Übungen - Bildgenierung

Übung 07.

Jose Jimenez

Angewandte Informatik
Bergische Universität Wuppertal

December 21, 2022

Table of Contents

Objekterzeugung mit Grammatiken, Blumenwiese

Ok, wir brauchen wieder eine Animation. Und wir brauchen auch einige Pflanzen.

Objekterzeugung mit Grammatiken, Blumenwiese

Ok, wir brauchen wieder eine Animation. Und wir brauchen auch einige Pflanzen. Wie üblich können wir Vektoren verwenden.

```
const int anzahlBilder = 30;
int maindraw()
{
  vector<Drawing> pics(anzahlBilder);
  vector<Blume> blumen{  .... } ?


  Drawing pic(1400, 800, 0);
  pic.show();
}
```

Objekterzeugung mit Grammatiken, Blumenwiese

dann wollen wir in einer for-Schleife so etwas tun...

```
const int anzahlBilder = 30;
int maindraw()
{
  ...
  for (int i = 0; i < anzahlBilder; ++i){
      // 1. Background malen

      // 2. Blumen weiterentwickeln lassen

      // 3. Blume malen

      // 4. Bild im Vektor speichern
  }
}
```

Objekterzeugung mit Grammatiken, Blumenwiese

1 und **4** sind einfach und wir haben die schon gemacht.

1. Background malen
2. Blumen weiterentwickeln lassen
3. Blume malen
4. Bild im Vektor speichern

Objekterzeugung mit Grammatiken, Blumenwiese

1 und **4** sind einfach und wir haben die schon gemacht.

1. Background malen
2. Blumen weiterentwickeln lassen
3. Blume malen
4. Bild im Vektor speichern

2 und **3**... sie könnten Methoden der Klasse Blume sein.

Objekterzeugung mit Grammatiken, Blumenwiese

Eine Blume sollte eine Position, eine Ableitung und 2 Methoden haben....

Objekterzeugung mit Grammatiken, Blumenwiese

Eine Blume sollte eine Position, eine Ableitung und 2 Methoden haben....

```
const int anzahlBilder = 30;
class Blume{
private:
  DPoint2D position;
  string ableitung;
public:
  Blume(DPoint2D pos){
    position = pos;
    ableitung = "zK";  // Warum?
  }
  void weiter();  // Blumen weiterentwickeln lassen
  void maleMich(Drawing& pic);   // Blume malen

}
```

Beginnen wir mit "**weiter**".

Objekterzeugung mit Grammatiken, Blumenwiese

```
void weiter(){

}
```

"Wählen Sie bei mehreren Ableitungsmöglichkeiten jeweils passende Wahrscheinlichkeiten, so dass sich insbesondere ein sinnvoller Anteil an Knospen weiterentwickelt".

Objekterzeugung mit Grammatiken, Blumenwiese

Zu beginnen

Nichtterminale:	**K**, **B**
Terminale:	**z**, **I**, **(**, **)**, **[**, **]**
Produktionen:	$\mathbf{K} \rightarrow \mathbf{K} - \epsilon \qquad \epsilon =:$ "nichts"
	$\mathbf{K} \rightarrow \mathbf{B}$
	$\mathbf{B} \rightarrow \mathbf{B} - \epsilon$
	$\mathbf{K} \rightarrow \mathbf{I}$
	$\mathbf{K} \rightarrow \mathbf{zK} - \mathbf{(K)zK} - \mathbf{[K]zK} - \mathbf{(K)[K]zK}$
Startwort:	**zK**

```
Blume(DPoint2D pos)  {
    position = pos;
    ableitung = "zK";   //Warum?
  }
```

Z = Zweig, K= Knospe, B= Blute, I= blatt.

Objekterzeugung mit Grammatiken, Blumenwiese

Es gibt 2 Fälle, in denen wir Produktionen haben: **K** und **B**.

Nichtterminale:	**K**, **B**
Terminale:	**z**, **I**, **(**, **)**, **[**, **]**
Produktionen:	$\mathbf{K} \rightarrow \mathbf{K} - \epsilon \qquad \epsilon =:$ "nichts"
	$\mathbf{K} \rightarrow \mathbf{B}$
	$\mathbf{B} \rightarrow \mathbf{B} - \epsilon$
	$\mathbf{K} \rightarrow \mathbf{I}$
	$\mathbf{K} \rightarrow \mathbf{zK} - \mathbf{(K)zK} - \mathbf{[K]zK} - \mathbf{(K)[K]zK}$
Startwort:	**zK**

Z = Zweig, K= Knospe, B= Blute, I= blatt.

