DIE SECHS SONDERZEICHEN

WIKIDOKUMENTATION

Die Generierung des Jungles

von Marius Stein

Man kann den Jungle, der sich aus zufällig generierten Wegen zusammensetzen soll, sehr gut über die Berechnung eines Minimal Spanning Trees erstellen.
Hierzu habe ich zuerst einmal alle Koordinaten berechnet, in welchen sich der Weg des Jungles befinden darf. Hierbei habe ich mich darauf beschränkt nur den Bereich zwischen dem mittleren und dem oberen Weg zu berechnen, da man diesen am Ende einfach auf alle vier anderen Bereiche spiegeln kann.
Als nächstes habe ich dann den berechneten Bereich in kleinere Junglesektoren eingeteilt.
Zuerst habe ich eine feste Größe für alle Junglesektoren definiert. Danach habe ich den ersten Sektor so platziert, dass er zwar im Jungle liegt, aber gleichzeitig auch so nah wie möglich an der unteren Basis.
Von diesem ersten Sektor wird nun in alle vier Richtungen(oben, unten, links, rechts)
eine Breitensuche ausgeführt, die überprüft ob in der jeweiligen Richtung noch ein weiterer Junglesektor platziert werden kann, ohne dass dieser außerhalb des Jungles liegt.
Ist diese Bedingung erfüllt, wird ein neuer Jungle Sektor erstellt. Gleichzeitig wird nun im neu gefundenen und im alten Sektor eine Verbindungslinie gespeichert, diese Verbindungslinie liegt an einer der vier Seiten des Jungle Sektors. Hierbei muss beachtet werden, ob der neu gefundene Sektor bereits schon vorher einmal besucht wurde, falls dies so ist muss hier nurnoch die Verbindungslinie eingetragen werden.
Wenn man nun für jeden gefundenen Junglesektor von jeder Verbindungslinie zum Mittelpunkt dieses Sektors einen Weg berechnen würde, hätte man bereits einen zusammenhängenden Jungle erstellt. Man müsste nurnoch die Verbindungen zum Hauptweg erstellen. Dieser Jungle würde nun aber immer gleich aussehen.
Damit der Jungle auch von Berechnung zu Berechnung anders aussieht, kann man die berechneten Jungle Sektoren als Vertex und die Verbindungslinien als Arc in einem Graph speichern. Wenn man jeden Arc nun mit zufälligen Kosten versieht und hieraus einen Minimal Spanning Tree berechnet hat man nun einen Graphen, welcher von Berechnung zu Berechnung anders aussieht.
Diesen MST muss man nun nur noch wie oben beschrieben auf der Karte darstellen und man hat einen zufällig generierten Jungle erstellt.

Das Laden von Images automatisieren

von Christian Westhoff

Da bei größeren Projekten auch die Anzahl der zu ladenene Bilddateien erheblich steigt, ist es von Vorteil hier Abhilfe zu schaffen und diesen Schritt zu automatisieren, da so erheblich Zeit eingespaart werden kann.

Da wir selber in unserem Projekt eine Vielzahl von Bildern laden müssen, habe ich ein Methode geschrieben, die rekursiv sämtliche Dateien in einem Archiv einliesst und als Datentype File in einer LinkedList abspeichert. Die Voraussetzung ist hierfür, dass die Images natürlich alle in einem Ordner seien müssen. Diese Liste kann dann später durchgegangen werden, um die Dateien einzulesen und z.B. in einer HashMap mit dem Pfad als Key gespeichert werden.

