BLITZmag

project

vier

Versionsverwaltung mit Git

Sexy Codes
Nützliches, lustiges und fantastisches

Noobody im Interview
Woran er arbeitet und arbeiten wird

Rekursion
Was das ist und was sie nutzt

Inhalt

Vorwort

Wie die, die den Thread verfolgt haben wissen, hatte ich eigentlich nicht die Absicht, die Leitung der neuen PBM-Ausgabe zu übernehmen.

Eigentlich hatte ich den Vorschlag nur gemacht, weil ich die anderen PBMs gerade durchstöbert hatte, und nun auch gerne an einem neuen mitwirken wollte.

Doch dann war die Resonanz des Vorschlags zwar erstaunlich groß, aber es wollte leider niemand der Chefredakteur werden.

Nachdem ich noch nie etwas, was einem Magazin auch nur im Entferntesten nahekam, geleitet hatte, war ich zunächst ratlos.

Dann aber trat Xeres im IRC an mich heran, und erbot sich, das Layout zu übernehmen, gab mir wichtige Tipps, riet mir, erst mal alles abzuspeichern, als ich in Panik geriet und half mir auch sonst in allen möglichen Punkten bei der Organisation, Interviewführung...

Deshalb gilt mein Dank vor allem Xeres.

Als nächstes möchte ich all den Autoren danken, die sich die Zeit genommen haben, zu einem ihrer Fachgebiete einen Artikel zu schreiben, oder auch zu einem Stammtisch gekommen sind und alles Wichtige für uns dokumentiert haben.

Außerdem danke ich meinen "Interview-Opfern", die meine Fragen geduldig beantwortet und auch auf Nachfragen zu bestimmten Themen später noch geantwortet haben.

Auch die fleißigen Testleser, die (hoffentlich) jeden einzelnen Fehler entdeckt haben, verdienen unseren Dank.

Besonderer Dank geht an die Administration, die das PBM-Subforum den Autoren zugänglich gemacht hat.

Und nun viel Spaß mit der neuen, fünften Ausgabe des PBM!

Joshmami

NICKNAMES – WIE ES DAZU KAM

VON JOSH

Ihr fragt euch jetzt sicher: Was soll an Nicks denn so interessant sein?

Es sind eigentlich meist nur wahllos ausgesuchte Namen, die unsere Anonymität schützen sollen. Aber Fakt ist, dass wir ihnen jeden Tag im Internet begegnen, ob in Chatrooms, Foren oder auf anderen Websites.

Und wie ihr gleich sehen werdet, stecken hinter manchen dieser Namen wahre Überraschungen. Aber durch die Nicks verstehen wir auch die Leute, die dahinter stecken, manchmal besser, und können sie uns besser vorstellen. Deshalb habe ich hier einige Leute aus der BB-Community befragt und auch durchaus interessante Antworten erhalten. Lest selbst!

hamZta
Er nannte sich früher „Master“, aber die Leute vertippten sich und machten es zu „amster“.
So beschloss er, sich Hamster zu nennen und aus Designgründen wurde daraus „hamZta“.

Xeres
Er nannte sich so, weil er einen anonymen Namen brauchte und Xeres mit X anfängt, leicht mit Xerxes zu verwechseln ist und an den Zwergplaneten Ceres erinnert.

Pulverfass
Er ist im Reallife sehr an Modellbau interessiert, und verwendet dort unter anderem Backpulver als Baumaterial. Als Nick wählte er dann Pulverfass.

Propellator
Er nannte sich zuerst „The_Nici“, aber nachdem er wegen provozierendem Verhalten aus dem IRC - Chat des Blitzforums gebannt wurde, suchte er sich „Popeller“. Das änderte er jedoch bald in „Propellator“ ab.

Noobody
Bei einem neuen Spiel in WoW wandelte er seinen Nick aus vorherigen Spielen („Nobody“) in „Noobody“ ab, wegen der Ähnlichkeit mit Noob (der er damals noch war) und weil „Nobody“ nicht mehr frei war.

bruZard
Er machte '94 ein Wallpaint in einem Berliner Club und wurde von einem betrunkenen Australier deswegen Brushwizard genannt, aber weil der Australier so nuschelte verstand man nur „Brshzrd“.

Lobby
Das ist schlicht und einfach sein Vorname. Zwar ungewöhnlich, aber es ist so.

