Bundeswettbewerb Informatik: Aufgabe 2

Lösungsidee

Die Anzahl der Robuttons, Münzen und die Tischform sind entscheidend für den Ausgang der Simulation. Im günstigsten Fall schieben die Robuttons die Münzen zu Ansammlungen zusammen.

Voraussetzungen

Es wird mindestens ein Robutton und eine Münze auf dem Tisch platziert.

Die Münzen und Robuttons werden zu Beginn der Simulation zufällig auf der Tischfläche platziert. Die Platzierung der Münzen und Robuttons nimmt Einfluss auf den Verlauf der Simulation.

Die Robuttons haben zu Beginn der Simulation eine zufällige Richtung.

Anzahl der Münzen und Robuttons

Folgende Fälle können unterschieden werden:

Die Anzahl der Münzen ist sehr viel kleiner als die Anzahl der Robuttons:

$$\text{Anzahl Münzen} \ll \text{Anzahl Robuttons}$$

In diesem Fall ist es sehr wahrscheinlich, dass die Robuttons alle Münzen aufnehmen und nie wieder ablegen, da keine Münzen mehr auf dem Tisch sind. Die gleichen Ergebnisse gibt es bei nur einer Münze.

Die Anzahl der Münzen ist ungefähr oder genau gleich der Anzahl der Robuttons:

$$\text{Anzahl Münzen} \simeq \text{Anzahl Robuttons}$$

Die Robuttons tragen die Münzen zu mehreren Ansammlungen zusammen. Abhängig von der Tischform und der Zeit kann es sogar dazu kommen, dass alle Münzen zu einer großen Ansammlung von Münzen zusammengetragen werden. Ansonsten ist die Position der entstandenen Ansammlung zufällig.

Die Anzahl der Münzen ist sehr viel größer als die Anzahl der Robuttons:

$$\text{Anzahl Münzen} \gg \text{Anzahl Robuttons}$$

Die Robuttons tragen die Münzen zu Ansammlungen zusammen. Es dauert jedoch viel länger, bis größere Ansammlungen entstehen.

Es gibt genau einen Robutton:

$$\text{Anzahl der Robuttons} = 1$$

Nach einer gewissen Zeit bewegt sich der Robutton zwischen einer oder zwei Ansammlungen von Münzen hin und her. Er pendelt von einer Ansammlung von Münzen zur nächsten und ändert seinen Weg nicht mehr.

Fazit:

Der einzig interessante Fall ist der Zweite (Anzahl der Robuttons entspricht ungefähr der Anzahl der Münzen), denn dort erfüllen die Robuttons einen Zweck: Die Umlagerung der Münzen zu einer oder mehreren Ansammlungen.

Tischgröße

Die Dichte der Robuttons und Münzen darf nicht zu groß sein. Die Anzahl der Münzen und Robuttons sollten demnach der Tischgröße angepasst sein. Den Robuttons sollte es möglich sein sich einigermaßen frei auf der Tischfläche zu bewegen. Ist die Dichte der Robuttons so groß, dass ein Robutton kaum von der Stelle kommt, weil er den anderen Robuttons ständig ausweichen muss, so muss entweder die Tischfläche vergrößert oder die Anzahl der Robuttons verringert werden. Ähnliches gilt bei den Münzen: Ist die Dichte der Münzen zu groß, kann es vorkommen, dass Robuttons zwischen Ansammlungen von Münzen pendeln und so auch nicht wirklich vorwärts kommen.

Ist die Dichte der Robuttons und/oder Münzen zu groß, dauert es länger die Münzen zu Ansammlungen zusammenzutragen.

Für den Fall, dass die Anzahl der Robuttons ungefähr der Anzahl der Münzen entsprechen, kann man noch folgendes beobachten:

Wird die Tischfläche bei gleichbleibender Anzahl von Robuttons und Münzen verringert, entstehen die Ansammlungen von Münzen schneller. Dies gilt nur bis zu dem Zeitpunkt, wo die Dichte der Robuttons und/oder Münzen zu groß wird.

Oft konnte man auch sehen, dass bei einer hohen Dichte von Robuttons und Münzen mehr Ansammlungen entstanden, die dann auch nicht mehr zu größeren Ansammlungen zusammengeschoben werden können.

Tischform

Folgende Tischformen können unterschieden werden: runde Tische, rechteckige Tische und polygonförmige Tische. Zu den rechteckigen Tischen gehören auch die quadratischen Tische. Dreieckige Tische gehören zu den polygonförmigen Tischen.

Polygonförmige Tische sind letztendlich eine Zusammenstellung von mehreren Tischen.

Die Tischform hat in manchen Fällen Einfluss darauf, wo die Ansammlungen der Münzen entstehen.

Programm-Dokumentation

Die Lösungsidee ist von mir in der Programmiersprache C umgesetzt worden.

Zur grafischen Darstellung der Simulation habe ich die SDL-Bibliothek[1] genutzt. Mit der ist es möglich ein Fenster zu erstellen und darin Pixel zu setzen.

Ein Robutton und eine Münze haben jeweils einen Radius von 1.

Strukturen

Zum Lösen dieser Aufgabe werden die Strukturen **`vektor`**, **`strecke`**, **`tisch`**, **`robutton`** und **`simulation`** benötigt.

