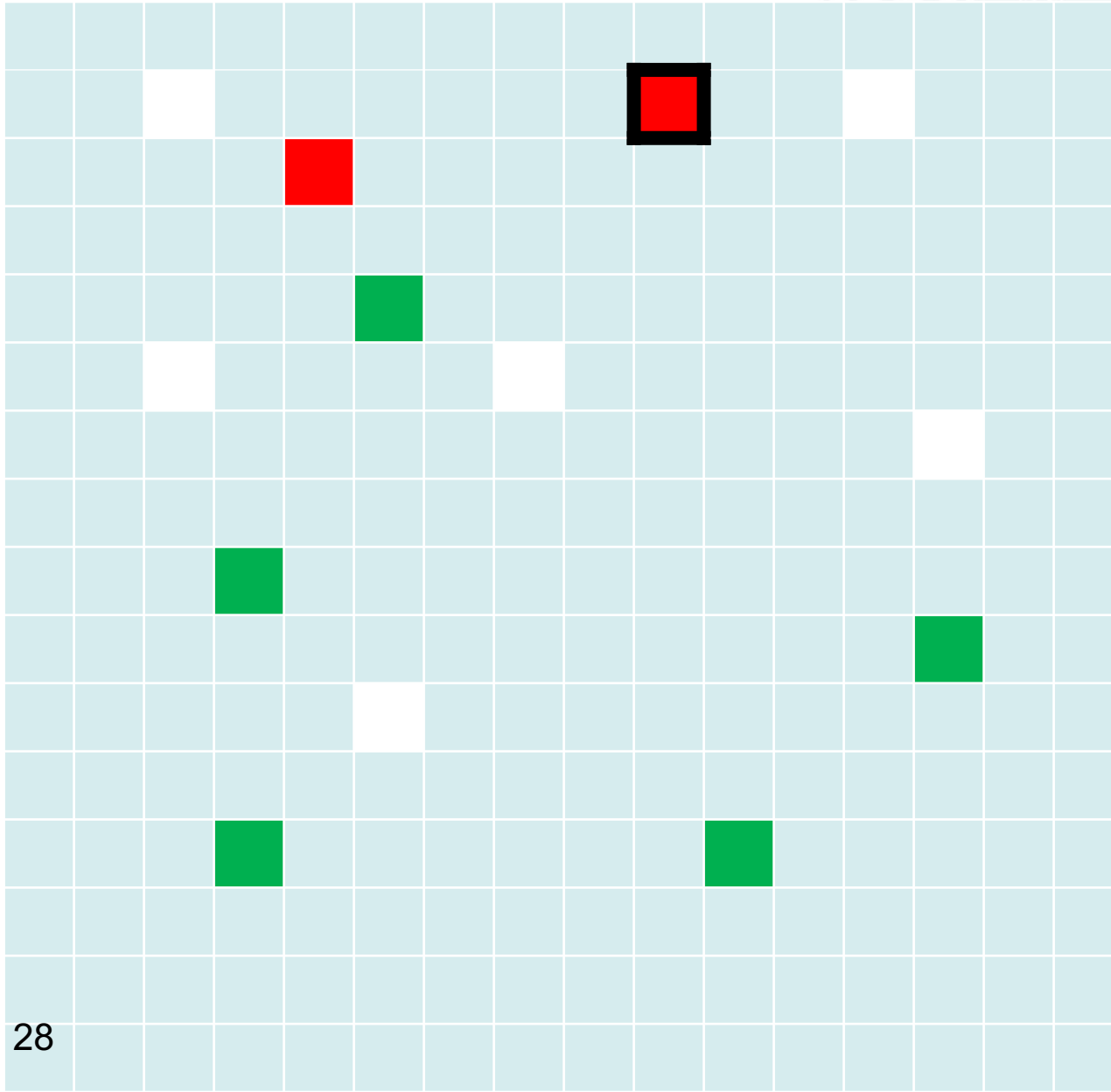


Diese enthält:

 Türme.

Stationär,
gefährlich,
nicht
bekämpfbar.