Tagirijus
Der Name ist einer von ihm entworfenen Plansprache entsprungen. Er hat gerade dieses Wort genommen, weil es 9 Buchstaben hat und cool klingt.
Er verwendet diesen Nick öfters im Web - auch als Domain für seine *Website*.

Klepto2
Er ist auf seinen Nick beim Spielen eines Pen & Paper RPG's gekommen. Inspiriert wurde er von seiner Figur, einem Dieb, und deswegen nannte er sich Klepto(mane). Das 2 hängte er nur an, weil schon jemand mit dem Namen da war.

Joshmami
Als er mit einem Freund sein erstes Spiel spielte, waren die Namen der Figuren schon vorgegeben, man konnte nur Buchstaben anhängen.
An die coolste Figur „Josh“ hängten sein Freund und er die Anfangsbuchstaben ihrer Vornamen an, und später verwendete er diesen Nick auch alleine.

Count-Doku
Bei seiner Anmeldung im Blitzforum war er unkonzentriert und schrieb statt „Count-Dooku“ eben Count-Doku. Später benutzte er die einfallsreiche Ausrede, er fühle sich fürs Zählen von Dokumentationen verantwortlich.

DarkCorner
DC sagte wörtlich: "Ich schäme mich für meinen Nick!", wollte aber nicht verraten, warum.

Hyde
Er hat als Nick den Namen Hyde aus dem Buch „Dr. Jekyll & Mr. Hyde“ gewählt, weil er sich selbst als das wandelbare personifizierte Gute im Bösen sieht.

Ohaz
… ist der letzte Teil seiner Studenten-ID. Und weil er keine Lust hatte, sich etwas auszudenken, hat er das genommen. Im Forum ist er unter dem Namen TheKilledDeath angemeldet.

BladeRunner
Als Rollenspieler haben ihn die High Tech Settings, Dark Fiction und Cyberpunk am meisten in den Bann gezogen. Als „Blade“ schon vergeben war, war der Weg zum gleichnamigen Film nicht weit.

Abrexxes
In einem alten Schinken aus der griechischen Mythologie war von einem Drachen „Abraxxas“ die Rede.
Der Name wurde für seine Rockband abgelehnt, aber Jahre später wurde daraus „Abrexxes“ für das World Wide Web, inklusive eines Drachenavatars.