Die Struktur **`simulation`** besteht aus einem **`robutton`**-Array, zwei **`vektoren`**-Arrays, einem **`tisch`**, einer **`SDL_Surface`**, einem **`char`**-Array, die Anzahl der Robuttons (**`int`**), die Anzahl der Münzen (**`int`**), Variablen zum Zoomen (**`int`**) und dem **`frameskip`** (**`int`**). In dem **`robutton`**-Array sind die Robuttons gespeichert. In einem der **`vektoren`**-Arrays sind die Positionen der Münzen gespeichert. Das andere **`vektoren`**-Array ist für die Kollisionsbehandlung notwendig. Die **`SDL_Surface`** ist das Fenster, in dem die Simulation ausgegeben wird. Das **`char`**-Array wird zur Beschleunigung der Ausgabe auf dem Bildschirm benutzt. Der **`frameskip`** gibt an, wie viele Frames der Ausgabe übersprungen werden sollen. Dies ist zur Beschleunigung der Simulation sinnvoll, denn dadurch ist es möglich die Simulation wie in einem Zeitraffer ablaufen zu lassen.

Die Struktur **`robutton`** enthält einen **`vektor`** für die Position des Robuttons und einen zweiten **`vektor`** für die Richtung des Robuttons. Die Länge des Richtungsvektor eines Robuttons beträgt immer 1. Zusätzlich wird die Information, ob der Robutton eine Münze trägt oder nicht in einer **`char`**-Variable gespeichert.

Die **`tisch`**-Struktur besteht aus einem **`vektor`**-Array, der Anzahl der Eckpunkte (**`int`**) und dem Typ des Tisches (**`char`**). In dem **`vektor`**-Array werden die Eckpunkte des Tisches gespeichert. Ist der Tisch rund, dann besitzt die **`tisch`**-Struktur nur zwei **`vektor`**-Strukturen: in dem ersten Vektor wird der Mittelpunkt und in dem zweiten Vektor der Radius gespeichert. Der Typ des Tisches kann die Werte **`KREIS`**, **`VIERECK`** oder **`POLYGON`** annehmen. Die Variable für die Anzahl der Eckpunkte wird im Falle eines runden Tisches nicht benötigt.

Die Struktur **`vektor`** besteht aus einem x- und y-Wert (**`double`**).

Die Struktur **`strecke`** besteht aus einer Anfangs- und Endkoordinate (**`vektor`**).

Konfigurationsdatei

Das Programm wird mit dem Dateinamen einer Konfigurationsdatei aufgerufen. Die Konfigurationsdatei enthält alle Parameter für die Simulation.

Eine Konfigurationsdatei sieht für einen polygonförmigen Tisch beispielsweise so aus:

1 http://www.libsdl.org/

```
# Typ
Polygon
# Anzahl Robuttons
100
# Anzahl Muenzen
500
# Anzahl Ecken
7
# Koordinaten der Ecken
# x y
50 10
400 300
795 10
750 700
500 500
350 790
50 600
```

Die Reihenfolge der Daten in der Konfigurationsdatei sind: Typ des Tisches (Kreis, Viereck, Polygon), Anzahl der Robuttons, Anzahl der Münzen, Anzahl der Ecken des Tisches und die Koordinaten der Ecken.

Zeilen, die mit einer Raute beginnen sind Kommentarzeilen.

Ist der Tisch rund wird statt der Anzahl und Koordinaten der Ecken, der Mittelpunkt und Radius des Kreises angegeben.

Ist der Tisch ein viereckiger Tisch, d.h. hat der Tisch die Form eines Rechtecks oder Quadrats, muss die Anzahl der Ecken nicht angegeben werden und es reicht die Koordinate der oberen linken Ecke und der unteren rechten Ecke des Tisches anzugeben.

Die Koordinate (0 | 0) entspricht der oberen linken Ecke im Fenster. Die Koordinate (800 | 800) entspricht der unteren rechten Ecke des Fensters. Die x-Achse verläuft horizontal und die y-Achse vertikal.

Funktionen

In der **`main`**-Funktion wird die Funktion **`simulation_starten`** mit dem Dateinamen einer Konfigurationsdatei aufgerufen.

In der Funktion **`simulation_starten`** wird dann eine Simulationsstrukur anhand der Konfigurationsdatei erstellt. Außerdem wird die **`SDL_Surface`** initialisiert, sodass ein Fenster geöffnet wird, indem die Simulation ausgegeben werden kann. Die Funktion **`simulation_starten`** bleibt solange aktiv, bis das Fenster geschlossen wird.

Als nächstes werden die Robuttons mit der Funktion **`bewege_robuttons`** bewegt. Bevor die Ausgabe auf dem Bildschirm aktualisiert wird, wird die Funktion **`bewege_robuttons`** so oft aufgerufen, wie es der eingestellt Frameskip vorgibt. Bei einem Frameskip von z. B. 2 wird die Funktion **`bewege_robuttons`** dreimal aufgerufen bevor die Ausgabe aktualisiert wird.

Die Ausgabe auf dem Bildschirm wird mit der Funktion **`update_ausgabe`** aktualisiert. Diese liest die Daten aus der Simulationsstruktur und setzt dementsprechend Pixel auf der **`SDL_Surface`**.

Die Funktion **`bewege_robuttons`** ruft die Funktion **`bestimme_neue_positionen`** auf, die die Robuttons um die Länge und Richtung ihres Richtungsvektors bewegt. Die so entstandenen neuen Positionen der Robuttons werden temporär in dem Array **`neue_positionen`** gespeichert.

Damit hat man die Bewegungsstrecken der Robuttons ermittelt, die sie in einem Tick der Simulation zurücklegen. Der Startpunkt der Bewegungsstrecke ist in dem Array **`robuttons`** und der Endpunkt der Bewegungsstrecke ist in dem Array **`neue_positionen`** enthalten.

Bevor die Robuttons den Endpunkt der Bewegungsstrecke einnehmen, müssen zunächst eventuelle Kollisionen behandelt werden. Die Kollisionsbehandlung erfolgt in der Funktion **`bewege_robuttons`**.