Achtung! Es muss darauf geachtet werden, dass UNIX-Systeme (OSX, Ubuntu usw.) den Slasch ("/") als Trennzeichen verwenden, Windows aber den Backslasch ("\"). Beim Einlesen der Bilder akzeptiert Windows auch den Slash. Somit kann noch eine Methode geschrieben werden, die den Backslash durch den Slash ersetzt.

```
1  /**
2   * Methode, die saemtliche Dateien in einem Ordner rekursiv einliesst.
3       * Dabei wird darauf geachtet, dass nur Bilder, des Types FILE_TYPE eingelesen und
4       * .svn dateien ausgelassen werden
5   * @param folder StartOrdner
6   * @param files Liste, in der die gefundenen Dateien gespeichert werden
7   */
8  private void getFilesInFolder(final LinkedList<File> files, final File folder) {
9    File[] currentFiles = folder.listFiles();
10   if (currentFiles != null) {
11     for (File currentFile : currentFiles) {
12       if (currentFile.isDirectory() && !currentFile.getName().contains(".svn")) {
13         this.getFilesInFolder(files, currentFile);
14       } else if (currentFile.getName().contains(FILE_TYPE)) {
15         files.add(currentFile);
16       }
17     }
18   }
19 }
```

```
20
21  private static final String FILE_TYPE = ".png";
```

```
1  /**
2   * Formatiert den String so, dass er unter UNIX und WINDOWS verwendet werden kann. Diese
3  Methode
4   * ersetzt den Backslash durch den Slash.
5   * @param input String
6   * @return  String
7   */
8  private String prepareString(final String input) {
9    return input.replace("\\", "/");
   }
```

CPU Auslastung verringern und Flaschenhälse finden (auf Windows)

von Sean Dieterle

1. ProcessExplorer herunterladen & starten
    - Zuerst müsst ihr euch den ProcessExplorer herunterladen: http://technet.microsoft.com/de-de/sysinternals/bb896653.aspx
2. JVisualVM starten
    - Die JVisualVM kommt direkt mit dem JDK. Sucht es also auf eurem PC. Z.B. unter: C:\Program Files\Java\jdk1.7.0_05\bin\jvisualvm.exe
    - Startet auch dieses Programm bevor ihr eure selbstgeschriebene Software ausführt.
3. Programm starten
    - Nun aktiviert ihr eurer Programm und wechselt in den ProcessExplorer.
4. Threads in ProcessExplorer analysieren
    - Dort seht ihr unterhalb von java.exe eurer Programm laufen (falls nicht, dann auf hierarchische Ansicht umstellen). Wenn ihr Server und Client gleichzeitig startet, kann es sein, dass ihr nicht auf Anhieb erkennt welcher der beiden Prozesse der richtige ist. Dafür bringt ProcessExplorer glücklicherweise ein Tool mit.
      Klickt oben auf das Fadenkreuz, haltet die Maustaste gedrückt, visiert die GUI eures Clients an und lasst los. Nun wird der Prozess markiert. Öffnet danach seine Übersicht, indem ihr doppelt darauf klickt und wechselt nun auf den Reiter Threads.
      Hier seht ihr alle Threads die in eurem Programm laufen mit der dazugehörigen CPU Auslastung. Schreibt euch die TID (Thread-ID) von den höchstausgelasteten Threads auf und wechselt zu JVisualVM.
5. Thread Dump in JVisualVM erstellen
    - Klickt hier in der linken Leiste per Rechtsklick auf euer Programm und wählt *Thread Dump* aus. Nachdem dieser erstellt wurde, könnt ihr eure selbstgeschriebenen Programme beenden.
6. Thread-ID umrechnen
    - Da JVisualVM die Thread-IDs als Hexadezimalzahl speichert, müsst ihr eure aufgeschrieben ID noch umrechnen. Einfach auf den Windows Taschenrechner gehen unter *Ansicht* auf *Programmierer* stellen und links *Dez* auswählen. Nun eure notierte Thread-ID eingeben und links auf *Hex* umstellen.
7. Flaschenhals finden
    - Jetzt könnt ihr wieder in den Thread-Dump wechseln und nach der Thread-ID suchen. Unterhalb dieser wird euch der Flaschenhals angezeigt, der die meisten Ressourcen verbraucht. Es wird die Datei (bzw. Dateien) sowie Zeilennummer(n) anzeigt, in der der Flaschenhals ist.
      Nun kennt ihr die (ungefähre) Stelle im Code und könnt diese überprüfen. Ungefähr habe ich deshalb geschrieben, da vielleicht nicht der konkrete Befehl in der Zeile das Problem ist, sondern z.B. eine while(true) Schleife, die unendlich oft bearbeitet wird. Falls dies der Fall sein sollte, genügt ein Thread.sleep oder Thread.wait und die Auslastung sollte sich gleich verringern.

