Tuxeditor - Handbuch

Inhaltsverzeichnis

1. Features

Tuxeditor ist ein Programm zur Erstellung und Bearbeitung von Kursen für Tuxracer 1.1.

Du hast zwei Möglichkeiten, mit Tuxeditor zu arbeiten:

1. Du kannst neue, eigene Kurse erstellen und bearbeiten.
2. Du kannst die Originalkurse untersuchen, verändern oder austauschen

Wenn du eigene Kurse erstellst, werden die Orginalkurse nicht angetastet. Das Problem ist, diese Kurse in Tuxracer anzumelden. Arthur J. Yarwood hat zwar einen Weg gefunden, aber die Ideallösung ist das noch nicht. Solange ich noch keine bessere Lösung gefunden habe, wird Tuxeditor sich konsequent nach den Vorgaben von Arthur richten, d.h. neue Kurse werden grundsätzlich unter .../tuxracer/courses/contrib eingerichtet, und damit die Kurse auch gefunden werden, müssen einige Skripte geändert bzw. hinzugefügt werden. Tuxeditor macht das zwar automatisch, aber wenn später jemand deinen Kurs installieren möchte, muss er ein wenig Hand anlegen.

Ein weiterer Nachteil: Wenn du deinen eigenen Kurs testen willst - und das ist erfahrungsgemäß sehr häufig der Fall -, musst du dich erst durch alle Orginalkurse hindurchklicken, denn eigene Kurse werden in der Liste hinten angefügt. Das kann sehr lästig werden. Zwar gibt es auch die Möglichkeit, über einen undefinierten Cup dorthin zu gelangen, aber es sieht so aus, als ließe dieses Verfahren nur einen Kurs pro Cup zu.

Wesentlich eleganter geht es zu, wenn du die Orginalkurse änderst oder innerhalb der Struktur der Originalkurse einen neuen Kurs (oder gleich einen neuen Pokal mit drei Kursen) einrichtest. Das klingt gefährlich, ist es aber nicht. Du brauchst vorher nur ein Backup des Originalkurses (oder -pokals) zu machen. Eigentlich nicht mal das, denn auch darum kümmert sich Tuxeditor. Der Vorteil: Die Kurse sind schneller erreichbar, und später können sie durch einfachen Tausch auch relativ leicht installiert werden. Allerdings sollte man hierbei etwas zurückhaltend mit den Änderungen sein, vor allem ist es nicht ratsam, die Pokalstrukturen durcheinanderzuwerfen. Aber es ist doch schon eine ganze Menge erreicht, wenn z.B. die drei Forest-Kurse in einem völlig neuen Gesicht und mit ganz anderen Anforderungen daherkommen, oder?

2. Installation

2.1 Linux

Ich empfehle, Tuxeditor als User zu installieren, denn auch bei einem Mehrbenutzersystem ist kaum zu erwarten, dass außer dir noch eine zweite Person mit Tuxeditor arbeiten wird.

1. Entpacke das Archiv an eine geeignete Stelle, z.B. /home/user/tuxeditor
2. Mache die Kylix-Library zugänglich, am besten durch folgenden Eintrag in deiner ~/.bashrc:

```
export LD_LIBRARY_PATH=/home/user/tuxeditor/lib
```

3. Melde dich erneut an, damit die .bashrc Wirkung zeigen kann, und starte das Programm (nach einer Neuanmeldung) mit

```
/home/user/tuxeditor/bin/tuxeditor
```

2.2 Windows

Zur Zeit gibt es noch keine Windows-Version. Ob es jemals eine Portierung nach Windows geben wird, hängt von dem Interesse ab. So gerne arbeite ich nicht unter Windows, dass ich ohne zwingenden Grund das Projekt zweigleisig fahren möchte. Vielleicht kann das auch ein anderer übernehmen; Voraussetzung ist eine Delphiversion >= 7.

3. Was ist ein Tuxracer-Kurs?

Wer sich mit der Erstellung von eigenen Kursen befasst, sollte wissen, wie ein Kurs aufgebaut ist und welche Auswirkungen die einzelnen Bestandteile auf das Erscheinungsbild und die „Renneigenschaften“ haben. Deshalb möchte ich zunächst eine Einführung in die Struktur von Tuxracer-Kursen geben.

3.1 Das Dreigespann: Höhenmodell, Texturierung und 3D-Objekte

Im Prinzip unterscheidet sich ein Tuxracer-Kurs nicht von den Szenerien anderer 3D-Programme; es sind immer 3 Dinge erforderlich, um eine Landschaft zu formen:

a) Das Höhenmodell. Ein Gitter von sogenannten „Höhenpunkten“ bestimmt die plastische Struktur der Oberfläche. Durch Interpolation werden die Zwischenwerte berechnet, so dass sich eine Struktur mit mehr oder weniger fließenden Höhenausprägungen ergibt. Je enger das Gitter angelegt ist, desto detaillierter und „glatter“ erscheinen die Erhebungen. Teilweise wird diese aufwendige Rechenarbeit von 3D-Grafikkarten übernommen. In vielen 3D-Programmen wird das Höhenmodell auch als „Mesh“ bezeichnet.

Tuxracer verwendet ein elegantes und einfach zu handhabendes Verfahren für das Höhenmodell. Es wird eine Graustufen-Bitmap angelegt, in der jedes Pixel einen Höhenpunkt darstellt. Die Werte schwanken von 0 (= schwarz, tiefste Stelle) bis 255 (= weiß, höchste Stelle). Durch grafische Verfahren, z.B. durch die Verwendung weicher Pinsel in einem Grafikprogramm, können sehr gut gleitende, harmonische Höhenausprägungen erzeugt werden. Die Bezeichnung der Höhenbitmap ist immer gleich: elev.png.

b) Die Texturierung. Das Höhenmodell ist abstrakt - sichtbar wird die Landschaft erst, indem sie mit Texturen belegt wird (quasi „tapeziert“ wird). Bei den Texturen handelt es sich um monochrome Farbflächen oder Bilder, die auf das Höhenmodell projiziert werden. Bei echten Landschaftsdarstellungen (z.B. in einem Flugsimulator) können die Texturen sogar aus realistischen Fotos bestehen (Satellitenaufnahmen).

Auch die Texturierung ist in Tuxracer recht einfach gelöst: Wie beim Höhenmodell sind die Informationen in einer Bitmap gespeichert, die natürlich deckungsgleich mit der Höhen-Bitmap sein muss. Tuxracer stellt eine Reihe von Texturen (Terrains) in Form von Png-Bitmaps zur Verfügung: Schnee, verschiedene Eissorten, Gras usw. Welche Textur an welcher Stelle erscheinen soll, wird durch eine Farbcodierung festgelegt. Man braucht also nur die entsprechende Farbe auf die Bitmap zu malen, muss aber im Gegensatz zur Höhenstruktur unbedingt eine harten Pinsel benutzen, weil sich bei einem weichen Pinsel Zwischentöne mit nicht interpretierbaren Farben ergeben würden.

c) Die Objektbelegung. Auf dem Terrain werden schließlich die dreidimensionalen Objekte wie Bäume, Heringe, Häuser usw. angebracht. Ein Objekt besteht aus einer Beschreibungsdatei, die alle erforderlichen Angaben für die Formgebung enthält, sowie den Objekttexturen. So werden z.B. bei einem Haus die Fenster und Türen meistens nicht plastisch herausgebildet, sondern einfach mit einer Textur auf die Wand „gemalt“.

Beim Tuxracer 0.61 war es noch einfach, die Objekte anzubringen. Dazu gab es eine dritte Bitmap (trees.png) auf die man die Objekte in Form farbiger Pixel aufzeichnete. Allerdings waren die Objekte dort noch einfach aufgebaut. Es gab nur wenige davon, und bei allen handelte es sich um sogenannte „Punktobjekte“, die mit „einem Bein“ auf dem Schnee oder Eis standen.

Tuxracer 1.1 bietet wesentlich mehr Möglichkeiten. Zum einen gibt es eine Menge verschiedener Objekte; zum anderen können praktisch alle Objekte in den drei Dimensionen skaliert bzw. um drei Achsen rotiert werden. Außerdem werden sie nicht automatisch mehr weniger genau auf die Oberfläche gesetzt, sondern sie lassen sich höher oder tiefer stellen, z.B. teilweise in den Schnee versenken. Bei diesen Variationsmöglichkeiten ist es mit einem schlichten Aufklicken von Objektpunkten nicht getan. Deshalb werden die Objekte in einer Textdatei (items.tcl) mit allen erforderlichen Parametern aufgelistet. Theoretisch könnte man mit einem Texteditor diese Datei erstellen, aber praktisch stellt sich das als ein aussichtsloses Unterfangen heraus. Es ist nicht nur unwahrscheinlich mühsam, die vielen Parameter in richtiger Reihenfolge anzugeben, sondern man hat nicht mal eine vernünftige Möglichkeit, das zu überprüfen. Als besonders kritisch erweist sich nämlich die Höhenpositionierung. Schon bei geringfügigen Abweichungen vom Höhenlevel schwebt das Objekt unsichtbar in der dunstigen Luft oder es liegt tief im Schnee begraben. Was man nicht sieht, kann man nicht zurechtrücken ...

Hier beginnt die Arbeit von Tuxeditor. Die Objekte können damit wie bei Tuxracer 0.61 aufgeklickt werden, wenn auch in anderer Weise, denn die Objekte sollen außerdem noch bequem skaliert und rotiert werden können. Auch eine Feinjustierung in allen drei Dimensionen ist möglich. Dass die meisten Objekte von Tuxracer 1.1 zur Verfügung stehen, versteht sich von selbst. Zwei Objektarten sind zur Zeit noch nicht realisierbar, und zwar die animierten Objekte (z.B. die Lawinen in Forest 2 oder die fahrenden Autos in Swiss) sowie die speziellen Soundobjekte (die ich selber aber noch nie bewusst wahrgenommen habe).

3.2 Die Geometrie eines Kurses (course.tcl)

Man kann sich einen Kurs als eine Art Brett vorstellen, auf dem die Landschaft modelliert wird. Es leuchtet ein, dass zunächst die Länge und Breite festgelegt werden müssen. Es ist keineswegs so, dass Länge und Breite von den Bitmaps bestimmt werden, sondern die Werte lassen sich weitgehend unabhängig davon festlegen „Weitgehend“ deshalb, weil ein großer Kurs entsprechend große Bitmaps benötigt, um eine hinreichende Detailgenauigkeit zu erreichen. Prinzipiell sind die Kursabmessungen aber unabhängig von den Bitmapabmessungen. Als günstig hat sich ein Verhältnis von 1 : 2 herausgestellt: Um einen Kurs von 1000 Längeneinheiten und 100 Breiteneinheiten zu beschreiben, werden dazu Bitmaps mit 500 x 50 Pixeln benötigt.