Beispiel: Aufzucht von Trollen

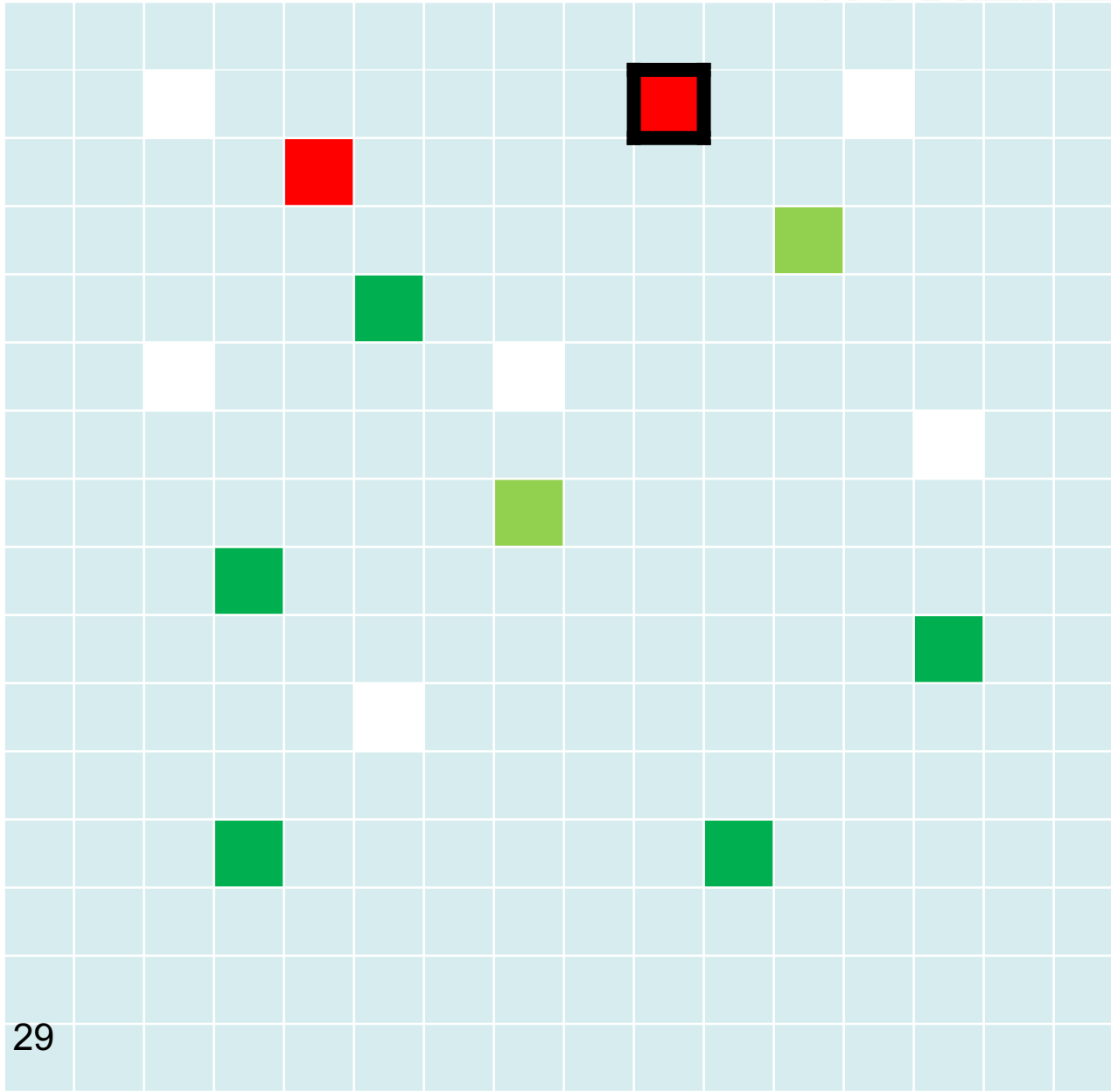


Diese enthält:

 Refugien.

Sicherer
Aufenthalt.
Trolle
„heilen“.