Erweiterter Konsistenzcheck

von Alexander Alt

Wenn im Spiel das Ausführen einer Bewegung und das Ausführen eines Angriffs mit derselben Taste ausgeführt werden, (z.B. Rechtsklick mit der Maus) kann man den Konsistenzcheck erweitern. Der Vorteil der sich dadurch ergibt ist schnell ersichtlich: Klickt der User in der GUI, wird auf jeden Fall eine Aktion ausgeführt.
Bei Klick wird von der GUI ein Punkt der Map an die Client-Engine weitergegeben. Diese muss nun zuerst überprüfen ob auf dem angeklickten Punkt ein angreifbares Objekt steht. Wenn das der Fall ist wird der Befehl des Angriffs an den Server gesendet.
Ist dies nicht der Fall so soll eine Bewegung ausgeführt werden. Im einfachsten Fall ist das angeklickte Feld leer und der Held kann sich dorthin bewegen. Also wird ein Befehl mit der Bewegung mit Ziel des angeklickten Punktes an den Server gesendet.
Wenn nun das angeklickte Feld besetzt ist also nicht begehbar ist, soll trotzdem eine Bewegung ausgeführt werden. Nun muss noch nach einem geeigneten Ziel gesucht werden.
Wir haben das mithilfe der Breitensuche realisiert. Im Umkreis des angeklickten Punktes werden zunächst alle Punkte mit Abstand eins auf Begehbarkeit untersucht, ist hier keiner begehbar wird der Abstand erhöht. Der Abstand wird so lange erhöht bis mindestens ein begehbares Feld gefunden wurde. Wenn nun nur ein begehbares Feld gefunden wurde wird dieses als Ziel festgelegt und der Befehl der Bewegung an den Server gesendet.
Werden mehr als ein begehbares Feld gefunden, so muss Entschieden werden, welches der Felder man als Ziel der Bewegung nimmt. Da alle Felder die gleiche Entfernung zum ursprünglichen Ziel haben, sie sich also darin nicht unterscheiden, suchen wir den Punkt, der zum eigenen Helden den geringsten Abstand (Luftlinie) hat. (Alternativ: Geringster Weg, ermittelt über A*) Ist dieser bestimmt, so senden wir einen Befehl mit der Bewegung mit dem ermittelten Punkt als Ziel an den Server.

Realistische Sichtbereiche unter Berücksichtigung direkter Sichtlinien

von Tristan Rechenberger

Anstatt den Sichtbereich kreisförmig um die Einheiten aufzudecken ist es wesentlich realistischer mit wirklichen Sichtlinien zu arbeiten. D. h. Felder die hinter undurchsichtigen Objekten stehen in Nebel zu tauchen. Folgende Vorgehensweise liefert dieses Ergebnis für die einzelnen Einheiten.

- Bilde Liste mit allen Punkten im Kreis (mit Radius der Sichtweite) um die Einheit.
- Sortiere Liste nach absteigendem Abstand zum Mittelpunkt (also von außen nach innen)
- Für jeden Punkt in Liste, der noch nicht *überprüft* wurde:
    - Liste mit Punkten auf Luftlinie vom Mittelpunkt zum aktuellen Punkt
    - *Sichtbar* auf wahr setzten.
    - Für jeden Punkt der Luftlinie (von innen nach außen):
        - Falls *Sichtbar* wahr ist, füge Punkt Liste der Sichtbaren Punkte hinzu.
        - Falls Feld nicht durchsichtbar, also dort ein Hinderniss die Sicht versperrt:
            - *Sichtbar* auf falsch setzten.
        - Punkt als *überprüft* merken.
- Ausgabe der Liste sichtbarer Punkte

Diese Listen der einzelnen Einheiten lassen sich nun mit Hilfe eines HashSet schnell zusammenführen um eine Liste der Sichtbaren Felder ganzer Fraktionen zu bilden.