Wie alle anderen geometrischen Parameter sind Länge und Breite in der Datei course.tcl festgelegt (length und width). Im engen Zusammenhang mit Länge und Breite stehen die Parameter play_length und play_width, die immer etwas kleiner ausgelegt sind. Play_width begrenzt den „befahrbaren“ Raum in der Breite, so dass Tux nicht ganz an die Seitenränder gelangen kann. Man kann zwar nicht abstürzen, aber es wirkt doch unschön und unrealistisch, wenn man seitlich nur noch eine neblige Leere erblickt. Eine zusätzliche, auf bestimmte Stellen bezogene Begrenzung ist durch das Objekt boundary möglich. Mehr dazu weiter unten.

Wichtiger noch als die Breitenbegrenzung ist die Längenbegrenzung play_length. An dieser Stelle endet nämlich das Rennen: Die Zeitzählung stoppt, und Tux springt im eleganten Bogen den Zielhang hinunter. Zweckmäßigerweise wird hier das Zieltor plaziert. Für den Zielbereich sollte man etwa 40 - 50 Einheiten bereitstellen.

Bleiben noch zwei Parameter. Die Bitmap erlaubt eine Höhenmodellierung in 256 Stufen, was aber nicht bedeutet, dass die Höhenverhältnisse damit in absoluten Werten festgelegt werden. Wie Länge und Breite lässt sich auch die Höhe skalieren, und zwar mit dem Parameter elev_scale. Übliche Werte liegen bei etwa 20 - 30, die sich daraus ergebenden Landschaften können als „hügelig“ bezeichnet werden. Um richtiges Gebirge zu erzeugen, muss die Skalierung heraufgesetzt werden (Werte > 50). Allerdings ergibt sich dadurch ein unangenehmer Nebeneffekt: Je höher die Skalierung, desto schwieriger wird es, glatte Pistenverläufe zu erreichen. Es kommt zu Unebenheiten und ungewollten Abstufungen, die sehr viel Feinarbeit erfordern, um sie auszugleichen.

Damit schließlich unser Tux sich ohne Motorkraft bewegen kann, muss das „Brett“ noch geneigt werden. Der Neigungswinkel wird im Parameter angle festgelegt. Es leuchtet ein, dass der Neigungswinkel einen erheblichen Einfluss auf die Geschwindigkeit und die Rasanz des Kurses hat. Bei 15° geht es recht gemächlich zu, mit 30° oder mehr erzeugen wird Hochgeschwindigkeitskurse. Der Neigungswinkel ist übrigens kein theoretischer Wert zur Geschwindigkeitssteuerung, sondern wird geometrisch exakt umgesetzt, was man auch optisch sehr gut erkennen kann, wenn man Tux zur Seite steuert.

Die Auswirkungen des Neigungswinkels übertreffen die Auswirkungen der Höhenskalierung um ein Vielfaches. Bei einem normalen Kurs von 1000 Längeneinheiten beträgt der Höhenunterschied zwischen Start und Ziel etwa 300 - 400 Einheiten, während der Höhenunterschied zwischen einem Berggipfel und der Talsohle allenfalls mit 20 - 30 Einheiten in Erscheinung tritt. Diese Tatsache kann eine wichtige Rolle spielen, wenn man „Kletterpartien“ oder Abfahrtshänge in seinen Kurs einbaut.

3.3 Die Geometrie der Objekte (items.tcl)

Damit ein Objekt an der richtigen Stelle erscheint, sind drei Angaben unerlässlich: die x/y-Koordinaten sowie die Höhenangabe. Besonders die Höhenangabe erfüllt bei den Positionsparametern eine wichtige und kritische Funktion. Es ist nämlich nicht so, dass die Objekte automatisch auf den Untergrund gesetzt werden, sondern entsprechend der Terrainhöhe an der jeweiligen Stelle und der durch die Neigung des Kurses bestimmten, mit dem Kursverlauf abnehmenden „Basishöhe“ müssen die Objekte genau eingestellt werden. Darüber hinaus kann es sinnvoll sein, Objekte ein wenig zu versenken, z.B. um bei einer Brücke die Fahrbahnhöhe dem Gelände anzupassen. Die Geometrie der Objekte wind in items.tcl eingestellt; für die Position sind die Parameter {-position} zuständig.

Ein weiterer Aspekt der Objektgeometrie ist die Skalierung. Fast alle Objekte lassen sich beliebig spreizen oder schrumpfen, und da das noch in den drei Dimensionen möglich ist, gibt es eine Unmenge an Gestaltungsmöglichkeiten. Hochhaus gefällig? Einfach die Höhe eines Hauses verdoppeln. Dass dieser „Turm“ immer noch mit nur zwei Stockwerken daherkommt, fällt im Eifer des Rennens kaum auf (siehe Originalkurs Swiss 2). Oder der Stein (boulder): Groß, aber flach gehalten, stellt er eine steinige Bodenerhebung da, in Normalgröße ein Hindernis. Sogar eine Steinwand lässt sich damit produzieren. Die

Skalierung wird im Parameterblock {-scale} erfasst.

Während die Positionsparameter immer vorhanden sein müssen, sind die Skalierungsparameter optional. Ohne explizite Angabe wird automatisch mit den Werten {1.0 1.0 1.0} skaliert. Doch was bedeutet diese Standardgröße? Es bedeutet auf jeden Fall nicht, dass alle unskalierten Objekte gleich groß erscheinen, sondern sie werden so skaliert, dass sie aus der Sicht von Tux einigermaßen realistisch erscheinen. Ein Baumstamm ist etwas so dick wie der Pinguin, ein Haus so groß, dass Tux, wenn die Tür offen wäre, bequem hineinfahren könnte. Diese Standardgrößen machen natürlich nur Sinn, wenn die Objekte in unmittelbarer Nähe der Rennstrecke liegen. „Fernere“ Objekte sollten entsprechend kleiner gestaltet werden, und wenn man den Eindruck der Nähe noch vertiefen will, kann man größer skalieren (siehe die Bäume in den Forest-Kursen - hier wird der Tux ganz klein).

Schließlich die Rotationsparamter {-rotation}. Auch sie sind optional und erlauben es, die Objekte beliebig zu drehen. Ein Baumstamm kann aufrecht als langer Stumpf im Gelände stehen, er kann auch quer über den Weg gelegt werden. Selbst bei Objekten wie Bäumen macht die Drehung Sinn. Ohne Drehung sähen alle Bäume gleich aus; deshalb werden sie meistens nach dem Zufallsprinzip um einen beliebigen Winkel um die „Hochachse“ gedreht.

Bleibt noch zu klären, in welcher Reihenfolge die Parameter in items.tcl angegeben werden. Prinzipiell gilt die Reihenfolge x, z, y, wobei z die Höhe angibt. Auf den ersten Blick etwas irritierend wirkt sich diese Reihenfolge bei den Rotationsparametern aus, obwohl sie auch hier konsequent eingehalten wird. Wenn ich den ersten Wert (x) verändere, neigt sich das Objekt nicht etwa nach links oder rechts, sondern nach vorne oder hinten, in Fahrtrichtung gesehen. Es wird nicht in x-Richtung, sondern um die x-Achse, die quer zur Fahrbahn verläuft, gedreht!

Schließlich die Vorzeichen der Parameter. Höhenpositionen erscheinen fast durchweg negativ. Nur ganz am Anfang des Kurses und bei Höhen über 50% elev_scale können positive Werte auftreten. Aber schon am Startpunkt ist man in der Regel im negativen Bereich, wegen der kräftigen Auswirkung der Kursneigung.

x-Positionen sind ausschließlich positiv, ausgehend vom linken Rand des Kurses (vom rechten Rand der Bitmap). y-Positionen sind ausschließlich negativ, ausgehend vom Anfang des Kurses (vom oberen Rand der Bitmap).

Alle Scale-Parameter sind naturgemäß positiv.

Besonders schwierig (und undurchsichtig) wird es bei den Rotationsparametern. Da es sich um Winkel handelt, könnte man im Prinzip durchweg mit positven Werten arbeiten, und es scheint auch so zu sein, dass sich die Winkel +270° und -90° entsprechen, aber um die Abweichung von der Standardrichtung deutlich zu machen, ist es günster, mit zwei Richtungen zu arbeiten,a lso Drehungen von -180 bis +180 zuzulassen. Im einzelnen:

Positive x-Werte drehen in Fahrtrichtung (nach unten auf der Bitmap), negative x-Werte in Richtung Start. Positive y-Werte drehen, in Fahrtrichtung gesehen, nach rechts (nach links auf der Bitmap), negative y-Werte entsprechend anders herum. Positive z-Werte drehen, von oben gesehen, nach links (hier stimmen die Sichtweisen von Kurs und Bitmap überein).

3.4 Objektarten

Man könnte auf den Gedanken verfallen, die Objekte in Bäume, Bauwerke, Fahrzeuge usw. einzuteilen, aber das ist nicht besonders sinnvoll. Wichtiger ist die Frage, wie man mit den Objekten bei der Gestaltung von Kursen umgeht. Von daher drängt sich eine andere Unterscheidung auf, nämlich die nach dem Grundriss:

Punktobjekte. Sie haben einen vernachlässigbar kleinen Grundriss und lassen sich daher recht einfach positionieren. Dabei spielt es keine Rolle, wenn das Objekt wie bei einem Baum oben doch eine beachtliche Abmessung haben kann. Entscheidend ist, dass der Baum nur mit dem Stamm den Boden berührt und infolgedessen nicht den Schrägungen angepasst werden muss. Auch Objekte mit geringer Ausdehnung (wie Flaggen) kann man noch zu den Punktobjekten zählen.

Linienobjekte. Davon gibt es nicht sehr viele, z.B. die Baumstämme der Forestkurse, aber auch die unsichtbaren „boundaries“. Hier muss die Länge beachtet und in den meisten Fällen der Umgebung angepasst werden. Auch die Rotationswinkel sind sehr wichtig.

Flächenobjekte. Dazu zählen typischerweise die Häuser, die eine zum Teil beachtliche Fläche bedecken.

Diese Fläche muss freigehalten werden, sonst ragt evtl. aus dem Dach eine Tannenspitze hervor. Dass Skalierung und Rotation enorme Auswirkungen haben, ist leicht nachvollziehbar. Flächenobjekte sind deshalb nicht immer einfach anzubringen. Lies dazu bitte die Hinweise unter 7.2. Es gibt allerdings auch Flächenobjekte, die relativ unkritisch sind, z.B. die hübsch-hässlichen „Overhangs“. Die kann man nach Belieben drehen oder versenken und somit der Landschaft anpassen

3.5 Die Terrains

Die Terrains wurden oben vereinfachend als „Tapete“ bezeichnet, und zweifellos ist es zunächst die optische Wirkung, an die wir denken. Aber in Tuxracer gibt es noch weitere, wichtige Merkmals der Terrains:

1. Sie bestimmen die Gleiteigenschaften. Auf Eis kann man hohe Geschwindigkeiten erreichen, andererseits aber schlecht abbremsen; auf Gestein geht es quälend langsam voran.
2. Sie erzeugen ein Gleitgeräusch, man hört es förmlich blubbern, wenn man durch Matsche fährt.
3. Einige Terranis hinterlassen Spuren, vor allem die weichen Terrains wie Schnee oder Sumpf. Es gibt aber auch Schneetexturen, die diesbezüglich spurlos reagieren.
4. In sehr glatten Terrains wie Eis spiegelt sich Tux.
5. Bei einigen Schneetexturen wirbelt der Pinguin Schneeflocken auf ...