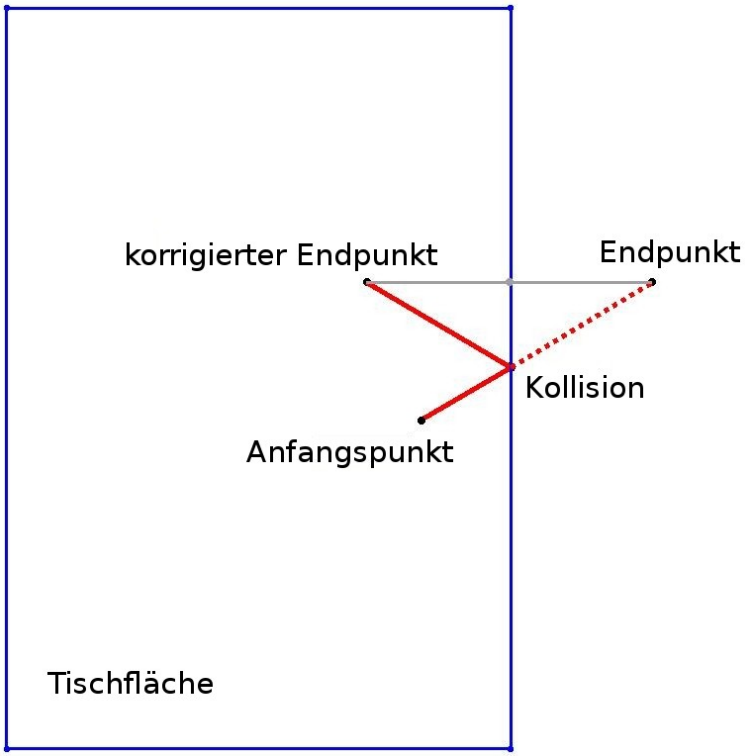
Kollisionen

Eine Kollision liegt vor, wenn sich der Endpunkt der Bewegungsstrecke eines Robuttons ändert. Der Endpunkt ändert sich, wenn ein Robutton auf eine Tischkante, einen anderen Robutton oder eine Münze trifft. Der letztere Fall gilt nur als Kollision, wenn der Robutton bereits eine Münze trägt, denn nur dann ändert sich der Endpunkt der Bewegungsstrecke.

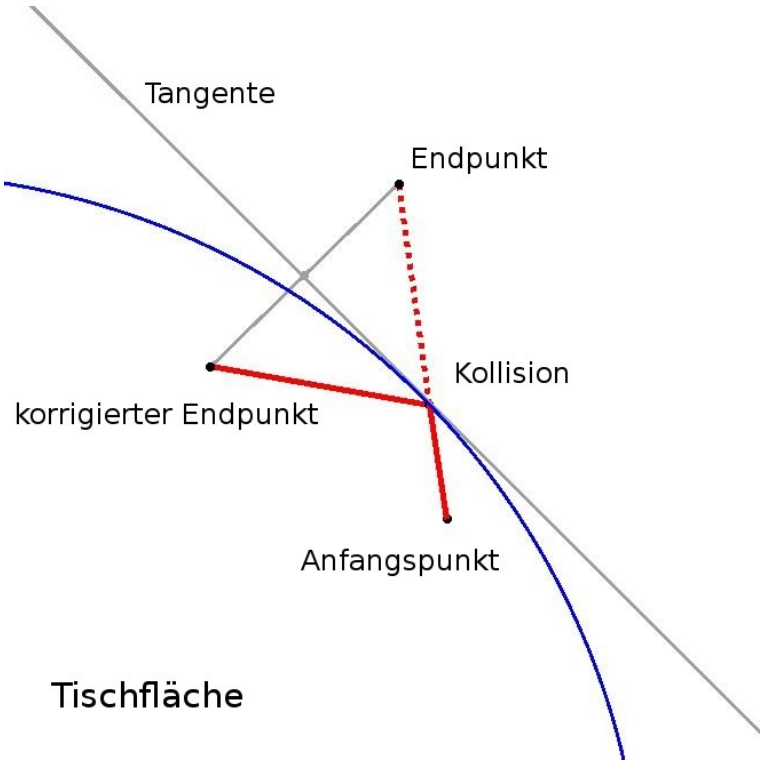
Jede der neu bestimmten Positionen muss auf die drei möglichen Kollisionen geprüft werden. Sobald eine Kollision auftrat, müssen die alle Kollisionen nochmal geprüft werden. Es kann z. B. vorkommen, dass ein Robutton von einer Tischkante reflektiert wird und dann auf eine Münze, einen anderen Robutton oder eine Tischkante trifft.

Die Funktion **`pruefe_kollision_tischkante_rund`** ist für die Reflexion der Robuttons an der Tischkante von runden Tischen zuständig. Die Funktion **`pruefe_kollision_tischkante_eckig`** dementsprechend für die eckigen Tische (rechteckig oder polygonförmig).

Eine Kollision des Robuttons mit einer Tischkante liegt vor, wenn der Endpunkt der Bewegungsstrecke des Robuttons außerhalb der Tischfläche ist. In diesem Fall muss eine Geradengleichung für die Tischkante ermittelt werden. Die Reflexion des Robuttons an dieser Tischkante ist gleichbedeutend mit der Spiegelung des Endpunktes der Bewegungsstrecke an der Geraden:



Bei einem runden Tisch muss zunächst eine Tangente an dem Kollisionspunkt ermittelt werden. Die Reflexion des Robuttons ist dann gleichbedeutend mit der Spiegelung des Endpunktes der Bewegungsstrecke an der Tangenten:



Die Funktion **`pruefe_kollision_robuttons`** ist für die Kollision eines Robuttons mit einem anderen Robutton zuständig. Eine Kollision zwischen zwei Robuttons liegt vor wenn der Abstand der Endpunkte ihrer Bewegungsstrecken weniger als 1 beträgt. In diesem Fall drehen sich beide

Robuttons wie in der Aufgabenstellung beschrieben und setzen ihren Weg fort.