Beispiel: Aufzucht von Trollen



Diese enthält:

 Fallen.

Sehen aus wie
Refugien.

„Fesseln“ den
Troll für eine
gewisse Zeit.
Machen ihn
verwundbarer.

Test eines Trolls:

- Einzelner Troll wird in Welt „geboren“.
- Er muss 500 „Züge“ überleben.
- Bei jedem Zug wird seine Reaktion durch seine „Gene“ gesteuert.
- Am Ende wird der Erfolg seines Lebens bewertet:

```
double Troll::GetEvaluation() {  
    double score = 8.0 * KnightsKilled +  
    10.0 * SheepEaten +  
    1.5 * TurnsSurvived -  
    1.0 * TurnsCaptive -  
    2.5 * DamageTaken;  
    return score; }  
}
```



Test eines Trolls: „Gene“.

Beispiel: Ein Troll steht genau zwischen einem Ritter, einem Schaf und einem Refugium.

Er habe drei Gene:

{Sicherheitsstreben, Hunger, Aggressivität}

Der Troll { 1.0, 0.5, 0.5} läuft ins Refugium.

Der Troll { 0.5, 1.0, 0.5 } frisst das Schaf.

Der Troll { 0.5, 0.5, 1.0 } attackiert den Ritter.



Tatsächliche Gene

1. "Hunger" oder: Zuneigung zu Schafen.
2. "Aggressivität" oder: Abneigung gegen Ritter.
3. "Gesundheit" oder: Geschwindigkeit der Heilung.
4. "Fluchtbereitschaft" oder: Misstrauen gegen potentielle Gefahren.
5. "Neugier" oder: Bereitschaft, in neue Teile des Spielfeldes zu ziehen, auch wenn kein unmittelbarer Anreiz in unmittelbarer Umgebung ist.



Testanordnung

1. Am Anfang werden 100 Trolle, anfänglich mit vorgegebenen „Genen“, in 100 unterschiedlichen Welten ausgesetzt.
2. Diese Welten repräsentieren unterschiedliche Umwelttypen: „schafreich“, „ritterreich“, „bevölkerungsarm“.
3. Am Ende wird der „Lebenserfolg“ jedes Trolls gemessen.
4. Und die überlebenden Trolle zeugen eine neue Generation von 100 weiteren Trollen ...
5. ... die bei Schritt 1 fortsetzen.



Die Partnerwahl des Trolls

- Top 20 Clonen. (= 20 junge Trolle)
- Das erfolgreichste Drittel paart sich. Je erfolgreicher ein Troll, desto öfter paart er sich. (= 70 junge Trolle)
- Einige wenige neu (mit Zufallsgenen) erzeugen. (= 10 junge Trolle)



Wie paaren sich Trolle?

Troll-Elter 1: { 1.0, 0.8, 0.8, 0.2, 1.0 }

Troll-Elter 2: { 0.8, 0.6, 0.8, 0.4, 0.9 }

Troll- Kind: { 0.9, 0.7, 0.8, 0.3, 0.95 }



... oder auch so:

Troll-Elter 1: { 1.0, 0.8, 0.8, 0.2, 1.0 }

Troll-Elter 2: { 0.8, 0.6, 0.8, 0.4, 0.9 }

Troll- Kind: { 1.0, 0.6, 0.8, 0.4, 1.0 }



Wozu macht man sowas?

Es ist auf andere Weise kaum feststellbar, welche Eigenschaften einen Troll in einem Spiel zu einem Gegner mit wechselnden, aber in etwa vorhersehbaren Eigenschaften machen. (Spielgrad für einen Ritter „leicht“, „mittel“, „schwierig“).

Das Prinzip hat zahlreiche Anwendungen außerhalb der Spieleentwicklung.



Wozu macht man sowas?

Ingenieurwissenschaften: Testen des Verhaltens von komplexen technischen Komponenten.