All diese Effekte sind in Tuxracer fest vorprogrammiert; man braucht sich darum also nicht zu kümmern. Aber man sollte darauf achten, dass die Terrains einen wichtigen Beitrag zur „Befahrbarkeit“ eines Kurses leisten.

3.6 Umgebunggestaltung

Um noch einmal auf das „Modellbrett“ zurückzukommen: Wenn wir so ein Modell in den Raum stellen, dann ist es nicht ganz einfach, uns in die Modelllandschaft hineinzuversetzen - es gibt zu viel Störendes drumherum, das Modell wirkt isoliert. Ganz anders, wenn wir den Kurs in eine Art Box einbauen, ihn also rundherum mit kulissenartigen Landschaftsatrappen umgeben. Tuxracer verwendet dafür 6 Bitmaps, die - wie sollte es anders sein - Berglandschaften darstellen.

Um die Illusion perfekt zu machen, muss noch für die richtige Beleuchtung bzw. eine geeignete Luftkonsistenz gesorgt werden. In Tuxracer lassen sich drei Situationen einstellen (Sonne, Sonnenuntergang und Dunst). Jede Beleuchtungssituation wird wiederum durch eine Reihe von Parametern erzeugt (Lichtfarbe, Standort und Einfallswinkel des Lichtes usw.) Und damit auch alles zusammenpasst, steht für jede dieser Beleuchtungssituationen ein kompletter Satz von jeweils 6 Landschaftsbildern zur Verfügung.

Bei der Erstellung eigener Kurse wird man in der Regel auf die vorhandenen Ressourcen zurückgreifen, aber es ist ein Anliegen von Tuxeditor, diese möglichst flexibel zusammenstellen zu können. Die für die Umgebung relevanten Parameter befinden sich in den Dateien sunny_light.tcl, sunset_light.tcl und foggy_light.tcl. Sie sind jedoch, ebenso wie die Umgebungsbilder, nicht direkt auf den einzelnen Kurs bezogen, sondern gelten für die 3 Kurse, die zu einem Cup gehören.

4. Die Struktur von Tuxracer und die Einbindung von Kursen

Computerspiele haben ihre eigene Struktur, und Tuxracer macht da keine Ausnahme: Bevor man sich schwierigeren Aufgaben widmen kann, muss man sich dafür erst mal qualifizieren. Für Tuxracer heißt das konkret: Du kannst die Rennen des 2. Cups („Arctic“) erst starten, wenn du die Rennen des Beginnercups erfolgreich absolviert hast, usw. Das führt

a) zu einem benutzerorientierten Ergebnisprotokoll mit der unvermeidlichen „Highscore-Liste“,
b) zu einer Gliederung der Programmstruktur in Cups und Kurse, mit einer streng sequentiellen Ordnung.

Während der erste Aspekt für den Kursmacher relativ bedeutungslos ist, zieht der zweite Aspekt große Probleme nach sich. Wo soll der eigene Kurs untergebracht werden? Eigentlich sind wir schon zufrieden, wenn der eigene Kurs nebenher, praktisch „außer Konkurrenz“ gestartet werden könnte, nur zum Vergnügen, frei von Gewinnerstress. Aber auch dazu muss die Möglichkeit einer Einbindung vorhanden sein. Es gibt schwache Hinweise darauf, dass die Programmierer von Tuxracer daran gedacht haben, aber eine echte Schnittstelle, die ohne Tricks zum Ziel führt, ist nicht vorhanden. Ich werde darauf noch zurückkommen, möchte aber zuerst die Struktur ein wenig erläutern.

4.1 Bestandteile von Kursen

Jeder Kurs enthält folgende Elemente:

- die beiden bereits erwähnten Bitmaps elev.png und terrain.png mit den Höhen- und Bodenmodellen,
- die zentrale Kursdatei course.tcl mit den Angaben zur Geometrie des Kurses,
- die Objektliste items.tcl mit den geometrischen Angaben zu den angebrachten Objekten,
- das (eher nebensächliche) Vorschaubild preview.png.

Diese Dateien sind im sogenannten „Trackordner“ untergebracht. Für jeden Kurs existiert ein solcher Ordner. Ich verwende den Begriff „Trackordner“, um eine Verwechslung mit dem globalen Ordner courses, worin alle Kurse und Cups enthalten sind, zu vermeiden.

4.2 Bestandteile von Cups

Jeweils 3 Kurse bilden einen Cup, der im „Cupordner“ untergebracht ist. Somit enthält ein Cuprordner zunächst die drei Trackordner, daneben aber noch eine Reihe von Dateien, die für für alle Kurse dieses Cups gültig sind. Im wesentlichen lassen sich vier Gruppen von Dateien unterscheiden:

1. Die Pokaldateien (trophy.tcl ...). Sie regeln das Zwischenspiel zwischen den Rennen, signalisieren Erfolg und Misserfolg und sind für Zusatzkurse eher belanglos.

2. Die Umgebungsdateien. Dabei handelt es sich um die 3 Beleuchtungsdateien (sunny_light.tcl, sunset_light.tcl und foggy_light.tcl) sowie die 18 zugehörigen Umgebungsbitmaps.

3. Die Objekt-Definitions-Dateien. Das sind zwei äußerst wichtige Dateien: objects.tcl gibt an, welche Objekte für die 3 Rennen dieses Cups zur Verfügung stehen und welche grundsätzlichen Eigenschaften sie haben. Wenn der Zugriff auf diese Datei in irgendeiner Weise gestört ist, lässt sich der Kurs nicht starten. Parallel dazu gibt es die Datei object_textures.tcl, die ähnlich arbeitet wie objects.tcl, nur auf die Objekttexturen bezogen ist (nicht zu verwechseln mit den Geländetexturen).

4. Die Objekt-Erzeugungs-Dateien. Die in objects.tcl definierten Objekte müssen irgendwo erzeugt werden, und das geschieht in der Regel mit Hilfe von zwei Dateien für jedes Objekt. Die dreidimensionale Form wird durch die Zahlenwerte in objectname.obj.strip gebildet, die Texturierung durch das Bild in objectname.png.

Noch einmal, in aller Deutlichkeit: Alle Dateien und Einstellungen innerhalb des Cupordners beziehen sich auf alle Kurse dieses Cups. Es ist z.B. nicht möglich, jedem einzelnen Kurs eine eigene Objektressource zuzuweisen, dasselbe gilt für die Lichtverhältnisse. Wenn ich einen Kurs mit anderen Objekten ausstatten will, muss ich auch die evtl. schon bestehenden Kurse dieses Cups aufs neue mit Objekten bestücken!

4.3 Gemeinsame Ressourcen? Leider ein Durcheinander!

Die Ideallösung sähe so aus: Alle Objekte wären in einem speziellen Objektordner untergebracht, und jeder Kurs könnte auf die gemeinsame Ressource zugreifen. Ob in diesem Objektordner eine große, alle Objekte umfassende Definitionsdatei objects.tcl existieren würde, oder verschiedene kleinere mit jeweis einer sinnvollen Auswahl, ist eher nebensächlich. Ferner wären alle Umgebungsdateien in einem gemeinsamen Environment-Ordner untergebracht, und jeder Kurs könnte mit einer eigenen, passenden Umgebung ausgestattet werden.

Tatsächlich sind die Ressourcen aber verteilt: Die Swiss-Kurse verwenden z.B. andere Objekte als die Forest-Kurse, und wo es (an wenigen Stellen) zu Gemeinsamkeiten kommt, sind Objekte und Definitionen doppelt vorhanden. Die Gründe dafür glaube ich inzwischen zu kennen; im Abschnitt „Intern“ habe ich sie aufgeführt.

Um die Objektressourcen möglichst flexibel für eigene Kursentwicklungen verwenden zu können, müssen wir die Zugriffsmechanismen kennen. Jeder Kurs erwartet im übergeordneten Verzeichnis, dem Cupordner, die Datei objects.tcl. (Dasselbe gilt für object_textures.tcl, aber da die Verhältnisse vergleichbar sind, gehe ich nur noch auf objects.tcl ein.) Damit steht schon fest, dass ein Ausbrechen aus der gemeinsamen Cupbindung nicht möglich ist. Normalerweise enthält objects.tcl die Auflistung der Objekte, die sich im gleichen Ordner befinden müssen. Allerdings gibt es einen Ausweg: objects.tcl kann anstatt der Liste auch einen Hinweis auf eine andere Ressource enthalten, die sich z.B. in einem parallelen Cup befindet. Damit scheint auf den ersten Blick der Zugriff auf alle Objektressourcen gesichert zu sein. Aber diese Freiheit können sich nur Kurse ohne eigene Ressource leisten, das sind lediglich die Kurse im Beginner-Cup sowie

die Contrib-Kurse. Die anderen Kurse würden nämlich den Zugang zu ihren eigenen Objekten versperren, weil die Objektliste verloren ginge.

Dieses Dilemma macht die Nachteile der Originalstruktur von Tuxracer sehr deutlich. Um hier alle Möglichkeiten auszuschöpfen, müssten die Dateien objects.tcl umbenannt werden, etwa in forestobjects.tcl. Dann könnte der Zugriff der Forest-Kurse dadurch gesichert werden, indem in objects.tcl nichts weiter steht als „source forestobjects.tcl“. Die Kurse der anderen Cups müssten zusätzlich eine Instruktion zum Verzeichniswechsel enthalten. Warum also nicht diese Änderung vornehmen? Es gibt einen einfachen Grund. Wir als Kursprogrammierer hätten damit keine Probleme, aber der Kurs soll ja auch von anderen Leuten möglichst einfach zu installieren sein. Natürlich könnte man ein Skript mitliefern, das diese Schritte automatisiert, aber ich kann mich im Augenblick noch nicht dazu entschließen. Außerdem wäre das nur eine halbe Sache. Wenn man schon Änderungen an der Originalstruktur vornimmt, kann man auch gleich gründlich vorgehen und alle Objekte dorthin packen, wohin sie gehören: in einen gemeinsamen Objektordner. Später vielleicht.

Mit den Umgegungsdateien verhält es sich ähnlich. Da diese Dateien aber relativ kurz sind, können sie entsprechend den gewünschten Situationen neu generiert werden. Hier gibt es also keine Einschränkungen - bis auf die, dass die Einstellungen für den gesamten Cup gelten.

4.4 Der Common-Ordner - doch etwas Gemeinsames

Neben den Cupordnern existiert ein Ordner, der allen Kursen zur Verfügung stehe, weil Tuxracer automatisch zuerst hier nachschaut. Es ist der Ordner common, parallel zu den Cupordnern. Was steckt drin? Zunächst einmal die Geländestrukturen, was durchaus sinnvoll ist. Noch besser (weil übersichtlicher) wäre es gewesen, die Terrains in einem eigenen Ordner zu verwalten, aber wir wollen nicht übertrieben kritisch sein. Ferner finden wir in common einige allgemeine Bilder sowie einige grundlegende Objekte wie Heringe, Flaggen, verschiedene „Pads“, Startrampe, Zieltor usw. Das, was wirklich in allen Kursen benötigt wird, ist hier vorhanden. Wenigstens das. Zur Ausschmückung eines Kurses reichen die Objekte allerdings nicht, auch wenn eine einzige Baumsorte immerhin eine monokulturelle Aufforstung erlaubt.