Objekterzeugung mit Grammatiken, Blumenwiese

Es gibt 2 Fälle, in denen wir Produktionen haben: **K** und **B**.

```
void weiter(){
   string neu = "";
   double r;
   for (auto z : ableitung)
     switch (z){
         case 'K':
          ...
         case 'B':
           ...
         default:
           neu += z;
       }
   ableitung = neu;
 }
```

(nächste Folie + Code anzeigen)

Objekterzeugung mit Grammatiken, Blumenwiese

"Wählen Sie bei mehreren Ableitungsmöglichkeiten jeweils passende Wahrscheinlichkeiten, so dass sich insbesondere ein sinnvoller Anteil an Knospen weiterentwickelt".

Nichtterminale:	**K**, **B**
Terminale:	**z**, **I**, **(**, **)**, **[**, **]**
Produktionen:	$\mathbf{K} \rightarrow \mathbf{K} - \epsilon \qquad \epsilon =:$ "nichts"
	$\mathbf{K} \rightarrow \mathbf{B}$
	$\mathbf{B} \rightarrow \mathbf{B} - \epsilon$
	$\mathbf{K} \rightarrow \mathbf{I}$
	$\mathbf{K} \rightarrow \mathbf{zK} - \mathbf{(K)zK} - \mathbf{[K]zK} - \mathbf{(K)[K]zK}$
Startwort:	**zK**

Objekterzeugung mit Grammatiken, Blumenwiese

Jetz, male Funktion:

```
 void maleMich(Drawing& pic) const{

}
```

Objekterzeugung mit Grammatiken, Blumenwiese

```
void maleMich(Drawing& pic) const{
   for (auto z : ableitung)
     switch (z) {
         case 'K': // Knospe: kleiner blauer Kreis
         case 'B': // Blüte: drei rote Kreise, zwei klein, einer größer
         case 'z': // Zweig: braune Linie
         case 'l': // Blatt: größerer grüner Kreis
         case '(': // Neigung nach Links
         case '[': // Neigung nach Rechts
         case ')':
         case ']':
   }
 }
```

Objekterzeugung mit Grammatiken, Blumenwiese

wir brauchen...

```
void maleMich(Drawing& pic) const{
   DPoint2D pos = position;
   DPoint2D posneu;
   double winkel(M_PI / 2);
   stack<DPoint2D> posstack;
   stack<double> winkelstack;
   int level = 1;

   for (auto z : ableitung)
     switch (z) {
        ...
   }
 }
```

Warum: Code Anzeigen.

Partikelsysteme, Feuer

Ok, wir brauchen wieder eine Animation. Und wir brauchen auch viele Partikeln.

Partikelsysteme, Feuer

Ok, wir brauchen wieder eine Animation. Und wir brauchen auch viele Partikeln. Wie üblich können wir Vektoren verwenden.

```
int maindraw()
{
  vector<Drawing> pics(anzahlBilder);
  vector<Partikel> p;  //(???)

  Drawing pic(600, 600, 0);

}
```

Partikelsysteme, Feuer

Ok, wir brauchen wieder eine Animation. Und wir brauchen auch viele Partikeln. Wie üblich können wir Vektoren verwenden. dann wollen wir in einer for-Schleife so etwas tun...

```
int maindraw(){
  for (int i = 0; i < anzahlBilder; ++i){
      // 1. Background

      // 2. neue Partikel erzeugen

      // 3. Partikel bewegen

      // 4. verschwundene Partikel löschen

      // 5. Partikel malen

      // 6. Bild Speichern

  }
```

Partikelsysteme, Feuer

Konstanten

```
const int anzahlBilder = 500;
const int neuProBild = 1000;
const double g = 0.003;          // Gravitation
const DrawColour farbverlauf[] = { DrawColour(255, 255, 255), // weiß
                                   DrawColour(255, 255, 0),   // gelb
                                   DrawColour(255, 05, 0),    // rot
                                   DrawColour(0, 0, 0) };     // schwarz

int maindraw(){
  ...
}
```

Ok, zurück zu mandraw()

Partikelsysteme, Feuer

was können wir schon tun?

```
int maindraw(){
  for (int i = 0; i < anzahlBilder; ++i){
      // 1. Background

      // 2. neue Partikel erzeugen

      // 3. Partikel bewegen

      // 4. verschwundene Partikel löschen

      // 5. Partikel malen

      // 6. Bild Speichern

    }
}
```

Partikelsysteme, Feuer

was können wir schon tun?

```
int maindraw(){
  for (int i = 0; i < anzahlBilder; ++i){
      // 1. Background
        pic = DrawColour(90, 180, 90); // grün
      // 2. neue Partikel erzeugen
        for (int j = 0; j < neuProBild; ++j)
        p.push_back(Partikel(.....));

      // 3. Partikel bewegen

      // 4. verschwundene Partikel löschen

      // 5. Partikel malen

      // 6. Bild Speichern
       pics[i] = pic;

    }
```

Partikelsysteme, Feuer

3. Partikel bewegen
4. verschwundene Partikel löschen
5. Partikel malen

Partikelsysteme, Feuer

Noch einmal: Wir brauchen so etwas wie eine "Partikel"-Klasse.