Die Funktion **`pruefe_kollision_muenze`** ist für die Kollision eines Robuttons mit einer Münze zuständig. Eine Kollision mit einer Münze liegt vor, wenn der Abstand des Endpunktes der Bewegungsstrecke weniger als 1 beträgt. In diesem Fall verhält sich der Robutton wie es in der Aufgabenstellung beschrieben ist.

Solange also eine der Funktionen **`pruefe_kollision_tischkante_rund`** (bzw. **`pruefe_kollision_tischkante_eckig`**), **`pruefe_kollision_robutton`** oder **`pruefe_kollision_muenze`** den Wert 1 zurückgibt (Endpunkt der Bewegungsstrecke hat sich geändert), müssen alle Kollisionsprüfungen nochmal durchgeführt werden. Dabei kann es aber auch vorkommen, dass ein Robutton irgendwo eingeklemmt ist, z. B. wenn um ihn herum nur Robuttons sind. Dies würde eine Endlosschleife bedeuten. Um dies zu vermeiden ist es sinnvoll, dass der Robutton dann erst mal stehen bleibt. Die Funktion **`pruefe_kollision_muenze`** gibt nur den Wert 1 zurück, wenn sich der Endpunkt des Robuttons geändert hat, also wenn der Robutton eine Münze ablegt.

Nachdem alle Kollisionen behandelt wurden, bekommen die Robuttons den (eventuell korrigierten) Endpunkt ihrer Bewegungsstrecke als neue Position zugewiesen. Dies ist die letzte Aktion der Funktion **`bewege_robuttons`**.

Tischfläche

Die Tischfläche wird durch die Funktion **`ist_tischflaeche`** bestimmt. Sie bekommt eine Tischstruktur und eine Koordinate als Parameter übergeben (x- und y-Wert). Die Funktion gibt 1 zurück, wenn die Koordinate Teil der Tischfläche ist und 0, wenn dies nicht der Fall ist.

Je nach Typ des Tisches werden unterschiedliche Methoden zur Prüfung, ob die Koordinate zur Tischfläche gehört, verwendet:

In den Fällen, wo der Tisch rund oder rechteckig ist, ist die Prüfung trivial.

Bei einem runden Tisch muss lediglich geprüft werden, ob der Abstand des Robuttons zum Mittelpunkt kleiner als der Radius ist.

Bei einem rechteckigen Tisch muss die x-Koordinate des Robuttons innerhalb der x-Koordinaten der senkrechten Tischkanten liegen. Die y-Koordinate des Robuttons muss innerhalb der y-Koordinaten der waagerechten Tischkanten liegen.

Um bei einem polygonförmigen Tisch zu prüfen, ob die Koordinate zur Tischfläche gehört, wird ausgehend von der zu überprüfenden Koordinate ein Strahl in eine Richtung „geschossen“. Dann wird gezählt, mit wie vielen Tischkanten der Strahl einen Schnittpunkt besitzt. Ist die Anzahl der Schnittpunkte ungerade, gehört die Koordinate zur Tischfläche. Bei gerader Anzahl der Schnittpunkte gehört die Koordinate nicht zur Tischfläche.

Programm-Ablaufprotokoll

Das Programm wird mit dem Dateinamen einer Konfigurationsdatei aufgerufen.

Tastaturbelegung

Bei dem Aufruf des Programms wird die Tastaturbelegung ausgegeben und ein Fenster erstellt, indem die Simulation abläuft:

```
$ ./robuttons rund.tisch

Folgende Tasten koennen waehrend der Simulation
benutzt werden:

+               Hineinzoomen

-               Hinauszoomen

Pfeiltasten     Scrollen

Tasten 0 - 9    Frameskip einstellen:
                1 entspricht 10% des max. Frameskip
                2 entspricht 20% des max. Frameskip
                ...
                0 entspricht dem max. Frameskip

Strg (links)    Frameskip um 1 verringern

Strg (rechts)   Frameskip um 1 erhoehen

r               Frameskip auf 0 setzen
```

Es kann also während der Simulation gezoomt und gescrollt werden. Das Scrollen ist nur bei einem Frameskip von 0 sinnvoll. Scrollen ist auch nur möglich, wenn hineingezoomt wurde.

Mit den Tasten 0 bis 9 kann die Simulation beschleunigt werden, weil dann die *ständige* Aktualisierung der Ausgabe wegfällt. Die Ausgabe wird dann einfach nicht mehr so häufig aktualisiert. Möchte man den Frameskip wieder zurücksetzen, drückt man die Taste „r“.

Farben

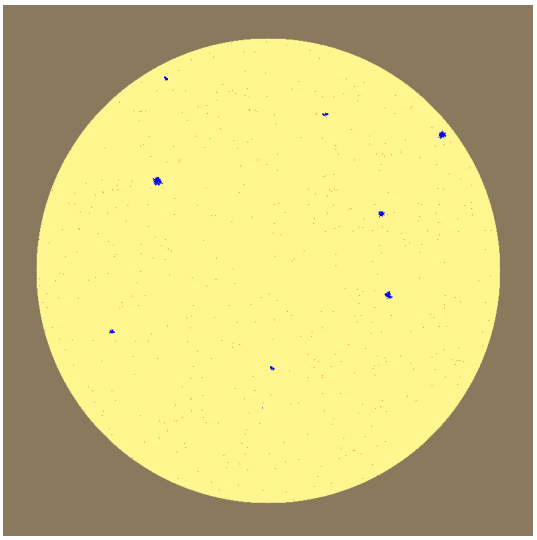
In dem Ausgabefenster schaut man von oben auf dem Tisch. Dabei sind braune Pixel der Boden und beige/gelbe Pixel die Tischfläche.