4.5 Kurse in die Struktur einbinden

Diese Kapitel ist noch leer, und ich muss gestehen, dass ich noch keinen günstigen Weg gefunden habe, Kurse für 1.1 einzubinden. Wohlgemerkt, es geht nicht darum, wie der „Kursmacher“ damit umgeht (für den wird es kein Problem sein), sondern es geht um die späteren Benutzer der Kurse, die sich nicht ohne weiteres in der Kursstruktur auskennen, und denen man nicht zumuten sollte, irgendwelche Dateien zu verändern.

Alles deutet darauf hin, dass es für diesen Zweck ein Installationsprogramm geben sollte, quasi ein Programm, mit dem der Benutzer „sein“ Tuxracer nach Belieben (und narrensicher) zusammenstellen kann. Wie das genau aussieht, kann ich zur Zeit noch nicht sagen.

5. Einige Workarounds mit Tuxeditor

Bevor ich auf die eigentliche Bedienung des Programms eingehe, möchte ich einige Arbeitsweisen vorstellen.

5.1 Kurse von Tuxracer 0.61 übernehmen.

Das ist ein sehr guter Weg, zu eigenen Kursen zu kommen. Erforderlich ist das Konvertierungsprogramm Tuxcourser (http://tuxcourser.sourceforge.net). Das Programm ist sehr leistungsfähig und bequem zu bedienen, leider aber nicht ganz so bequem zu installieren, da es sich um ein Java-Programm handelt. Ich habe die Installation von Tuxcourser weiter unten beschrieben.

Die Kurse werden mit Tuxcourser sehr vollständig übertragen und vor allem sauber in die Contrib-Struktur von Tuxracer 1.1 eingefügt. Natürlich kann das Programm nur das herausholen, was in den Kursen von 0.61 steckt. Es gibt nur drei Terrains, und - neben den Heringen und Flaggen - als Gestaltungsobjekte nur schneebedeckte Tannen. Das liegt daran, dass Tuxracer 1.1 nicht die „toten Bäume“ und Büsche aus 0.61 kennt. Diese werden durch eine andere Art von Tannen dargestellt, die sich allerdings kaum von den „normalen“ Tannen unterscheiden.

Der entscheidenden Vorteil dieser Vorgehensweise: Man kann schon mal für eine vernünftige Höhenstruktur sorgen und diese mit Tuxracer 0.61 gründlich testen. Auch eine gewisse „Vortexturierung“ könnte sinnvoll sein. Allerdings lohnt es sich nicht, schon Objekte anzubringen. Diese können nach der Konvertierung mit

Tuxcourser mindestens ebenso komfortabel mit Tuxeditor auf den Kurs gesetzt werden, wobei dann eine viel größere Auswahl zur Verfügung steht.

5.2 Eigene Kurse „from the scratch“ erstellen

Die komplette Bearbeitung von Neukursen ist zur Zeit mit Tuxeditor nicht möglich. Die Terrains können zwar schon aufgemalt werden, aber die Bearbeitung des Höhenmodells muss noch „extern“ erledigt werden. Im Prinzip ist es egal, mit welchem Grafikprogramm elev.png editiert wird; unter Linux wird man jedoch nach wie vor auf Gimp zurückgreifen. Gimp kann übrigens zusammen mit elev.png bequem von Tuxeditor aus gestartet werden.

Besser wäre es, elev.png und terrain.png gemeinsam als Layer zu starten, so wie es unter 0.61 mit den drei RGB-Dateien geschieht. Im Augenblick habe ich noch Probleme mit dem Script-fu, und ich weiß auch noch nicht, ob man Gimp mit diesem Script-fu starten kann. Sollte das möglich sein, gibt es kaum noch einen Grund, den Kurs vorher für 0.61 zu erstellen.

5.3 Die Objekte der Originalkurse kennenlernen

Auch das kann spannend und aufschlussreich sein: einfach mal gucken, wie es die Originalkurse machen. An einigen Stellen kann man wirklich staunen, welche Wirkung durch geschickte Einstellung der Objektparameter zu erzielen ist. Manches findet eine einfache Erklärung, z.B. die Tatsache, dass die einzige Verzweigung in Mountain 2 nicht erreichbar ist. Oder es wird deutlich, dass die schönsten Tunnel in den Glacier-Kursen keine Tunnel sind, sondern spezielle Bögen („arches“). Oder dass man durch Drehung um 90° oder 180° den Eindruck erzielen kann, es handle sich um verschiedene Häuser. Es lohnt sich - man lernt eine Menge dabei.

In diesem Zusammenhang gleich ein wichtiger Hinweis: Es ist ungefährlich, einen Originalkurs zu öffnen. Zum einen wird von Originalkursen beim ersten Öffnen automatisch ein Backup angelegt (im Ordner _backup), zum anderen werden die Objekte ziemlich vollständig erfasst und weiterverwendet. Nur die animierten Objekte und die speziellen Soundobjekte gehen verloren. Wenn man von Tuxeditor aus einen Originalkurs startet, erkennt man nur am Fehlen der Animationen, dass dabei eine komplett neu generierte items.tcl ihren Dienst tut.

5.4 Originalkurse verändern

Warum soll man die eigenen Kurse nur außerhalb des „normalen Betriebes“ starten können? Tatsache ist, dass man die 18 Kurse irgenwann auswendig kennt und mal eine andere Landschaft sehen möchte, evtl. auch mit anderen Leistungsanforderungen. Dazu bietet sich an, einzelne Kurse oder - besser noch - ganze Cups auszutauschen, was am bequemsten innerhalb der Orginal-Struktur geht. Wie weit man die Änderung treiben möchte, ist freigestellt. So ist es z.B. sinnvoll, auf den vorhandenen Bitmaps aufzubauen (abgesehen von den Bitmaps in Mountain, die sind angesichts der spärlichen Ausstattung etwas überdimensioniert). Man kann das Höhenmodell abändern, die Terrains anders gestalten, insbesondere aber die Objekte komplett neu anbringen. Vor allem aber lassen sich diese Kurse ohne viel Aufwand in die Pokalstruktur einbinden; lediglich die Anforderungsprofile müssen in course_idx.tcl angepasst werden. Ob sich allerdings auch das Rennen gegeneinander (vs opponents) noch durchführen lässt, kann ich im Augenblick noch nicht sagen. Sofern zumindest die elev.png unverändert übernommen wird, dürfte es diesbezüglich keine Probleme geben.

5.5 Meine bevorzugte Arbeitsweise

Obwohl ich zum Zeitpunkt, wo ich dieses Manual schreibe, noch keine großen Ergebnisse vorzeigen kann, hat sich doch schon eine recht praktikable Arbeitsweise herausgestellt.

1. Ich ordne den Kurs so ein, dass ich ihn in Tuxracer schnell erreichen kann. Das heißt, es ist der einzige Kurs in contrib (schnell zu erreichen über „An einer Veranstaltung teilnehmen“), oder ich arbeite im Beginner-Bereich der Originalkurse (schnell zu erreichen über „Training“).
2. Ich öffne Tuxeditor und gleichzeitig Gimp, um elev.png zu bearbeiten. Beide Programme bleiben während einer Session geöffnet. Nachdem ich mit Gimp gearbeitet habe, sichere ich die Datei, gehe zurück zum Tuxeditor, klicke auf „Refresh“ und starte Tuxracer von Tuxeditor aus, um das Ergebnis zu testen.
3. Anschließend, oder auch parallel zur Bearbeitung des Höhenmodells gehe ich auf die Terrain-Seite und bringe einige Texturen auf, meistens ist es zunächst noch Eis, um schneller zu bestimmten Teststellen zu gelangen.
4. An Objekten befasse ich mich in dieser ersten Phase nur mit solchen, die evtl. eine Überarbeitung des Höhenmodells erfordern (z.B. Brücken, Häuser, querliegende Stämme usw.) Jetzt ist auch noch Zeit,

einige grundsätzliche Parameter auf der Course-Seite endgültig einzustellen. Kurs zu langsam? Der Neigungswinkel muss jetzt erhöht werden; später gibt es Schwierigkeiten. In dieser Beziehung muss man umdenken, wenn man an die Kurserstellung für Tuxracer 0.61 gewöhnt ist.
5. Wenn dann (nach einigen Sessions) die Höhenstruktur stimmt, kommen die endgültigen Terrains sowie die nicht so kritischen Objekte (Heringe, Bäume, Flaggen) dran. Gimp wird nun nicht mehr gebraucht.

Anmerkung: Prinzipiell wäre es auch möglich, Tuxracer geöffnet zu lassen und nach einem Testlauf nur das Rennen zu beenden, aber dieses Verfahren hat seine Tücken. Allzuschnell startet man von Tuxeditor aus versehentlich eine zweite Instanz von Tuxracer, mit recht unangenehmen Folgen. Irgendwie scheint Tuxracer nur einmal laufen zu wollen - oder aber der Arbeitsspeicher wird total zugestopft. Ich hab's noch nicht untersucht.

6. Die Bedienung von Tuxeditor

6.1 Die Dateifunktionen

Um einen Kurs zu öffnen, musst du die entsprechende Datei course.tcl öffnen.
Wenn es sich dabei um einen Originalkurs handelt, wird - soforn das nicht bereits geschehen ist - im Ordner ../course/_backup eine Sicherheitskopie angelegt. Mit der Option „restore original course“ kannst du jederzeit wieder den Originalzustand herstellen. Wichtig: Nach der Wiederherstellung des Originalzustandes wird der Kurs in Tuxeditor geschlossen, und er darf auch nicht wieder geöffnet werden, denn dadurch würde Tuxeditor erneut Besitz von dem Kurs ergreifen und z.B. die animierten Objekte unterdrücken.

Beim Speichern bzw. Schließen eines Kurses werden einige Dateien von Tuxeditor neu generiert, z.B. items.tcl. Die vorhandenen Objekte bleiben weitgehend erhalten - lediglich die animierten Objekte (z.B. fahrende Autos) und einige spezielle Sounds (z.B. Geräusch der Mühle in Forest 1) gehen in dieser Programmversion noch verloren.

Neue Kurse werden immer im Ordner ../courses/contrib erstellt. Wenn sich an der Stelle bereits ein Kurs befindet, wird er überschrieben. Tuxeditor bietet einige Standardgrößen von Bitmaps an und richtet die Kursparameter in course.tcl daran aus. Beachte bitte, dass die Performance von Tuxracer ganz erheblich von der Bitmapgröße abhängt (stärker noch als von der Anzahl der Objekte). Entscheide dich nur dann für die großen Bitmaps, wenn der Kurs sehr detailliert ausgearbeitet werden soll. Einen guten Kompromiss stellen Bitmaps mit 100 x 1000 Pixeln dar. Für die Umgebungsvariablen (siehe Seite „Environment“) werden zunächst Standardwerte aus dem Originalkurs „Arctic“ herangezogen. Du kannst die Einstellungen natürlich jederzeit ändern.