BOARD-STYLE

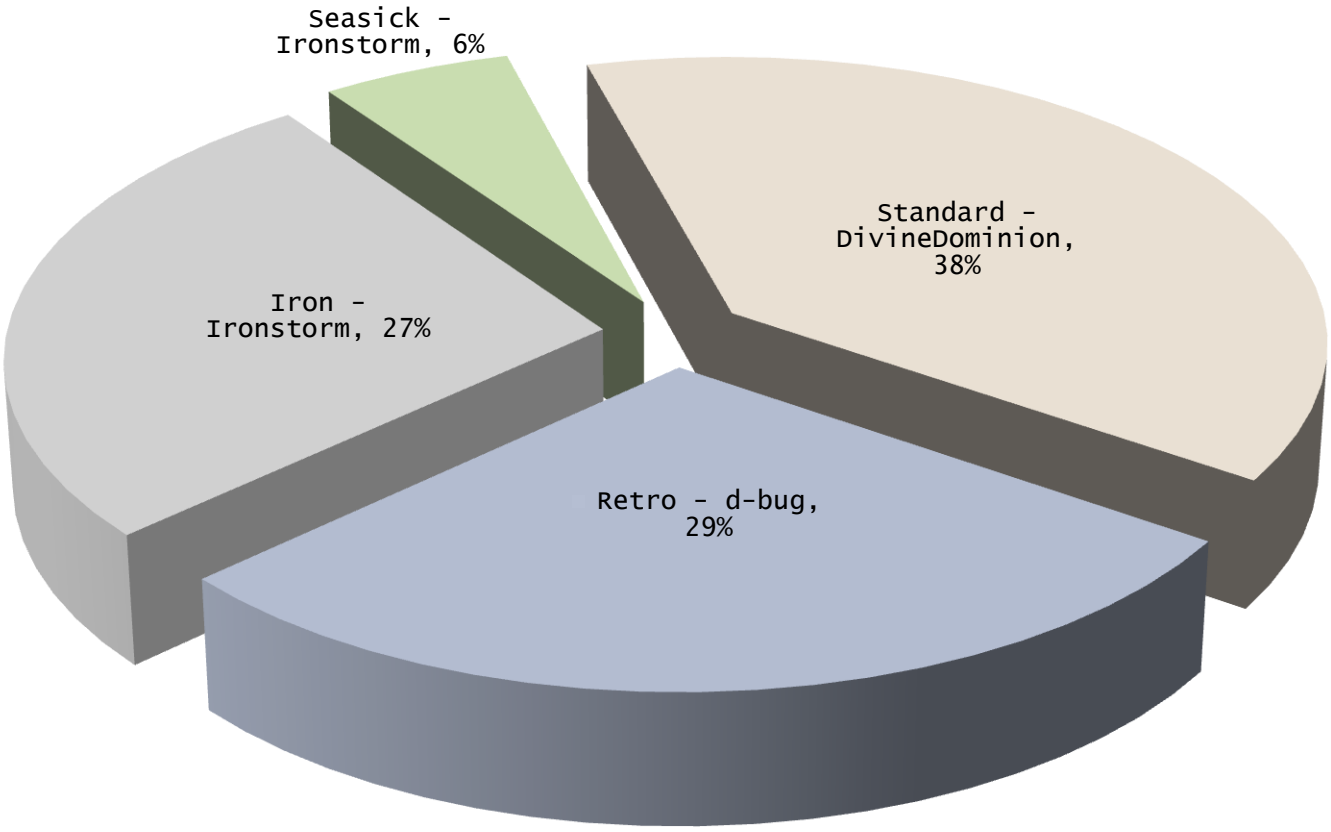

Abbildung 1: Ihr wusstet schon, dass es mehr als ein Style für das Portal gibt? Ihr müsst nicht Seasick benutzen!

INTERVIEW MIT NOOBODY

JOSH FRAGT NOOBODY

Ich habe Noobody vor allem interviewt, weil ich oft und viel von seinen Projekten mit Voxeln & Co hörte, und ich (und vielleicht auch viele Leser) überhaupt keine Vorstellung habe, was ein "Voxel Sparse Octree Raytracer" ist und was man damit anfangen kann.

Um seine Arbeit ein wenig verständlicher zu machen, habe ich einfach mal nachgefragt.

JOSH: Ok, dann frag ich einfach mal drauf los. Wie alt bist du?
NOOBODY: 19.
JOSH: Studierst/Arbeitest du?
NOOBODY: Ich studiere Informatik im 3. Semester.
JOSH: In welchem Land?
NOOBODY: In der Schweiz an der ETH.
JOSH: Wie bist du auf BB/BMax gekommen?
NOOBODY: Hmmm, das war glaube ich ein Freund, der mit mir ein Rennspiel programmieren wollte und mich dann auf die B2D Demo verlinkte. BMax kam dann ein paar Jahre später, weil ich ein wenig mehr Freiheit im Programmieren wollte.
JOSH: Und mit was für Sprachen programmierst du heute so?
NOOBODY: Das kommt ganz auf das Projekt drauf an. Meistens BMax, wenn es schnell sein muss C++, wenn es richtig schnell sein muss, dann CUDA / GLSL / OpenCL und Konsorten. Für die Uni dann je nach dem ASM oder Eiffel.
JOSH: Und woran arbeitest du im Moment?
NOOBODY: Größtenteils ein 2D J'n'R mit ein paar verrückten Russen.
Zur Abwechslung programmiere ich Demos mit ein paar Studienkollegen, und aus Interesse bastle ich an einem Sparse Voxel Octree Raytracer von NVIDIA Research weiter.
JOSH: Was ist ein Sparse Voxel Octree Raytracer?
NOOBODY: Das ist ein Algorithmus zum Rendern von einer speziellen Art von 3D Modellen.
Voxel sind ganz allgemein quasi Zellen in einem 3D-Array.
In der Computergrafik sind Voxel aber meistens eine Art 3D-Pixel. Statt dass man wie bei einem Bild ein Gitter von quadratischen Pixeln hat, hat man mit Voxeln ein 3D-Gitter mit Würfeln.
JOSH: Okay, und SVO ist dann eine Art System um diese Voxeldaten "aufzuschreiben", oder?
NOOBODY: Exakt.
Das Problem mit Voxeln ist, dass sie zum einen sehr speicheraufwändig sind. Um den "Würfellook" loszuwerden, braucht man normalerweise Gitter mit sehr hoher Auflösung, die dann sehr viel Speicher benötigen. Außerdem ist es nicht leicht, diese Gitter zu zeichnen. Darum hat man SVOs entwickelt, die sowohl den Speicherverbrauch senken, als auch das Rendern einfacher machen.