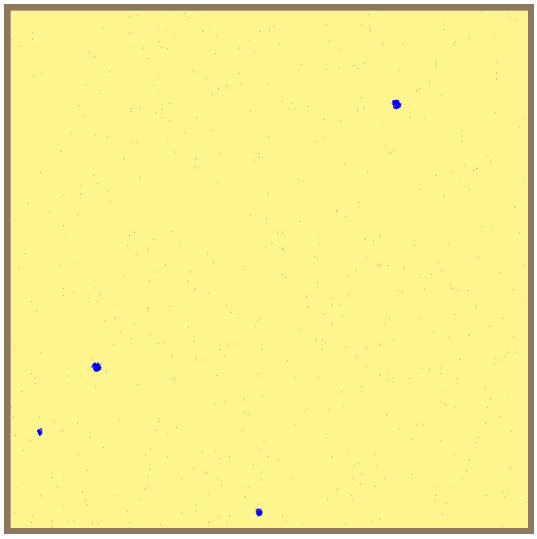
Die Robuttons sind rot, wenn sie keine Münze tragen und werden grün, wenn sie eine Münze aufnehmen. Solange ein Robutton grün ist, trägt er eine Münze. Lässt er seine Münze fallen wird er wieder rot dargestellt.

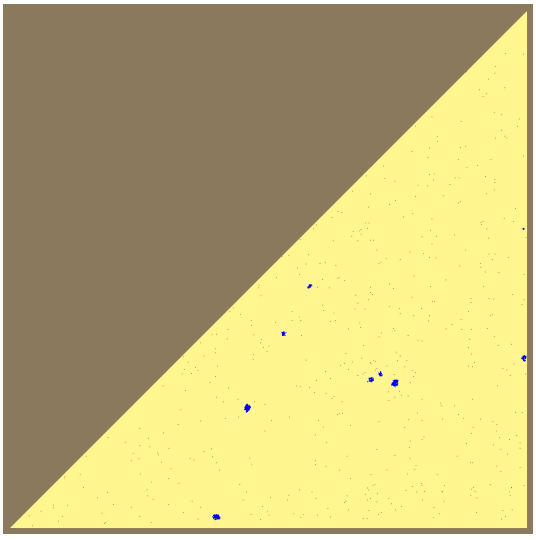
Münzen sind blau dargestellt.

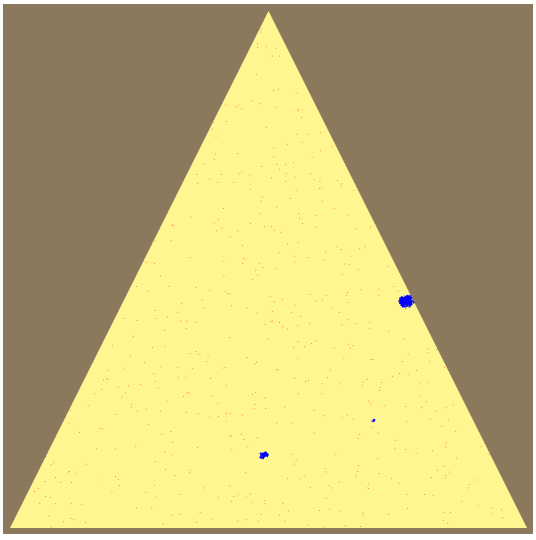
Testläufe

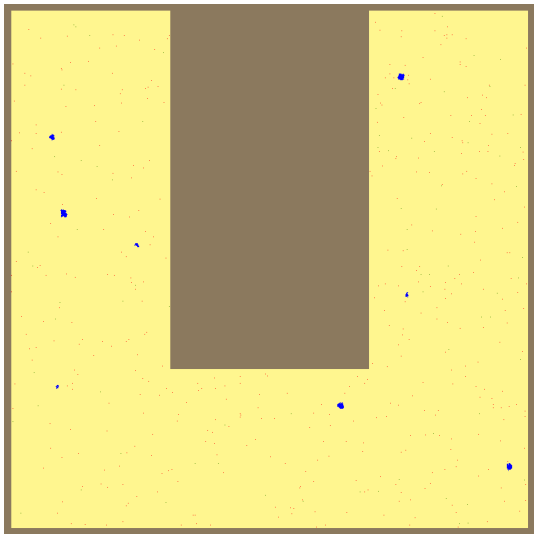
Im Folgenden sind die Ergebnisse der Simulation für verschiedene Tischformen zu sehen. Die Anzahl der Robuttons und Münzen blieb dabei gleich (500 Robuttons, 500 Münzen):