6.2 Die Seite „Course“

Nach Möglichkeit solltest du die Kursparameter einstellen, bevor du die Objekte anbringst. Andernfalls könnte bei einigen Objekten eine Nachbearbeitung erforderlich sein. Das gilt vor allem für die Objekte, die in Höhe und Neigung dem Gelände angepasst werden müssen.

6.3 Die Seite „Elevation“

Die Bearbeitung des Höhenmodells ist mit Tuxeditor noch nicht möglich, das muss mit einem externen Grafikprogramm geschehen. Falls du mit Gimp arbeitest, ist es am einfachsten, das Programm von Tuxeditor aus aufzurufen - dann wird die Bitmapt elev.png gleich geladen.

Nach dem Speichern der Grafik muss in Tuxeditor mit Hilfe des Buttons „Refresh“ die geänderte Bitmap neu geladen werden.

6.4 Die Seite „Terrain“

Entgegen meinem ursprünglichen Plan habe ich diese Option bereits in die erste Programmversion eingebaut. Es geht zwar nicht ganz so komfortabel zu wie mit Gimp, aber ich denke, man kann schon gut damit arbeiten.

Der Vorgang ist einfach: Wähle einen geeigneten Pinsel und das gewünschte Terrain und male auf der Bitmap herum. Wenn du „Elevation“ sichtbar gemacht hast, kannst du die Ergebnisse deiner Pinselei nicht direkt sehen, aber du kannst dich an der Höhenstruktur orientieren. Wenn du „Terrain“ sichtbar gemacht hast, kannst du pixelgenau zeichnen, aber es fehlt die Höhenstruktur als Anhaltspunkt. Mit Gimp geht das etwas einfacher, weil du die deckende Höhenstruktur transparent machen kannst.

6.5 Die Seite „Objects“

Links befindet sich die Arbeitsfläche mit der Bitmap im Hintergrund, rechts die Palette mit Werkzeugen und Objekten. Die häufig gebrauchten Standardobjekte liegen auf Buttons, seltener gebrauchte Objekte sind rechts daneben aufgelistet, und zwar auf zwei Tab-sheets. Der „Extended object set“ enthält die Objekte, die für den jeweiligen Cup definiert sind, während der „Common object set“ immer verfügbar ist.

Leider können nicht alle Objekte gleichzeitig zur Verfügung stehen, das hängt mit der etwas unglücklichen, starren Struktur von Tuxracer 1.1 zusammen. Ich bin auf diesen Missstand schon an verschiedenen Stellen eingegegangen.

6.5.1 Objekte anbringen und löschen

Objekte werden einfach auf die Bitmap geklickt und erscheinen dort als farbige Punkte. Im Zeigermodus kannst du mit dem Cursor darüber fahren und dir anzeigen lassen, um was für ein Objekt es sich handelt. Zum Löschen von Objekten verwende einen der drei Löschpinsel. Wenn mehrere Objekte derselben Art (z.B. Heringe) angebracht werden sollen, kann es sinnvoll sein, „Lock“ zu aktivieren - dann springt das Programm nicht jedesmal in den Zeigermodus zurück. Wenn du ein Objekt an einer Stelle anbringst, wo sich sich ein Objekt befindet, wird das alte Objekt überschrieben.

Die Bitmap lässt sich sowohl in Größe als auch in Helligkeit und Kontrast einstellen. Dazu dienen die Buttons in der Toolleiste. Hier musst du einfach ausprobieren, welche Darstellung am günstigsten ist.

6.5.2 Objekte bearbeiten

Wenn du im Zeigermodus ein Objekt anklickst, wird es markiert, und es öffnet sich das Editierpanel. Alle Einstellungen werden mit Hilfe von Schiebereglern oder Buttons vorgenommen. Wichtig: Die Richtung der Schieberegler orientiert sich grundsätzlich an der Bitmap. Wenn ein Objekt z.B. in Fahrtrichtung gedreht werden soll, muss der Regler nach unten (!) gezogen werden.

Move: Hiermit wird das Objekt pixelweise verschoben; die Schrittweite ist einstellbar. Wenn das Objekt dadurch in eine andere Höhenlage gerät, wird die erforderliche Neuberechnung der Höhenposition automatisch vorgenommen.

Scale: Die Funktionen dürften selbsterklärend sein. Wenn „Equal“ aktiviert ist, werden die drei Parameter synchron eingestellt.

Rotation: Hier ist etwas Vorstellungsvermögen erforderlich, die Symbole helfen u.U. weiter. Wenn „Set to course angle“ aktiviert ist, wird das Objekt automatisch so um die X-Achse gedreht, dass es sich der Neigung des Kurses anpasst. Doch aufpassen, dass der Turm nicht umstürzt.

Cast Shadow: Diese Merkmal ist nur bei größeren Objekten, die einen Schatten werfen können, einstellbar.

Position fine tuning: Die Höhenpositionen werden der Bitmap entnommen, aber auf schrägen Geländeteilen kann die Position nicht immer einwandfrei berechnet werden. Deshalb gibt es hier die Möglichkeit der Korrektur. Wenn eine Fahnenstange in der Luft baumelt, kann das Objekt ein wenig versenkt werden, oder wenn ein Hering nur halb aus dem Schnee schaut, wird er etwas angehoben. - Horizontal und vertikal lassen sich die Objekte um +/- 0.5 Pixel verschieben. Eine derartig genau Anpassung an das Gelände ist nur selten erforderlich, aber bei Baumreihen kann eine leichte Verschiebung sinnvoll sein, damit nicht die Wirkung einer schnurgeraden Allee entsteht.

Während in x/y-Richtung nur geringfügige Verschiebungen möglich sind, umfasst die Höhenverstellung einen ziemlich großen Bereich und ist deshalb auf zwei Schieberegler verteilt. Zur genauen Positionsanpassung (z.B. um einen Hering aus dem Schnee zu ziehen) werden nur Bruchteile einer Höheneinheit benötigt. Es gibt aber Fälle, wo kräftig angehoben oder gesenkt werden muss, z.B. wenn ein Auto oder ein Baum auf einer Brücke platziert werden soll.

6.6 Die Seite „Environment“

Bitte beachten: Die Umgebungsvariablen beziehen sich grundsätzlich auf einen Pokal, niemals auf einen einzelnen Kurs. Beachte deshalb die Rückwirkungen auf andere Kurse innerhalb des Pokals.

Alle Variablen lassen sich für die drei Lichtsituationen „Sonne“, „Sonnenuntergang“ und „Nebel“ getrennt einstellen. Deshalb muss zunächst eine dieser Sitationen durck Klick auf den entsprechenden Button aktiviert werden.

Es gibt im wesentlichen zwei Gruppen von Umgebungsvariblen, die ebenfalls unabhängig voneinander

einstellbar sind:

a) die Lichtverhältnisse (Lichtfarben, Lichtintensität, Lichtrichtung),
b) die umhüllenden „Kulissenbitmaps“

Um eine Lichtfarbe zu ändern, klicke auf den Farbbutton und wähle die Farbe im Farbdialog. Über die Auswirkung der Lichtrichtung bin ich mir noch nicht ganz im klaren. Experimentiere einfach. Alle Änderungen werden sofort in die entsprechenden Dateien sunny_light.tcl, sunset_light.tcl bzw. foggy_light.tcl eingetragen.

Ein kleiner Versuch, der die Wirkung der Umgebungseinstellungen deutlich macht: Starte einmal den ersten Forest-Kurs. Dann gehe hin und ändere zwei Sachen:
a) Wähle als Umgebungsbilder die aus dem Beginner-Cup
b) Stelle die Nebelfarbe der Situation „Sonne“ sehr hell (fast weiß) ein.
Nun starte den Kurs noch einmal.

Du kannst ja mal versuchen, ein Nachtlicht wie in Tuxracer 0.61 zu erzeugen.

6.7 Die Seite „Race vs opponents“

Hier muss ich noch vertrösten. Ich bin ziemlich optimistisch, dass es demnächst möglich sein wird, die entsprechenden Dateien (ai_targets) elegant und schnell mit Tuxeditor zu editieren, aber vorerst sind andere Dinge wichtiger. Nur soviel an dieser Stelle: Der Gegner gleitet oder springt von Punkt zu Punkt und hält dabei ein genau vorgegebenes Zeitschema ein. Diese Punkte werden (ähnlich wie die Objekte in items.tcl) in Listen erfasst, nämlich den genannten ai_targets. Jeder Punkt stellt eine Weg-Zeit-Markierung dar und hat sehr viel Ähnlichkeit mit den normalen Objekten.

Eigene Kurse müssen zunächst noch ohne diese Option auskommen, was aber nicht ins Gewicht fällt, da für sie der Cup „Race vs opponents“ ohnehin nicht definiert ist. Anders bei geänderten Originalkursen: Solange die Kursparameter und die Bitmap elev.png noch im Originalzustand sind, dürfte das Wettrennen weiterhin möglich sein. Wenn der Pistenverlauf jedoch verlegt wurde, gibt es Probleme, denn dann hopst der Konkurrent irgendwo abseits der Piste herum und wird kaum noch gesehen.

6.8 Die Seite „Internal“

Dies ist eigentlich meine private Seite - solange ich am Programm arbeite. Hier lasse ich mir Variablen anzeigen, überprüfe, ob Dateien richtig gerendert und generiert werden usw. Damit die Seite auch für dich einen Zweck erfüllt, kannst du dir eine Art Logfile anzeigen lassen. Fehler- und Erfolgsmeldungen werden nicht in einer Datei protokolliert, sondern zunächst nur in einer Liste, die du hier einsehen kannst.

6.9 Den Kurs testen

Ein Klick auf den Button mit dem Pinguin startet Tuxracer, wobei vorher die Datei items.tcl neu generiert wird. Beachte beim Testen der Originalkurse, dass in Tuxracer die Kurse Swiss 2 und Swiss 3 vertauscht sind. Um also Swiss 2 zu testen, musst du den allerletzten Kurs auswählen.

7. Hinweise zu den Objekten

7.1 Start und Ziel

Die Startpunkte (es gibt davon 4, entsprechend den Startpositionen beim Rennen mit mehreren Figuren) sind äußerst schwierig einzustellen. Höhe und Abstand müssen genau zur Startrampe passen. Um die Sache zu vereinfachen, sind in Tuxeditor die (abstrakten) Startpunkte und die Startrampe zu einem Objekt zusammengefasst. Du brauchst nur die Startrampe aufzuklicken, die anderen Positionen werden automatisch berechnet und in items.tcl eingefügt. Alternativ zur Startrampe gibt es noch den „Startpoint blank“, bei dem die Rampe wegfällt. Natürlich darf nur einer der beiden angebracht werden.

Im Gegensatz dazu sind Starttor (startgate) und Zieltor (finishgate) reine 3D-Objekte, ohne Einfluss auf den Rennverlauf. Sie können in beliebiger Anzahl an beliebigen Positionen angebracht werden, obwohl es sich empfiehlt, das Zieltor genau auf der Ziellinie zu platzieren.