Abbildung 2: Das Stanford Hässchen in 512x512x512 Voxeln.

JOSH: Okay, und was macht dann das Raytracing?
NOOBODY: Das Raytracing wird verwendet, um diese Voxelgitter zu zeichnen. Im Prinzip schickt man für jeden Pixel auf dem Bildschirm einen "Strahl" in das Voxelgitter und berechnet dann, wo der Strahl auftrifft. Dann berechnet man Dinge wie z.B. Farbe und Schatten an der Stelle, an welcher der Strahl auftraf.
JOSH: Und ist das ganze jetzt mit den SVO's schon schnell genug, um es auch

in Spielen einzusetzen? Oder wo wird das Ganze normalerweise verwendet?

NOOBODY: Es ist schon relativ schnell, vor allem, wenn man das Ganze noch mit der Grafikkarte beschleunigt. Allerdings können SVOs z.B. noch keine Animation, weswegen sie im Moment noch für Spiele ungeeignet sind.

JOSH: Aber texturieren kann man sie schon, oder?

NOOBODY: Ja, das schon. Im Moment verwendet man Voxel daher eher noch in z.B. Medizin oder Geologie, wo durch gewisse Methoden wie etwa Computertomographie diese Voxeldatensätze entstehen, welche man dann gerne untersuchen möchte.

JOSH: Braucht man dafür eine spezielle Grafikkarte oder sowas? Oder wie sind die Systemanforderungen?

NOOBODY: Die Version, die ich geschrieben habe, läuft nur auf der CPU und daher so ziemlich überall. Es gibt GPU-beschleunigte Varianten, allerdings braucht man dafür eine Grafikkarte, die eine GPGPU-Sprache wie CUDA oder OpenCL unterstützt - also vor allem neuere Modelle.

JOSH: Kann man es auch in ganz normalen Formaten wie z.B. *.3ds abspeichern?
Oder hat es ein eigenes Format?

NOOBODY: Formate wie *.3ds sind nur für Polygonmodelle und daher nicht für Voxel geeignet. Es gibt noch nicht wirklich standardisierte Voxelformate, aber in der Medizin gibt es ein relativ verbreitetes Format namens DICOM.

JOSH: Was sind die Vorteile / Unterschiede eines Voxelmodells gegenüber eines Polygonmodells?

NOOBODY: Heutzutage, wo wir dank größerer Rechenpower mehr und mehr Detail auf den Bildschirm bringen können, sind Voxel im Vorteil, da sie bei mehr Detail weniger Rechenaufwand benötigen als Polygonmodelle. Außerdem sind sie einfacher zu modellieren und zu texturieren.
Die Nachteile sind bisher der hohe Speicheraufwand und die fehlende Animation.

JOSH: Wer treibt die Weiterentwicklung so voran? Grafikkartenhersteller oder die Medizinindustrie oder so?

NOOBODY: Forschung findet zum einen an Universitäten statt als auch bei Grafikkartenherstellern, die neue Wege erkunden wollen. Die nächste Engine von id software soll ebenfalls voxelbasierent sein.

JOSH: Was willst du nach dem Studium machen?

NOOBODY: Wenn es geht, würde ich gerne in die Forschung einsteigen. Entweder direkt an der Uni, oder halt bei einem Forschungsprogramm einer Firma, z.B. NVIDIA Research.

JOSH: Und wann (mit wie viel Jahren) hast du mit dem Programmieren angefangen?

NOOBODY: Angefangen hat das mit 12, als ich ein Buch über Programmieren mit Visual Basic geschenkt bekam.

JOSH: Und machst du für deine Projekte auch Grafiken und Sound selber?

NOOBODY: Oh nein, das nicht. Grafisch bin ich absolut untalentiert und mit Sounds habe ich gar nicht erst angefangen.

JOSH: Deine Website ist auf Englisch: Heißt das, du arbeitest international mit Leuten aus allen Ländern (du hast vorhin Russen erwähnt)?