Für noch mehr Realismus lässt sich auch die Höhe der einzelnen Objekte mit einbeziehen. Dann können sich beispielsweise zwei Türme, zwischen welchen Bäume stehen, gegenseitig sehen. Dazu muss man sich beim Durchlaufen der Luftlinien die Höhe des bisher höchsten Objekts merken und mit der Höhe des Objekts des aktuellen Punktes auf der Luftlinie vergleichen.

Will man darüber hinaus auch, dass sich beispielsweise ein Turm und ein Held hinter einem Baum gegenseitig sehen, muss man sich den bisher größten Winkel arctan(Abstand zur sehenden Einheit / Höhenunterschied zur sehenden Einheit) merken. Ist der bisher größte Winkel kleiner als der des aktuellen Objekts auf der Luftlinie ist dieses Objekt für die sehende Einheit sichtbar.

Über Vererbung bei Javaprojekten

von Melanie Kuntze

Vererbung ist dazu da, um aus bereits bestehenden Klassen spezialisierte neue Klassen zu erstellen. Hierbei kann die neue Klasse die alte erweitern oder einschränken. Benutzt man zum Beispiel einen ähnlichen Aufbau einer Klasse sehr oft, oder muss man auf bestimmte Eigenschaften oft zugreifen, macht Vererbung Sinn. Zu beachten ist hierbei jedoch, dass dies gut überlegt werden muss. Der Quellcode wird oft sehr schnell unübersichtlich und schlecht nachvollziehbar. Um Übersichtlichkeit zu gewährleisten ist es gut, wenn die vererbten Klassen nicht in weit auseinandergelegenen Packages liegen.

Vererbung erleichtert die Programmierung, so kann man sich oft doppelte Codezeilen einfach sparen. Jedoch überlegt werden wann man wirklich auf Vererbung zurückgreifen möchte. Auch ist es Sinnvoll sich erstmals über die Implementierung von Interfaces Gedanken zu machen, da sie die Schnittstellen zwischen den einzelnen Programmbereichen bilden.
Wichtig ist auch, dass bei Vererbung immer ein Objekt der Eltern- oder Subklasse mittels super()-Konstruktor mit erzeugt wird. Demnach ist auch darauf zu achten, die Methoden der Elternklasse von denen der Unterklasse namentlich zu unterscheiden. Gibt es eine Methode mit gleichem Namen in der Unter- sowie auch Oberklasse, so überschreibt die Unterklasse die Methode der Oberklasse. Befindet sich jedoch der Methodenaufruf im Konstruktor der Oberklasse, si wird die Methode der Oberklasse aufgerufen.

Letztendlich ist zu sagen, dass Vererbung eine wichtige Sache in der objektorientierten Programmierung ist, man es aber nicht übertreiben sollte. Sinn macht es zum Beispiel, wenn man Items in einem Shop anbietet und alle den gleichen Aufbau haben, für jedes Bilder eingelesen werden müssen etc. Hier kann man sich sehr viel Arbeit durch Vererbung ersparen. Es wird eine Oberklasse item erstellt, von der alle anderen Items erben. Auch wird so die Benutzung der Items im Spiel vereinfacht. Implementiert man eine Klasse mit Items die nur einen einmaligen Effekt haben und eine Klasse der Items, die einen dauernden Effekt auf den Held ausüben, so kann man mittels Vererbung und der InstanceOf(Instance instance) Methode schnell feststellen zu welchem Typ das Item gehört und so die Programmierung deutlich vereinfachen.