7.2 Das Neigungsproblem

Als typisches Beispiel nehme ich ein Haus. Wenn du ein Haus auf die Piste klickst, wirst du beim Testlauf feststellen, dass es keineswegs so steht wie du dir das vorgestellt hast. Die Vorderkante versinkt im Schnee, die Hinterkante ragt in die Luft. Die Ursache liegt im Neigungswinkel des Kurses, der erhebliche Auswirkungen hat, auch wenn es normalerweise nicht auffällt. Das Haus wird natürlich zunächst exakt waagerecht in die Landschaft gesetzt. Um es mit der ganzen Grundfläche auf den Schnee zu setzen, gibt es vier Möglichkeiten:

1. Du kannst das Unterbau mit einer Steigung versehen, die den Neigungswinkel ausgleicht. Außerhalb der Piste geht das problemlos, aber wenn das Objekt direkt auf der Piste steht, muss Tux klettern, um an dem Objekt vorbeizukommen.
2. Du kannst das Objekt im selben Winkel wie den Kurs neigen. Von vorne fällt es kaum auf, aber wenn man vorbeifährt, wirkt es unnatürlich. Der „Schräge Turm von Pisa“ steht gerader. Am ehesten scheint diese Methode bei Objekten mit geringer Tiefe, die außerdem nicht von der Seite betrachtet werden können, sinnvoll zu sein (z.B. Brücken).
3. Du kannst das Objekt in den Schnee absenken. Auch hierbei ist Vorsicht angebracht - es macht sich nicht gut, wenn auf der Vorderseite die Fenster halb aus dem Schnee hervorlschauen.
4. Du kannst die Lücken, die unterhalb eines Objektes klaffen, mit anderen Objekten vertuschen. Du musst nur darauf achten, dass man die Sache nicht allzu genau untersuchen kann.

Facit: Alle Verfahren müssen zurückhaltend angewandt werden; am besten ist es, die Methoden miteinander zu kombinieren.

Im übrigen wird auch bei den Originalkursen ganz schön gepfuscht. Da gibt es Häuser, die weder geneigt noch abgesenkt sind, und auch der Untergrund weist nicht auf besondere Bearbeitung hin. Von der Piste aus gesehen fällt einem nichts auf. Teilweise ist die Sicht versperrt, etwa durch hohe Böschungen. Der Trick: Man kommt nicht an die Häuser ran, wegen der Boundaries. Radiert man die Boundaries aber weg und quält sich hinauf, dann sieht man die Bescherung ... Alles Illusion. Aber ist nicht der gesamte Kurs Illusion? Baumspitzen, die knapp über einen Hügel ragen, gaukeln eine verborgene Landschaft vor, die in Wirklichkeit nur aus einer Einöde besteht. Die Desillusionierung wird durch Unzugänglichkeit verhindert. Ich glaube, es ist wesentlich für den Reiz eines Kurses, wenn man solche Illusionen erzeugen kann.

7.3 Wie erzeut man eine waagerechte Fläche?

Die Steigung einer Fläche so einzustellen, dass sie im Kurs waagerecht erscheint und als Unterbau für ein Haus dienen kann, ist nicht ganz einfach, da sehr viele Faktoren mitspielen. Ohne Experimentieren geht es nicht. Aber es gibt einige Anhaltspunkte. Angenommen, das Haus benötigt eine Grundfläche von 15 Pixeln Tiefe (es geht hier nur um die Ausdehnung in y-Richtung). Nehmen wir ferner an, dass 1 Pixel 2 Längeneinheiten entspricht, was bei einer üblichen Kursskalierung mit dem Faktor 2 der Fall ist (z.B. Bitmaplänge 500, Kurslänge 1000). Dann haben wir 30 Längeneinheiten waagerecht zu legen. Bei einer Kursneigung von 25° sinkt der auszugleichende Höhenlevel um rund 14 Höheneinheiten (tan 25° x 30). Nun kommt noch die Höhenskalierung des Kurses (elev_scale) ins Spiel. Wenn wir hier 20 eingestellt haben, bedeutet es, dass wir auf den 15 Pixeln eine Steigung um 14 * 255 / 20 = 178 Graustufen unterbringen müssen. Das ist immerhin 2/3 der gesamten Graustufenskale, eine dramatische Steigung, die sich außerdem nur dann realisieren lässt, wenn der gesamte Höhenlevel an dieser Stelle ziemlich niedrig liegt. Aber die Rechnung stimmt. Ich habe sie auf einem Testkurs präzise umgesetzt und konnte feststellen, dass das Haus exakt aufliegt.

Hier zeigt sich erneut, welch starken Einfluss die Neigung des Kurses hat. Ein möglicher Kompromiss sähe so aus: Wir neigen das Haus um 5°, was nicht besonders auffällt. Weitere 5° Fehlanpassung nehmen wir in Kauf, weil wir das Haus etwas versenken wollen. Dann wären nur noch 15° auszugleichen. Außerdem machen wir die Auflagefläche nicht größer als unbedingt nötig, können demnach auf 12 Pixel zurückgehen. Rechnen wir noch einmal durch: Der auszugleichende Höhenunterschied beträgt nun bei 24 Längenheiten nur noch 6.5, was einer Graustufendifferenz von 82 entspricht. Das sieht schon wesentlich moderater aus.

Ein schönes Beispiel für diese Problemstellung findest du im Originalkurs „Swiss2“. Lade den Kurs in Tuxeditor und schau dir auf der Objektseite die Höhenstruktur an. Gleich am Anfang des Kurses erkennst du die Plattformen beidseitig der Straße, auf denen die Häuser plaziert sind. Wenn du diese Steigungen mit dem Cursor abtastest, stellst du Höhenunterschiede von etwa 100 Graustufenwerten fest. Das ist ein guter Anhaltspunkt für erste Versuche.

7.4 Der Positionspunkt

Flächenobjekte erscheinen beim Aufklicken zunächst als Punkt. Auch wenn im Editiermodus annähernd der Grundriss angezeigt wird, wird zusätzlich immer dieser Punkt markiert. Er hat nämlich eine doppelte Bedeutung:

a) Der Positionspunkt ist zugleich der Drehpunkt. Meistens liegt der Drehpunkt im Mittelpunkt der Grundfläche, doch bei den Pads hat man ihn auf die Vorderkante verlegt. Das ist auch sinnvoll, denn damit ist gewährleistet, dass das Pad ohne Stufe befahren werden kann.

b) Die Höhe des Objekts wird am Positionspunkt abgegriffen. Das kann dazu führen, dass einige Stellen des Objektes zu hoch oder zu tief erscheinen. Hier hilft nur der Ausgleich mittels „Fine tuning".

Facit: Manche ärgerliche Erscheinung findet eine logische Erklärung, wenn man den Positionspunkt genauer untersucht.

7.5 Besonderheiten des Speedpads

Der Geschwindigkeitimpuls kommt durch das Überfahren des Pads zustande, und je länger das Pad ist, desto mehr Fahrt bekommt der Pinguin. Ein aufschlussreiches Experiment: Spreize das Pad in Fahrtrichtung auf die dreifache Länge - Tux schießt förmlich nach vorne. Wenn das Pad entsprechend der Kursneigung ausgerichtet ist, wird er an der nächsten Felswand zerschellen. Wenn das Pad aber waagerecht liegt (scheinbar nach oben zeigt), geht es ab in die Luft, und Tux sieht die Welt unter sich erst dann wieder, wenn die Ziellinie überschritten wurde. Erinnerungen an den 0.61-Kurs „The Big Ass Jump" werden lebendig. Du kannst also durch geschickte Skalierung die Wirkung des Pads steuern.

7.6 Bäume

Bäume gehören zu den unproblematischen Objekten. Du musst nur eines bedenken: Weil die Bäume unsymmetrisch geformt sind, fällt es auf, wenn sie alle in die selbe Richtung schauen. Deshalb erhalten sie per Default einen zufälligen Rotationswinkel.

Außerdem werden die Standardbäume von Tuxeditor in drei verschiedenen Größen angeboten, um nicht jeden Baum einzeln skalieren zu müssen. Der „kleine Baum" hat die Größe 0.4, der mittere die Größe 0.8 und der groß die Größe 1.6. Selbstverständlich kannst du die Bäume jederzeit auf die von dir gewünschte Größe umstellen.

7.7 Boundaries

Boundaries sind unsichtbare Linienobjekte, die dafür sorgen sollen, dass Tux nicht aus der Bahn gerät oder verbotene Gefilde betritt. Die Originalkurse machen reichlich Gebrauch davon - zu reichlich, wie ich finde. Warum soll man nicht (auf Kosten der Geschwindigkeit) ein wenig das Gelände erkunden dürfen? Abgesehen davon gibt es noch andere, schönere Möglichkeiten, einen Weg zu versperren: steile Böschungen, unüberwindbare Felsen usw.

Wie dem auch sei: Boundaries werden durch 2 Parameter gekennzeichnet: Tiefe (= y-Skalierung) und Rotationswinkel (z-Rotation). Die anderen Parameter scheinen keine Rolle zu spielen, zumindest habe ich keine Auswirkungen feststellen können.

7.8 Tunnels

Die Tunnelobjekte (straight, left, right) sind sehr eigenwillig und nicht ganz einfach anzubringen. Eigentlich bestehen die Tunnels nur aus dünnen Eishülsen und müssen, damit sie von außen nicht als solche zu erkennen sind, mit den darauf zugeschnittenen „Snowcovers" bedeckt werden. Ich hatte zuerst geplant, die Snowcovers automatisch zu setzen, aber dann habe ich davon Abstand genommen, da die Covers oft unabhängig von den Tunnels der Gegend angepasst werden.

Eine besonders unangenehme Merkwürdigkeit: Tunnels sind nur von innen „collidable", von außen kann man die „Wand" einfach durchbrechen. Dasselbe gilt für die Covers. Tunnelobjekte sind demnach völlig ungeeignet für „Crossover"-Zwecke. Man muss sogar dafür sorgen, dass Tunnels und Snowcovers von außen nicht erreichbar sind. Da es sich bei beiden nur um dünne Schalen handelt, muss der Eingang immer (!) mit einer Fassade versehen werden, die die überaus hässlichen, scharfkantigen Ränder verdeckt. Dazu dienen die beiden „Tunnel-Opening"-Objekte.

Grundsätzlich:

- Tunnels und Snowcovers werden in unmittelbarer Nähe zueinander untergebracht. Dasselbe gilt für das Tunnel-Opening-Objekt. Es empfiehlt sich, das Snowcover 1 Pixel unterhalb des Tunnels zu setzen, das Opening noch 2 oder 3 Pixel tiefer (wegen dessen Tiefe).
- Der Neigungswinkel von Tunnel und Snowcover müssen übereinstimmen, am besten pass man beide dem Neigungswinkel des Kurses an.
- Länge von Tunnel und Snowcover müssen übereinstimmen; dagegen wird es oft sinnvoll sein, die Breite und Höhe des Covers größer als den Tunnel einzustellen.
- Von außen darf es keine Zugangsmöglichkeit zum Tunnel geben - der Effekt, sich auf einmal im Innern des Tunnels wiederzufinden, ist nicht besonders schön, allenfalls verblüffend - im negativen Sinne.