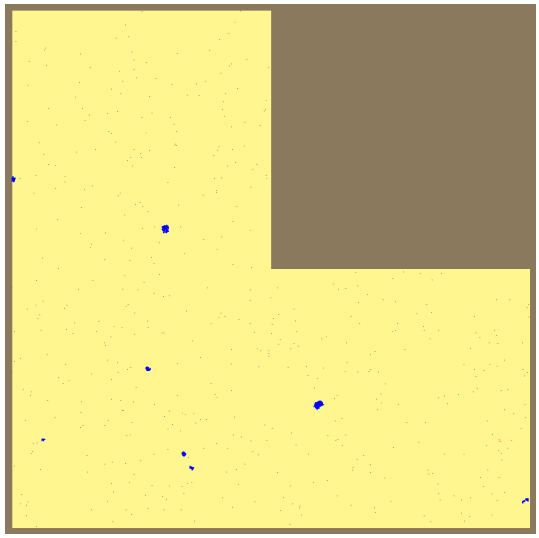


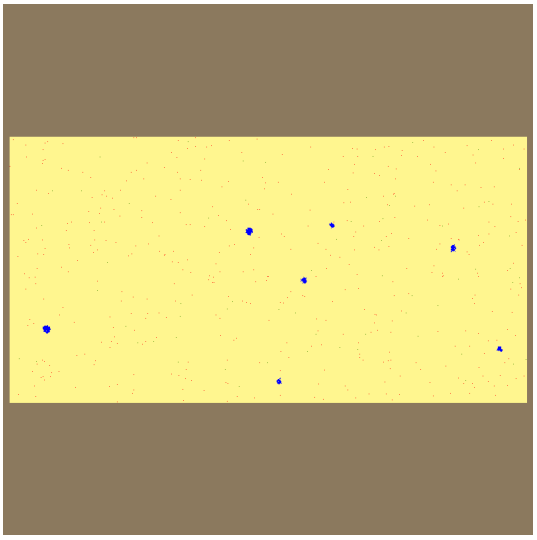


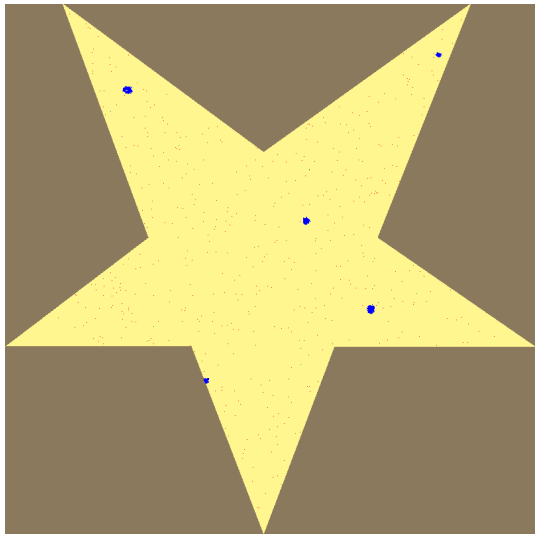


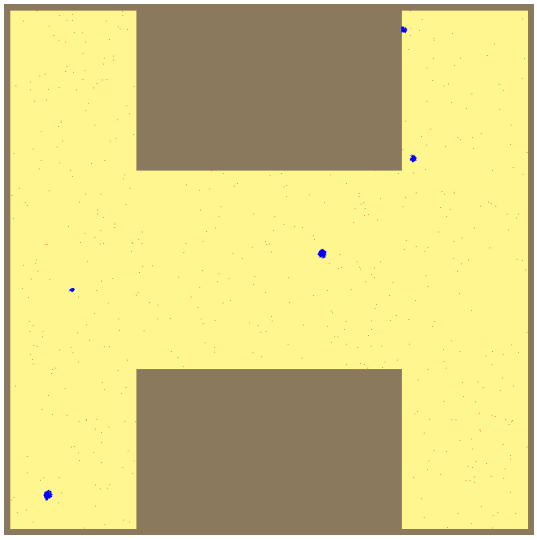


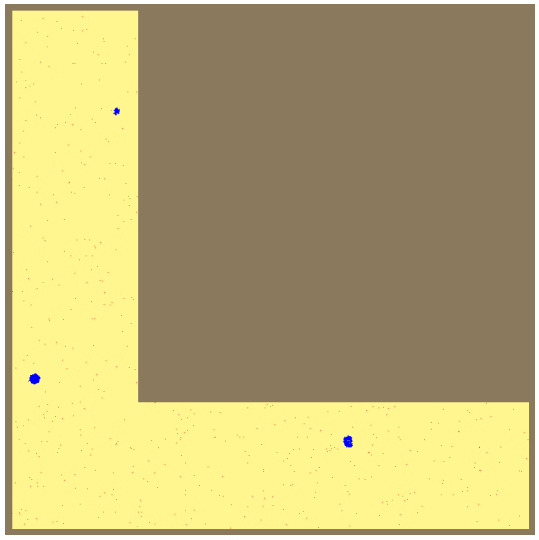




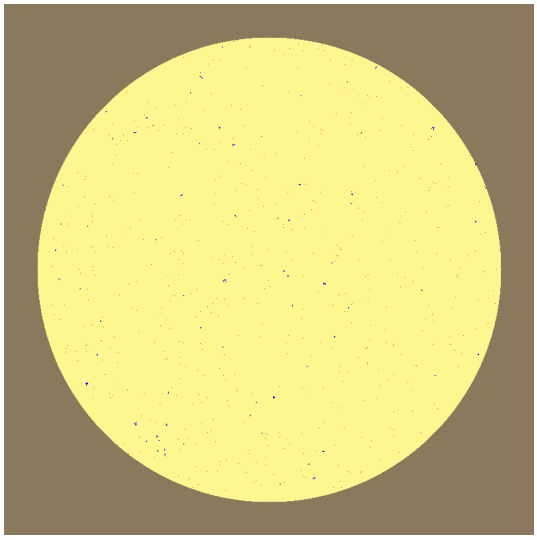




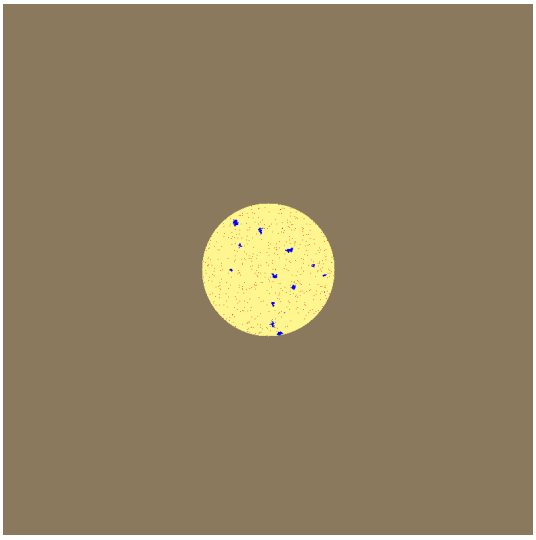




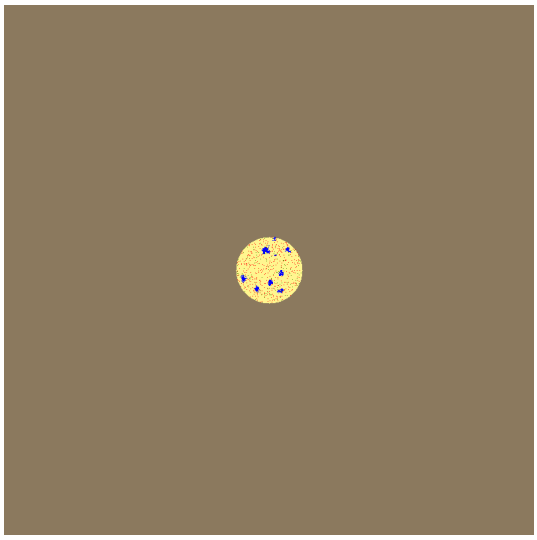