Umrechnung von Kartenkoordinaten zu Pixelkoordinaten

von Tristan Rechenberger

```
1   /**
2    * Berechnet zu Kartenkoordinaten die entsprechenden Pixelkoordiaten.
3    * @param coord Kartenkoordinaten
4    * @return Pixelkoordinaten
5    * @author Tristan
6    */
7   public Point calcCoordToPixel(final Point coord) {
8
9
10   // Wie viele Pixel um x verschieben, wenn coord.x um 1 groesser
11     Point coordX1 = new Point(-(this.tileWidth / 2), -(this.tileHeight / 2));
12
13
14   // Wie viele Pixel um y verschieben, wenn coord.y um 1 groesser
15   Point coordY1 = new Point(+(this.tileWidth / 2), -(this.tileHeight / 2));
16
17
18   // Berechnung der Pixel-Koordinaten
19     int pixelPointx = this.centerPixel.x
20        + (coordX1.x * (this.centerCoord.x - coord.x))
21        + (coordY1.x * (this.centerCoord.y - coord.y));
22     int pixelPointy = this.centerPixel.y
23        + (coordX1.y * (this.centerCoord.x - coord.x))
24        + (coordY1.y * (this.centerCoord.y - coord.y));
25
    // Ausgabe der Pixel-Koordinaten
     return new Point(pixelPointx, pixelPointy);
   }
```

[Tristan Rechenberger, Gruppe 06]

7.2.2 Umrechnung von Pixelkoordinaten zu Kartenkoordinaten

```
1   /**
2    * Berechnet aus Pixelkoordinaten die entsprechenden Kartenkoordinaten.
3    * @param pixelPoint Pixelkoordinaten
4    * @return Kartenkoordinaten
5    * @author Tristan
6    */
7   public Point calcPixelToCoord(final Point pixelPoint) {
8
9
10   // Wie viele Pixel verschieben, wenn coord.x um 1 groesser
11     Point coordX1 = new Point(-(this.tileWidth / 2), -(this.tileHeight / 2));
12
13
14   // Wie viele Pixel verschieben, wenn coord.y um 1 groesser
15     Point coordY1 = new Point(+(this.tileWidth / 2), -(this.tileHeight / 2));
16
17
18   // Als Matrix:
19   // a b
20   // c d
21     int a = coordX1.x;
22     int b = coordY1.x;
23     int c = coordX1.y;
24     int d = coordY1.y;
25
26
27   // Determinante dieser Matrix
28     float determinante = (float) a * d - b * c;
29
30
31   // Wieviele Felder in x-Richtung verschieben, wenn pixelPoint.x um 1 groesser
     float pixelX1x = d / determinante;
```

```
32    // Wieviele Felder in y-Richtung verschieben, wenn pixelPoint.x um 1 groesser
33      float pixelX1y = -c / determinante;
34    // Wieviele Felder in x-Richtung verschieben, wenn pixelPoint.y um 1 groesser
35      float pixelY1x = -b / determinante;
36    // Wieviele Felder in y-Richtung verschieben, wenn pixelPoint.y um 1 groesser
37      float pixelY1y = a / determinante;
38
39
40    // Berechnung der Karten-Koordinate
41      int coordX = (int) (((float) this.centerCoord.x)
42          + (pixelX1x * (((float) this.centerPixel.x) - ((float) pixelPoint.x))) + (pixelY1x *
(((float) this.centerPixel.y) - ((float) pixelPoint.y))));
43      int coordY = (int) (((float) this.centerCoord.y)
        + (pixelX1y * (((float) this.centerPixel.x) - ((float) pixelPoint.x))) + (pixelY1y *
(((float) this.centerPixel.y) - ((float) pixelPoint.y))));


    // Ausgabe der Kartenkoordinaten
      return new Point(coordX, coordY);
    }
```