Facit: Kaum ein Objekt benötigt soviel Fummelei wie der Tunnel. Man wird den Eindruck nicht los, dass die Kursgestalter von Mountain und Glacier versucht haben, irgendwie mit eher schlechten Mitteln so etwas wie einen Tunneleffekt zu erzeugen. Als „Vollobjekte“ wie z.B. die „Arches“ wären die Tunnels wesentlich besser zu handhaben und können auch als Crossovers dienen. Durch Spreizung in y-Richtung kann man mit einem „Arch“ auch so etwas wie einen Tunneleffekt erzeugen (sogar gebogen, wenn man mehrere Arches aneinanderreiht), aber dummerweise sind Arches auf der Oberseite nicht eben wie z.B. die Brücken.

7.9 Brücken - wenn sich Wege kreuzen

Wenn eine Brücke einigermaßen quer ausgerichtet ist (also z-Rotation annähernd 0), bereitet sie wenig Probleme. Aufgrund ihrer Bauweise kann sie gut in die Böschungen versenkt werden. Ein wenig höher oder niedriger einstellen, vielleicht links oder rechts etwas anheben oder absenken, vielleicht noch ein wenig spreizen - schon passt es. Aber eine solche Brücke ist nur Zierde, zum Überfahren ist sie schlecht geeignet, da die Überfahrt quer zur y-Richtung verlaufen und infolgedessen wegen mangelnden Gefälles stark abbremsen würde.

Bei gut gestalteten „Crossovers“ zeigen beide Wege nach unten und schneiden sich in einem möglichst spitzen Winkel. Das bedeutet, dass die Brücke mindestens 45° um die z-Achse gedreht werden muss, wodurch nun einige Aspekte bedeutsam werden, an die man möglicherweise nicht sofort denkt. Die Anschlussplattformen beidseitig der Brücke liegen nun auf verschiedenen Höhenleveln, da sie in y-Richtung gegeneinander verschoben sind. Wenn nicht die untere Plattform angehoben wird, muss die Brücke dasselbe Gefälle haben wie die Bahn, was aber automatisch durch die y-Rotation erreicht wird, sofern sie auf die Bahnneigung eingestellt wird. Die Brücke erhält durch die Diagonalstellung eine doppelte Neigung, einmal in Fahrbahnrichtung, was wünschenswert ist, zum andern quer zur Fahrbahn, was nicht wünschenswert, aber unvermeidlich ist. Erst wenn man einen solchen Brückensanschluss herstellt, fällt einem so richtig auf, dass auch schäg verlaufende Wege quer zur Fahrbahn geneigt sind. Man kann zwar diese Querneigung durch eine x-Rotation ausgleichen, aber dann passt es wieder nicht mit den Anschlüssen.

Einige Tips:

- Sorge zunächst für vernünftige Anschlussplattformen von gleicher Höhe (gleichem Grauwert). Die Böschungen zwischen den Plattformen sollten ruhig etwas steiler sein. Es sollen wirklich Plattformen sein, dargestellt durch gleichmäßige Grauflächen auf der Bitmap.
- Stelle die y-Rotation der Brücke exakt auf die Neigung des Kurses ein.
- Wenn die Brücke auf einer Seite einen guten Anschluss an die Plattform hat, auf der anderen Seite aber z.B. zu tief liegt, versuche nicht, durch x-Rotation das Übel zu beheben; das zieht eine Kette von weiteren Maßnahmen nach sich. Verschiebe vielmehr die Brücke so, dass die „Höhenfehler“ auf beiden Seiten etwa gleich groß sind. Dann hebe die ganze Brücke an und / oder spreize sie.
- Wichtig ist, dass man sich die Verhältnisse gut vorstellen kann, was nicht ganz einfach ist. Ein ganz banaler, aber wirksamer Tip: Nimm einen Bauklotz in die Hand und und rotiere ihn in den verschiedenen Richtungen. Wichtig ist, dass die Rotation nicht auf den Körper bezogen sind, sondern auf den Kurs, im Gegensatz zu den Skalierungen, die objektbezogen sind (ein um 90° gedrehtes Objekt wird durch die x-Skalierung nun in y-Richtung gespreizt oder gestaucht).

7.10 Objekte und Landschaft

Auch wenn sich ein Objekt sehr gut in die Landschaft einfügt - es ist doch immer etwas anderes als ein „natürlicher“ Bestandteil der Landschaft. Beispiel: Ich kann einen schneebedeckten Stein erzeugen, indem ich eine scharfe Erhebung in das Höhenmodell einbaue. Ich kann aber auch einen Stein (boulder) mit einer Schneetextur bedecken und als Objekt einfügen. Beide zeigen einen gravierenden Unterschied: Objekte

werden anders beleuchtet als Landschaftsteile. Schnee als Objekttextur liegt meistens so stark im Schatten, dass die Schneestruktur kaum erkennbar ist.

Was auch nicht immer sofort beachtet wird: Es gibt nur einen Höhenlevel. Wenn ich ein „collidable“ Objekt, etwa eine Brücke, überfahre, befinde ich mich „in der Luft“. So kann ich auch nicht einfach ein Auto auf die Brücke setzen - es wird per Default auf den Grund gestellt, also unter die Brücke oder gar in das Innnere des Brückenpfeilers. Um das Auto auf die Brücke zu bekommen, muss ich es anheben, was in Ermangelung absoluter Skalierungsmaße sehr viel Herumprobieren zur Folge hat.

Facit: Man kann eine Menge mit Objekten erreichen, aber man sollte immer daran denken, dass Objekte keine Landschaft ersetzen können. Sie bleiben im wahrsten Sinne des Wortes „aufgesetzt“.

7.11 Zur Anzeige von Flächen- und Linienobjekten

Wenn du ein Flächen- oder Linienobjekt markierst, wird (meistens) der Grundriß angezeigt. Allerdings ist diese Anzeige nicht sehr genau, weil mir keine exakten Daten über die Ausmaße der Objekte vorliegen. Wahrscheinlich kann man sie aus den Dateien xxx.obj.strip auslesen, aber damit kenne ich mich noch nicht aus. Als Anhaltspunkt kann die Anzeige jedoch hilfreich sein.

Noch ein Hinweis: Ein Punktobjekt wie z.B. ein Baum wird zu einem Linienobjekt, wenn man es um die x- oder y-Achse dreht. Bei einer Drehung von 90° liegt das Objekt flach, und die Länge auf der Grundfläche entspricht der Höhe des Objektes. Diese Besonderheiten werden nicht berücksichtigt. Wenn du wissen möchtest, ob ein gefällter Baum den gesamten Hohlweg überspannt, musst du das experimentell herausfinden.

Ähnliches gilt für Objekte, die in Normalposition liegen, also Linienobjekte sind, z.B. der Baumstamm („log“) aus dem Forest-Set. Die Verkürzung, bedingt durch das Aufrichten, wird nicht angezeigt. Der Aufwand wäre meines Erachtens nicht angemessen.

Es kann sein, dass Flächenobjekte beim Drehen stark verzerrt erscheinen. Das ist in Ordnung so. Dies tritt immer dann auf, wenn das Verhältnis der Bitmaplänge zur Bitmapbreite nicht mit dem Verhältnis der tatsächlichen Kurslänge zur Kursbreite übereinstimmt. Dann geben die Bitmaps die Größenverhältnisse verzerrt wieder.

7.12 Auswirkungen der Kursparameter auf die Objekte

Auf der Seite „Course“ gibt es einen Edit-Button, mit dem die Kursparamter eingestellt werden können. Die Frage ist, welche Auswirkungen diese Änderung auf die Objekte hat. Es gibt gewisse Auswirkungen, und in einigen Fällen müssen Objekte nachbearbeitet werden, aber im großen und Ganzen halten sich die Folgen in Grenzen. Trotzdem gilt der Grundsatz: Nach Möglichkeit sollte die Festlegung der Kursparamter der erste Schritt sein.

Was kann nun bei einer nachträglichen Änderung im einzelnen geschehen? Einige Beispiele zeigen Ursachen und Wirkungen.

Beispiel 1: Elev_scale wird von 24 auf 20 reduziert, um die Landschaft sanfter zu machen. Danach ragt die Brücke, die vorher den Hohlweg überspannte, etwas in die Luft. Es ist klar, dass die Höhe der Brücke nun neu angepasst werden muss.

Beispiel 2: Der Neigungswinkel wird, um den Kurs schneller zu machen, von 18° auf 25° erhöht. Nun ragt das Haus mit der Hinterkante in die Luft. Wenn man die Steigung des Untergrunds nicht erhöhen will, muss das Haus etwas stärker geneigt oder in den Schnee versenkt werden.

Beispiel 3: Die Länge (und gleichzeitig play_length) wird erhöht, um längeren Spielspaß am Rennen zu erreichen und einige zu scharf geratene Kurven zu entschärfen. Nun erreicht Tux das Zieltor nicht mehr, sondern schwingt sich schon vorher in den Schnee. Ursache: das Verhältnis von length und play_length wurde bei der Änderung nicht exakt eingehalten, wodurch sich die Ziellinie verschoben hat. Die einfache Lösung, nämlich das Zieltor auf die Ziellinie zu verschieben, ist in diesem Fall nicht die beste. Besser ist es, play_length so einzustellen, dass die Ziellinie wieder an der richtigen Stelle erscheint.

Die Beispiele dürften zeigen, dass die Auswirkungen immer logischer Natur sind. Je mehr Arbeit man mit dem Skalieren und Zurechtrücken von Objekten hatte, desto zurückhaltender sollte man mit der nachträglichen Änderung der Kursparameter sein.

8. Einige Interna

8.1 Zur Objektbehandlung

1. Damit das Programm weiß, wie es mit den Objekten umzugehen hat, existiert eine zentrale Objektliste (objects.lst) im Binärverzeichnis von Tuxeditor. Diese Liste enthält alle erforderlichen Informationen über die verfügbaren Objekte. Die Liste kann (mit Vorsicht) bearbeitet werden.
2. Tuxracer wertet bei einem Rennen die Datei items.tcl aus. Das Format ist aber völlig ungeeignet für die Bearbeitung von Objekten. Deshalb werden alle auf dem Kurs plazierten Objekte in einem eigenen Format verwaltet. Dieses Objektarray wird im Kursordner als objects.arr gespeichert. Erst wenn ein Testlauf gestartet wird, generiert Tuxeditor aus dem Objektarray die von Tuxracer lesbare items.tcl.
3. Parallel zum Objektarray verwendet Tuxeditor einen Positionsindex, in welchem die mit einem Objekt belegten Bildpunkte registriert werden. Dadurch kann schnell festgestellt werden, ob sich an einer Stelle ein Objekt befindet, und um welches es sich handelt. Der Index wird nicht gespeichert, sondern bei Bedarf intern aufgebaut bzw. aktualisiert.