NOOBODY: Ja, meistens. Dank dem Internet hat man ja heute oft Kontakt mit Leuten aus anderen Ländern, und da ist man mit Englisch gut beraten (obwohl diese Russen mit Englisch ein wenig Mühe haben *g*). Da ich ursprünglich auch plante, meine Website irgendwann als Portfolio zu gestalten, ist es auch eine gute Idee, sie in Englisch zu schreiben.

JOSH: Okay. Vielen Dank für das Interview!

NOOBODY: Die Freude war ganz meinerseits.

Abgeschlossene Projekte

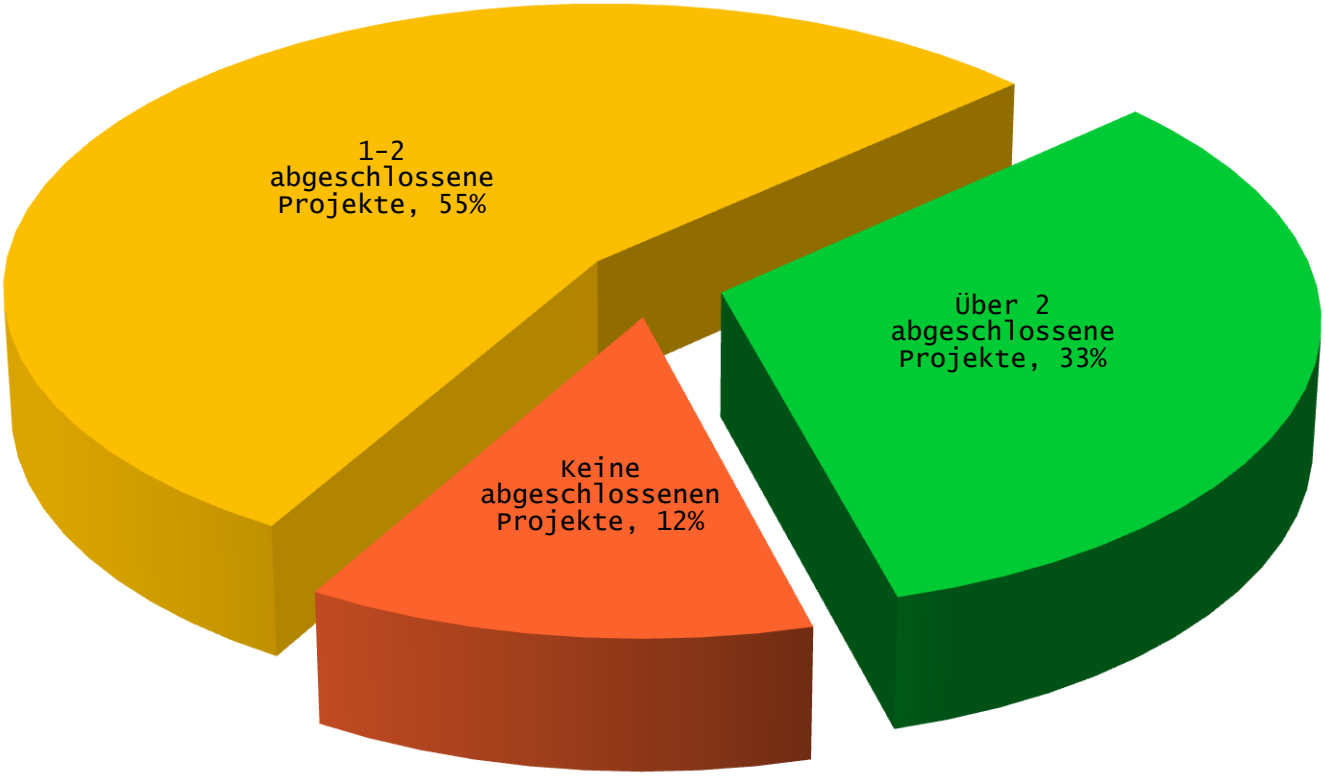

Abbildung 3: Die Community ist produktiv!

Interview mit Midimaster

Josh fragt Midimaster

Das Interview mit Midmaster habe ich geführt, weil mir seine Hilfsbereitschaft gegenüber Anfängern und sein Engagement im Forum aufgefallen sind. Ich habe mich gefragt: Wer ist dieser Mann und was macht er so, wenn er so eine unermüdliche Geduld hat und sogar ausführliche Tutorials schreibt? Nun werdet ihr es erfahren...