Verringert man die Tischfläche bei gleichbleibender Anzahl an Robuttons und Münzen geschieht das Zusammentragen der Münzen schneller. Beide Screenshots wurden nach der selben Anzahl an Ticks (3000) aufgenommen:



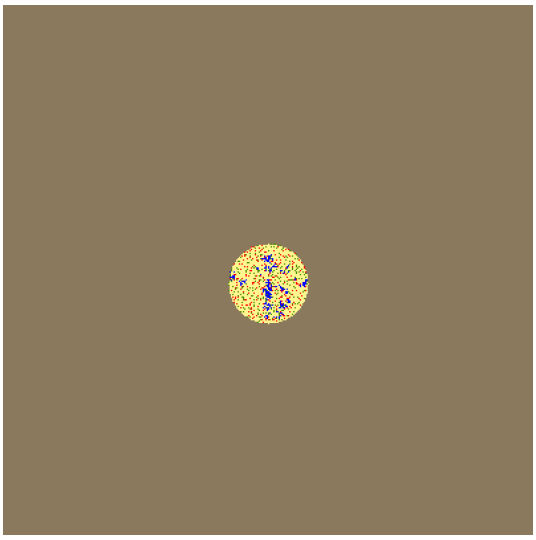
Radius: 300



Radius: 100



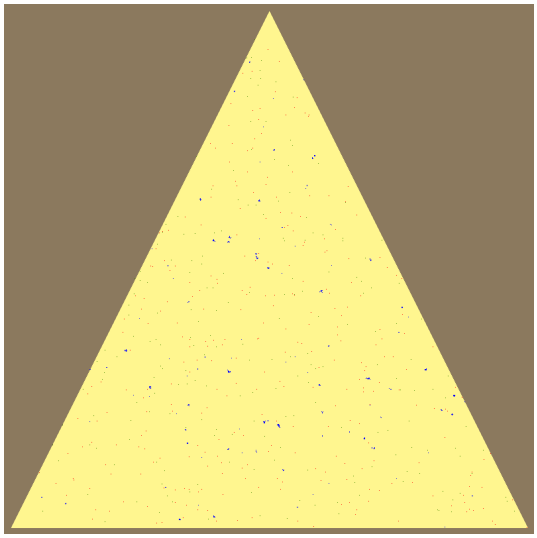
Radius: 50

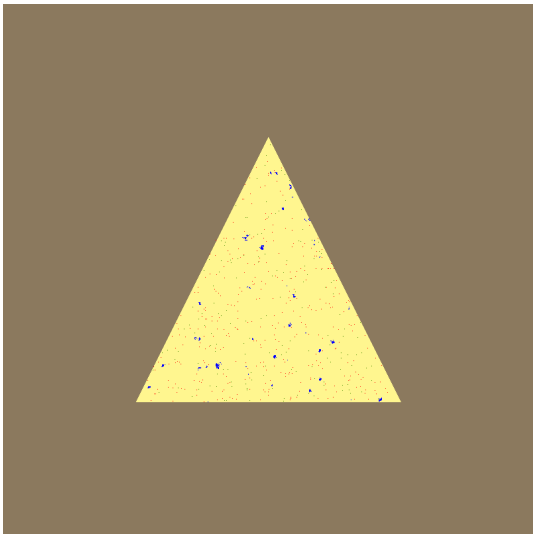


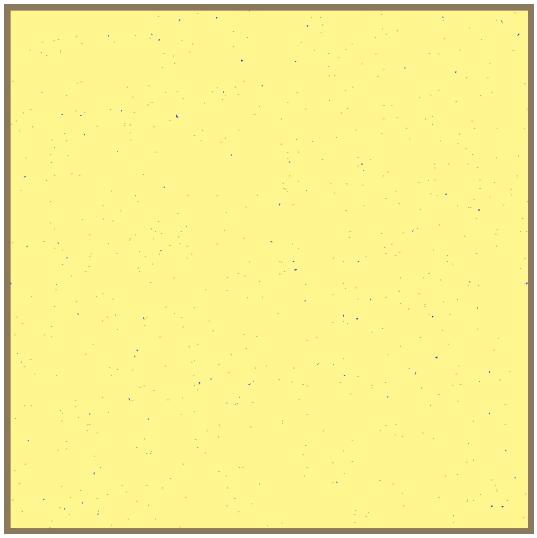
Radius: 25 (gezoomt)