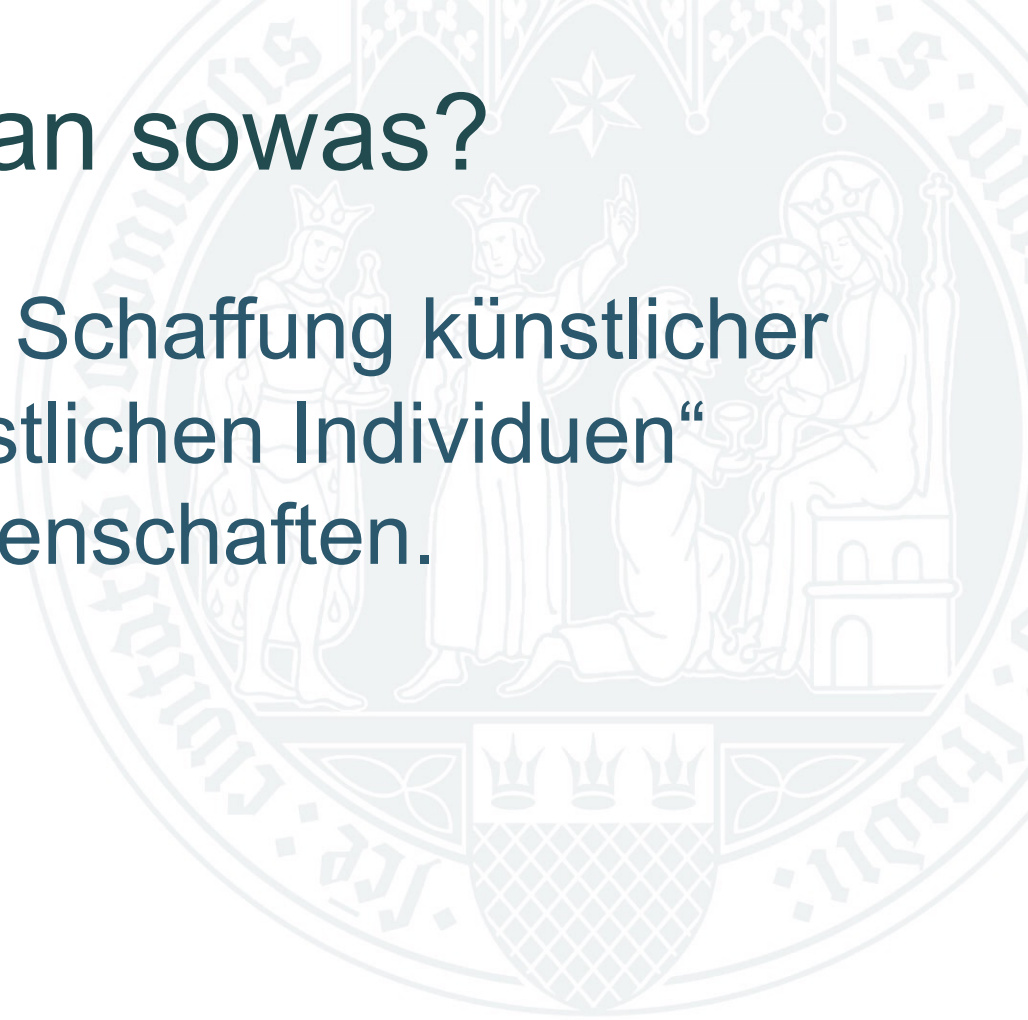
Biowissenschaften: Testen von Annahmen über genetische Gesetzmäßigkeiten / Interaktionen zwischen vererbten Eigenschaften und der Umwelt / Reaktion auf Umweltveränderungen.

Geistes- / Kulturwissenschaften: Sind wir in der Lage Entwicklungsprozesse zu modellieren, haben wir sie verstanden.



Wozu macht man sowas?

Interaktive Medien: Schaffung künstlicher Umwelten mit „künstlichen Individuen“ vorhersagbarer Eigenschaften.



Back to reality

„Genetische Algorithmen“ haben gezeigt dass die Rolle von Mutationen in der Evolution erheblich überschätzt wurde; „crossover“, i.e., die Neukombination von Genen, wahrscheinlich wesentlich einflussreicher.