3. Partikel bewegen
4. verschwundene Partikel löschen
5. Partikel malen

Klassenattribute

```
class Partikel
{
private:
  DPoint2D position;
  DPoint2D geschwindigkeit;
  // Alter des Partikels: 0 bis 4, Start bei 0.8 (weiße Phase soll kürzer
  double gluehzustand;
};
```

Methoden

```
class Partikel{
private:
  DPoint2D position;
  DPoint2D geschwindigkeit;
  double gluehzustand;
public:
  Partikel(DPoint2D startpos); // Konstruktor

};
```

Methoden

```
class Partikel{
private:
  DPoint2D position;
  DPoint2D geschwindigkeit;
  double gluehzustand;
public:
  Partikel(DPoint2D startpos);
  void weiter(); //bewegen
};
```

Methoden

```
class Partikel{
private:
  DPoint2D position;
  DPoint2D geschwindigkeit;
  double gluehzustand;
public:
  Partikel(DPoint2D startpos);
  void weiter();
  bool verschwunden() const { return gluehzustand >= 4; }
};
```

Methoden

```
class Partikel{
private:
  DPoint2D position;
  DPoint2D geschwindigkeit;
  double gluehzustand;
public:
  Partikel(DPoint2D startpos);
  void weiter();
  bool verschwunden() const { return gluehzustand >= 4; }
  DrawColour farbe() const
  { return farbverlauf[static_cast<int>(gluehzustand)]; }
};
```

Methoden

```
class Partikel{
private:
  DPoint2D position;
  DPoint2D geschwindigkeit;
  double gluehzustand;
public:
  Partikel(DPoint2D startpos);
  void weiter();
  bool verschwunden() const { return gluehzustand >= 4; }
  DrawColour farbe() const
  { return farbverlauf[static_cast<int>(gluehzustand)]; }
  // Partikel als einfache Punkte
  void maleMich(Drawing& pic) const { pic.drawPoint(position, farbe()); }
};
```

Partikelsysteme, Feuer

Methoden

```
Partikel::Partikel(DPoint2D startpos){


}
```

Methoden

```
Partikel::Partikel(DPoint2D startpos){
  // x variiert in [-20,20]
  position = startpos + DPoint2D(40 * rnd01() - 20, 0);
  // x variiert in [-1,1], y variiert in [1.5,3.5]
  geschwindigkeit
    = DPoint2D(0, 2.5) + DPoint2D(2 * rnd01() - 1, 2 * rnd01() - 1);
  gluehzustand = 0.8;

}
```

Partikelsysteme, Feuer

Methoden

```
void Partikel::weiter()
{

}
```

Methoden

```
void Partikel::weiter()
{
  // dx, dy variieren in [-0.3,0.3]
  geschwindigkeit
    = geschwindigkeit + 0.6 * DPoint2D(rnd01() - 0.5, rnd01() - 0.5);
  // Gravitation: y reduzieren um 0.0015
  position = position + geschwindigkeit - 0.5 * DPoint2D(0, g);
  // Gravitation: y reduzieren um 0.003
  geschwindigkeit = geschwindigkeit - DPoint2D(0, g);
  // Leuchtkraft reduzieren
  gluehzustand += 0.05 * rnd01();
}
```

Methoden

```
void Partikel::weiter()
{
  // dx, dy variieren in [-0.3,0.3]
  geschwindigkeit
    = geschwindigkeit + 0.6 * DPoint2D(rnd01() - 0.5, rnd01() - 0.5);
  // Gravitation: y reduzieren um 0.0015
  position = position + geschwindigkeit - 0.5 * DPoint2D(0, g);
  // Gravitation: y reduzieren um 0.003
  geschwindigkeit = geschwindigkeit - DPoint2D(0, g);
  // Leuchtkraft reduzieren
  gluehzustand += 0.05 * rnd01();
}
```

lass uns maindraw() gücken.

Hermite-Kurven Schleife