8.2 Was geschieht beim Öffnen eines Kurses?

Grundsätzlich können nur gültige Kurse geöffnet werden. Damit Tuxeditor einen Kurs als „gültig“ erkennt, müssen folgende Dateien an richtiger Stelle vorhanden sein:

- Die Bitmaps elev.png und terrain.png im Kursordner
- Die Beschreibungsdatei course.tcl im Kursordner
- Die Definitionsdateien objects.tcl und object_terrains.tcl im übergeordneten Pokalordner
- Die Umgebungsdateien sunny_light.tcl, sunset_light.tcl und foggy_light.tcl im Pokalordner

Wenn eine dieser Dateien fehlt, bricht der Vorgang ab.

1. Mit Hilfe des Open-Dialogs wird der Pfad zu course.tcl ermittelt. Anhand dieses Pfades kann die Gültigkeitsüberprüfung stattfinden. Wenn alles ok ist, beginnt der eigentliche Vorgang:
2. Zuerst werden die Bitmaps geladen und die Bitmapparameter notiert.
3. Dann wird die Datei course.tcl ausgewertet. Die Parameter werden wiederum notiert. Zusammen mit den Bitmapparametern bilden Sie die Grundlage für alle Berechnungen. Bei jedem Speichern des Kurses wird course.tcl neu generiert.
4. Als nächstes stellt Tuxracer anhand von objects.tcl fest, welche Objektsammlung der Kurs benutzt. Die zu dieser Sammlung gehörenden Objekte werden aus der zentralen Liste (objects.lst) herausgefiltert und in der erweiterten Objektliste angezeigt.
5. Schließlich werden die auf dem Kurs angebrachten Objekte selbst geladen. Wenn bereits ein Objektarray existiert, wird dieses geladen. Falls objects.arr nicht vorhanden ist, sucht Tuxeditor die items.tcl, rendert sie und übertragt die Daten in das Objektarray. Falls auch die items.tcl fehlt, wird ein neues, leeres Objektarray angelegt.

8.3 Wie geht Tuxeditor mit den Bitmaps um?

Beide Bitmaps (elev.png und terrain.png) werden geladen und stehen in entsprechenden Komponenten für verschiedene Zwecke unverändert zur Verfügung.

Um die Bitmaps als Hintergrund für die Objektberarbeitung anzuzeigen, werden sie in einem zweistufigen Prozess aufbereitet. Zunächst werden sie in Helligkeit und Kontrast eingestellt und umgespeichert, dann werden sie auf die gewünschte Größe skaliert und noch einmal umgespeichert, jetzt in die sichtbare Komponente.

Die Objektpunkte, die auf den Bitmaps erscheinen, sind nicht (!) Bestandteil einer weiteren Bitmap, sondern werden in den Canvas der Arbeitsmap gezeichnet, wobei Tuxeditor das Objektarray auswertet.

8.4 Ein paar Anmerkungen zur Genauigkeit der Objektparameter

In der items.tcl können (innerhalb des Kursbereiches) die Parmeter beliebig genau eingestellt werden. Beim Arbeiten auf einer Bitmap ist die Genauigkeit dagegen begrenzt: die x/y-Koordinaten durch die Pixelauflösung, die Höhenkoordinate durch die Graustufenauflösung. Um trotzdem eine exakte Anpassung an die Umgebung zu ermöglichen, werden in Tuxeditor die Korrekturparameter dx, dy und dz angewandt, die als „Position fine tuning“ in Erscheinung treten.

Wenn eine existierende items.tcl gerendert wird, müssen diese Korrekturparameter vom Programm ermittelt werden, damit eine möglichst genaue Anpassung an die Orginaleinstellungen gewährleistet wird. Bei den x/y-Koordinaten macht sich das kaum bemerkbar, da es sich allenfalls um Verschiebungen von +/- 0.5 Pixel handeln kann, aber die Höheneinstellungen sind äußerst sensibel. Hier kann es durchaus passieren, dass ein Objekt teilweise in die Luft ragt, weil beim Originalkurs, um diesen Effekt zu vermeiden, das Objekt mehr oder weniger abgesenkt wurde. Und um die Höhenkorrektur richtig berechnen zu können, müssen schließlich auch die x/y-Parameter korrigiert werden, denn die Höhe hängt auch davon ab.

Ziemlich unempfindlich sind die Rotationswerte um die x- oder z-Achse. Tuxeditor geht hier sogar ziemlich grob in 5°-Stufen vor. Wenn z.B. ein Baum im Originalkurs um 132.67° gedreht wird, nach dem Rendern jedoch um 135°, ist das völlig belanglos. Kritisch dagegen ist die Drehung um die x-Achse, weil sich hier die Kursneigung (etwa 20° bis 30°) ganz erheblich bemerkbar macht. Hier kann es unter Umständen auf jedes Grad ankommen.

8.5 Warum keine gemeinsamen Ressourcen in der Oringinalstruktur?

Dass die oben beschriebene, logische und einfach zu realisierende Idealstruktur nicht eingehalten wurde, hat einige Fragen nach den Gründen aufgeworfen, die schließlich zu einer naheliegenden Antwort führten: Die Cups wurden von Sunspire arbeitsteilig entwickelt, ohne dass eine intensive Kooperation stattfand. Darauf deuten nicht zuletzt die typischen Handschriften hin. Die Forest-Kurse sind so ganz anders gestaltet als z.B. die Glacier-Kurse. Besondere Vorlieben der Kursprogrammierer treten zutage. Die Mountain-Kurse: rasant, aber in der Landschaftsgestaltung primitiv, Vorliebe für Animationen. Die Forest-Kurse: landschaftlich anspruchsvoll, übergroße Bäume. Die Glacier-Kurse: relativ langsam, Vorliebe für Steigungen und Kurven, zum Teil komische Objekte. Die Swiss-Kurse: Objekte in Hülle und Fülle, Animationen, zweifellos die aufwendigsten Kurse. Und die ersten beiden Cups? Beginner-Kurse und Arctic-Kurse sind ähnlich: relativ unspektakulär, sanfte Winterlandschaften, als Objekte hauptsächlich Bäume und die komischen „Overhangs“. Eindeutig, diese beiden Cups stammen aus derselben Feder, und bezeichnendeweise sind es die einzigen Kurse, die sich eine Objectressource teilen, untergebracht in arctic.

9.6 Race vs opponents

Dieser Aspekt wurde bisher in Tuxeditor noch nicht beachtet. Sofern man die Originalkurse bearbeitet und die Kursparameter (insbesondere length und play_length) nicht verstellt, dürfte diese Form des Rennens nach wie vor möglich sein.

Andererseits bietet „Race vs opponents“ interessante Möglichkeiten, und vor allem scheint es relativ einfach zu sein, die dazu erforderlichen Dateien (unter ai_targets) mit einem Programm wie Tuxeditor zu erstellen. Die „Zeitmarken“ der einzelnen Positionen könnten durch Interpolation aus der „Zielzeit“ gewonnen werden, so dass man lediglich eine Kette solcher Marken aufklicken müsste. Und wenn man die Interpolation nicht linear gestaltet, sondern leicht logarithmisch oder exponentiell verzerrt, gibt es weitere, interessante Möglichkeiten. Der Gegner könnte es zu Beginn langsam angehen lassen und den Spieler in Sicherheit wiegen, während es im Endspurt dann zur Sache geht. Oder umgekehrt: Der Gegner braust davon, und während des ganzen Rennens bleibt die bange Frage, ob er noch rechtzeitig eingeholt werden kann ...

9. Ausblick auf die endgültige Version

Das Ziel ist relativ hoch gesteckt, aber nach den bisherigen Erfahrungen bin ich zuversichtlich, dass es weitgehend erreichbar sein müsste.

1. Die gesamte Kurs-Cup-Struktur soll frei gestaltbar sein. Der Anwender baut sich seine Rennstruktur selber auf, mit den Kursen, die er möchte, mit den Cups, die er möchte, mit den Anforderungen, die er als angemessen empfindet. Es sollte möglich sein, Cups mit weniger oder mehr als drei Kursen einzurichten, und es sollte dem Anwender frei gestellt sein, wieviel Cups er überhaupt installiert. Natürlich sollten sich dabei auch die Originalkurse (wieder) einbinden lassen. Dass, wie Arthur schreibt, Änderungen an der course_idx.tcl schnell zu einem Chaos führen, kann ich nicht bestätigen. Ich bin überzeugt, dass es möglich ist, diese zentrale Steuerungsdatei per Programm zu generieren.
2. Alle Objekte werden in einem Ordner gebündelt, stehen also für alle Kurse zur Verfügung. Ähnliches gilt für die Umgebungsressourcen.
3. Es sollte möglich sein, weitere Terrains oder Objekte einzubinden. Terrains sind relativ einfach zu installieren: Man benötigt eine entsprechende Bitmap (Aufnahme mit der Digitalkamera auf dem Sonntagsspazierung). Für die Definition der Gleiteigenschaften gibt es genügend Vorlagen, die man übernehmen oder anpassen kann. Trotz der reichhaltigen Auswahl an Terrains vermisse ich einige, z.B. eine farblich halbwegs realistische Felsstruktur oder eine satte Rasenfläche.

Neue Objekte dagegen erfordern wesentlich mehr Aufwand. Die Erzeugungsdateien objectname.obj.strip der Originalobjekte wurden mit Edges erstellt, einem 3D-Modellierungsprogramm, das schon seit Jahren nicht mehr gepflegt wird und bei mir die Funktion beharrlich verweigerte. Trotz intensiver Suche habe ich noch kein anderes 3D-Programm gefunden, dass die Objektbeschreibung im selben Format wie Edges speichert. Wenn mir jemand das Dateiformat detailliert erklären könnte, wäre ich schon einen Schritt weiter. Jedenfalls würde es Tuxracer gut tun, wenn einige Objekte hinzukämen: Arches, die als Crossovers geeignet sind, Laubbäume, Büsche, weitere Felsformationen, natürlich auch Gebäude, Mauern usw.

Es ist einleuchtend, dass man dem Anwender nicht zumuten kann, umfangreiche Basteleien an den Tcl-Dateien vorzunehmen oder gewaltige Kopieraktionen zu starten. Infolgedessen müsste ein Installationstool her, mit dem Kurse und Cups auf einfache Weise installiert und verwaltet werden können. Dieses Tool müsste zu Beginn erst mal die Kursstruktur umorganisieren, z.B. die Objekte in einen gemeinsamen Ordner schieben. Da Tuxracer 1.1 ein kommerzielles Programm ist, wird es wohl nicht erlaubt sein, einen mit Originalobjekten bestücken Ordner mitzulierfern.

Das Installationsprogramm müsste die Kontrolle über die gesamte Struktur übernehmen, insbesondere auch über die zentrale Datei course_idx.tcl.

All das erscheint zunächst etwas utopisch, aber nach meinen Erfahrungen, die ich beim Programmieren von Tuxeditor gesammelt habe, bin ich zuversichtlich, das Ziel weitgehend erreichen zu können. Was mich mehr bewegt, ist die Frage, ob es sich überhaupt lohnt. Tuxracer 1.1 scheint nicht mehr weiterentwickelt zu werden, und die Verbreitung des Programms scheint sich auch in Grenzen zu halten. Ich werde einfach beobachten, wie groß das Interesse an neuen Kursen für Tuxracer 1.1 ist.

Viel Erfolg mit Tuxeditor und viel Spaß mit Tuxracer

Reinhard