Josh: Hallo Midimaster! Bereit?
Midimaster: Jau, leg los!
Josh: Wie alt bist du?
Midimaster: 48
Josh: Hast du studiert? Wenn ja, was?
Midimaster: Jahrgang 63, Abi 83! Studiert habe ich nicht.
Josh: Warum schreibst du so viele Anfänger Tuts in der Community?
Midimaster: Viele Tutorials sind gründlich... oft zu gründlich. Der Anfänger will es gar nicht so genau wissen und schon gar nicht die ganze Wahrheit. Erst mal reicht ein Teil.
Josh: Unterrichtest du auch selbst? In einer Schule oder so?
Midimaster: Ja, ich habe schon während der Oberstufe immer Keyboard in einer Musikschule unterrichtet. Ich schlitterte so ins Musikbusiness rein und übernahm direkt nach dem Abi diese Musikschule und habe 15 Jahre erst mal nur Keyboard unterrichtet. Für die anderen Instrumente hatten wir Lehrer.
Josh: Und dann?
Midimaster: Und dann habe ich eine Software für meine Schüler geschrieben: *MIDILERN*.
Josh: Hast du wegen *MIDILERN* zu programmieren angefangen? Oder hast du auch schon vorher geproggt?
Midimaster: Nee, als wir in der 7. Klasse waren, war unser Jahrgang einer der Ersten, die auf PET[1] programmieren durften. Unsere Schule hatte gleich einen angeschafft und da gab's nur BASIC als Dialekt.
Josh: Was genau kann *MIDILERN*?
Midimaster: Damit konnte man am Atari ST das Notenlesen trainieren. Der Atari hatte gleich eine MIDI-Schnittstelle eingebaut und das legendäre GfA-Basic war superschnell. So konnte man erstmals mit Basic vernünftige Programme schreiben.
Josh: Also ähnlich, wie ein Vokabeltrainer, nur mit Noten?
Midimaster: Ja so ähnlich. Kollegen haben das gesehen und wollten das auch haben und in 1996 waren wir PD Programm des Jahres.
PD meint "Pubilc Domain" und war eine erste Festlegung einer kostenlosen Lizenz. Das war damals so eine Aktion zwischen einigen Fachzeitungen, Karstadt und einem Verein, der gesammelt hat.
Der Verein hat kostenlose Software gesammelt und jeden Monat zu einer Diskette zusammengestellt. Die konnte man dann bei Karstadt für wenig Geld kaufen - gab ja noch kein Internet!
Josh: Wow, kein Internet kann ich mir jetzt schon nicht mehr vorstellen.
Midimaster: Ich mir auch nicht mehr. Alles wurde über Fachzeitungen kommuniziert.
Daten gab's dann in der Computerabteilung von Karstadt, wo man eben die aktuelle PD-Diskette zum Kopieren bereithielt.
Josh: Okay, das war dein Einstieg. Aber jetzt schreibst du auch andere Programme, oder?
Midimaster: Nee, nur Lernsoftware im Musikschulbereich! Aus dem *MIDILERN* wurde der Score-Trainer. Der ist heute die Referenz für Notenlernsoftware und in 2003 glaub

[1] ***P**ersonal **E**lectronic **T**ransactor*, der Erste PC von Commodore und Vorläufer des C64.

ich, kam der Rhythmus-Trainer dazu. Eine Echtzeit-Kontrolle, ob jemand sauber im Beat spielt.

JOSH: Komponierst du auch selber Musik (Spielmusik oder so)?

MIDIMASTER: Ich habe eine Jazz Lounge, Downbeat, Chillout, etc. Band. Wir machen so sanfte Elektro-Sachen mit Latin Touch.
Für die Spiele benötige ich immer wieder "Kompositionen", aber das klingt schon nach mehr, als es ist.
Es sind eben kleine Hintergrundthemen, z.B. Western-Musik für unseren Tatiti.

JOSH: Wie würdest du so die Bedeutung von Spielmusik für ein Spiel einschätzen?

MIDIMASTER: Immens wichtig! Stell dir einen Film vor ohne Musik. Da ist es 100x schwieriger, die Stimmung rüberzubringen.
Wenn du Musik einsetzt, ist den Zuschauern schneller klar, wohin es gehen soll.