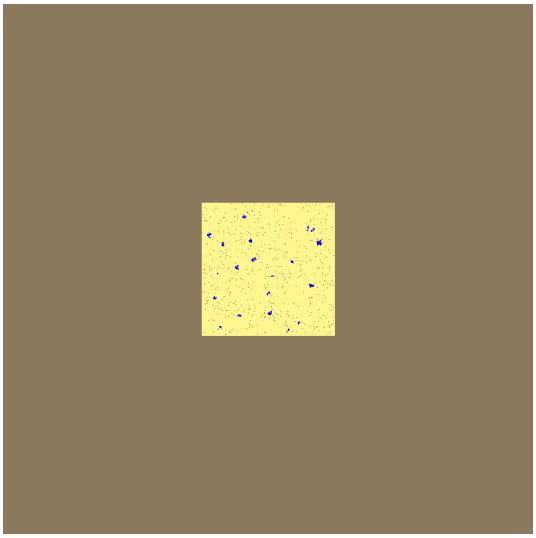
Bei einem Radius von 25 ist die Dichte der Robuttons und Münzen bereits zu hoch, sodass zwar Ansammlungen entstehen, diese aber erst nach sehr langer Zeit und mit viel Glück zu einer oder mehreren großen Ansammlungen zusammengetragen werden. Denn die Robuttons kommen dort kaum von der Stelle. Selbst nach 40.000 Ticks gibt es keine größere Ansammlung von Münzen, stattdessen „wandern“ die kleinen Münzansammlungen umher.

Die selben Ergebnisse erhält man bei anderen Tischformen: Das Zusammentragen der Münzen geschieht schneller, je kleiner die Tischfläche wird. Wieder wurden die Screenshots nach 3000 Ticks aufgenommen und die Anzahl der Robuttons und Münzen blieb unverändert:

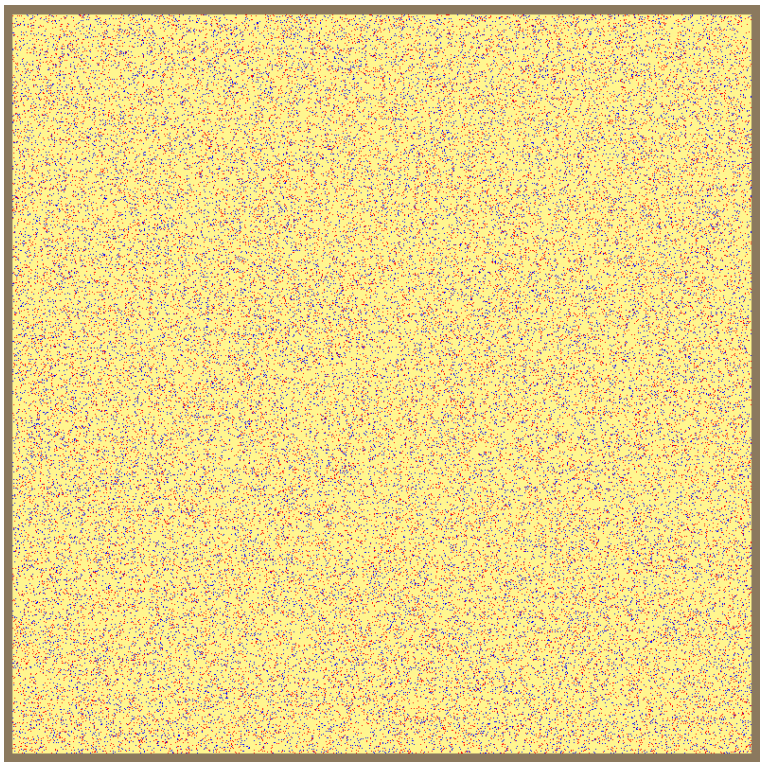






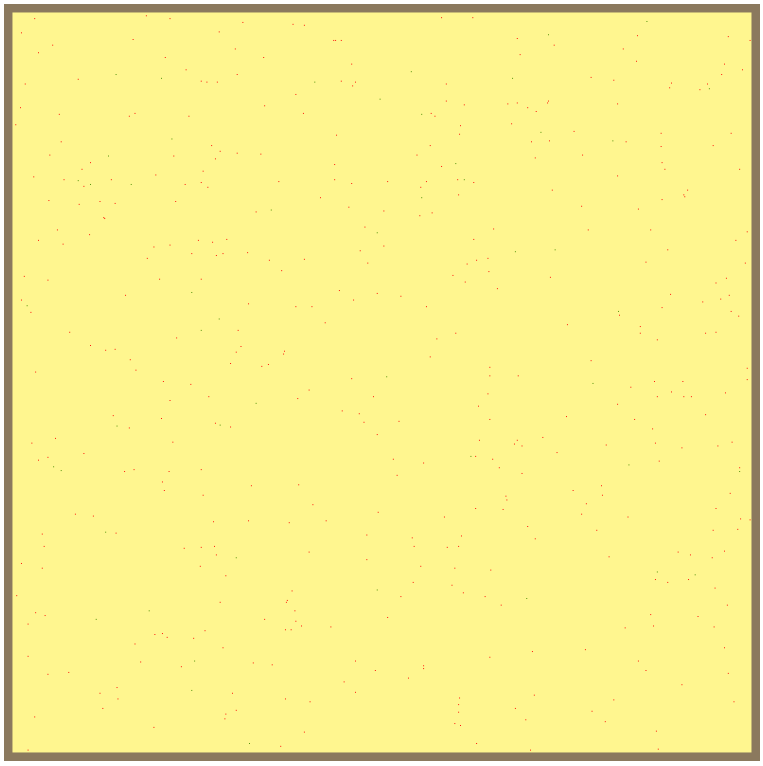


Ein Beispiel für eine zu große Dichte an Robuttons und Münzen:



Auch hier bilden sich Ansammlungen von Münzen, doch sind die Robuttons eher damit beschäftigt den anderen Robuttons auszuweichen. Und es dauert dementsprechend länger bis größere Ansammlungen entstehen.

Gibt es viel weniger Münzen als Robuttons, werden alle Münzen von den Robuttons aufgenommen:



Gibt es viel mehr Münzen als Robuttons dauert es viel länger, bis große Ansammlungen von Münzen entstehen.