JOSH: Klingt logisch.

MIDIMASTER: Zum Beispiel, ob ein Horrorstreifen echt Angst machen soll, oder ob er mehr Genre-Verarschung ist.
Und wer Musik sagt muss auch Geräusche sagen!
Ohne die geht es gar nicht. Erst die machen aus der Explosion eine glaubwürdige Explosion.

JOSH: Stimmt, sonst wäre das ja wie ein Stummfilm, nur mit Musik.
Und gibt es irgendwas, was man bei der Produktion von Spielmusik vermeiden sollte?

MIDIMASTER: Naja, was meinst du mit "vermeiden"? Sie muss halt immer für das Spielthema geschrieben sein.

JOSH: Und wie/wann bist du dann zu BB gekommen?

MIDIMASTER: Nachdem die Atari ST Zeit langsam zu Ende ging, stieg ich auf PC um, und musste für Jahre mit dem grässlichen VB3 und VB6 arbeiten.
In 2008 erst entdeckten wir BMax, weil für unseren Tatiti die Performance von VB nicht mehr ausreichte. Außerdem klang es für uns klasse, auch für den Mac entwickeln zu können.
BB hab ich also nie genutzt. Und B3D erst nach BMax.
Und das auch immer nur, um es dann anschließend gleich wieder in MiniB3D umzusetzen.

JOSH: Und arbeitest du heute immer noch nur mit BMax/BB/B3D? Oder auch in anderen Sprachen?

MIDIMASTER: Alle Programme, die ich verkaufe, sind immer in BMax.
Für eigene Zwecke brauche ich gelegentlich VB6 und PHP. Monkey hab ich gekauft, aber noch nix großes damit herausgebracht.

JOSH: Also verdienst du wirklich mit dem Programmieren freiberuflich genug, um davon zu leben?

MIDIMASTER: Ja, ich verdiene mein Geld damit und man kann davon leben.

JOSH: Wenn man dich googelt, kommt man auf über 1.000.000 Ergebnisse und sofort auf deine Website. Hast du irgendwann mal Werbung gemacht oder so?

MIDIMASTER: Naja, die Marke *MIDIMASTER* ist in dem Bereich schon was und wird auch immer wieder verlinkt. Viele Musikschulen halten viel von der Software.
Außerdem gebe ich bei Google 300 €/Monat für Adwords aus. Eine Kampagne, die einen auf die erste Seite bringt, wenn bestimmte Suchwörter auftauchen.

JOSH: Danke für das Interview nochmal!

MIDIMASTER: Schön! Viel Erfolg beim MAG!

Altersgruppen im Forum

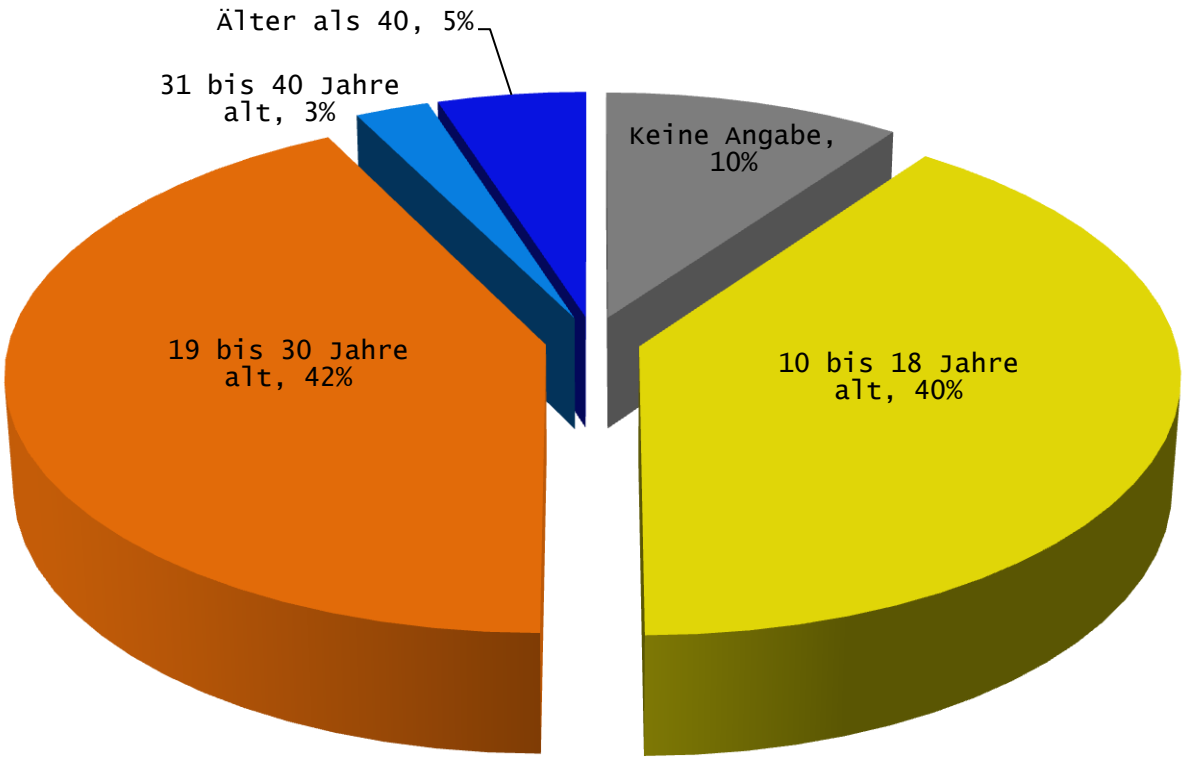

Abbildung 4: Eine gute Mischung aus Jung und Alt.

Beginner-Interview

Hagbard fragt BaseTH

Hagbard: Hey, es ist Punkt 15:30. Brauchst du noch Zeit?

BaseTH: Nö, 3,2,1 go.

Hagbard: Wie läuft's bei deinem Projekt? Du bist ja einer der wenigen, die gerade überhaupt aktiv sind.

BaseTH: Welches meinst du denn? Wenn es um den Spaceminator geht, da kam gestern ja ein neuer Worklog-Eintrag. Komme so voran wie ich es mir erwünsche. Gibt natürlich auch mal Momente wo ich mir denke „Hä warum geht das nicht.“, aber das kennt wohl jeder von uns. Was ich persönlich sehr sehr schade finde, sind Worklogs, die gestartet werden, aber dann doch nichts mehr kommt.

Hagbard: Wie beurteilst du die Aktivität im BB-Forum momentan? Oder sagen wir: für den Zeitraum, in dem du nun dabei bist.

BaseTH: Naja, dabei bin ich ja seit dem 01.04.2010 und habe ca. 6 Monate später angefangen auch mal was zu sagen. Also ich denke es ist und bleibt ein Kern von Usern die aktiv sind. Es melden sich zwar immer mal wieder neue Leute an, aber die meisten verschwinden nach einem Monat. Ist wie bei den Worklogs, es wird was gestartet und schwupp – das Ende ist da.

Hagbard: Persönlich merke ich, dass meine Motivation erheblich höher ist, wenn ich merke, dass im Forum viel los ist: Viele Projekte, Diskussionen, Ideen usw.

BaseTH: Genau, dass macht ein gutes Forum doch auch aus, Nutzer die aktiv diskutieren und ihre Ideen austauschen bzw. ihre Projekte vorstellen und zumindest ansatzweise beenden. Ich finde es z.B. sehr motivierend wenn andere meine Beiträge kommentieren. Auch wenn mal was Negatives dabei sein sollte ist es schön. Denn jeder sollte seine Meinung äußern können und aus Kritik kann ich doch auch lernen.

Hagbard: Gibst du dir Mühe, im Forum aktiv zu sein, anderen zu helfen oder Rückmeldung zu geben? (provokant gefragt, machst du es anders als andere?)

BaseTH: Also, wenn ich sehe, da ist eine Frage, die ich beantworten kann, oder Hilfe meinerseits ist möglich, dann antworte ich gerne im Forum.
Sollte die Antwort bereits durch einen anderen User erfolgt sein, dann lasse ich es in der Regel, da die Antwort ja schon gekommen ist. Meine Postingzahl muss ich nicht tunen.
Ich antworte in erste Linie da ich ja auch durch Beiträge im Forum lerne, obwohl ich selber bisher keine Frage wie z.B. wie mach ich das oder so stellte.
Es ist ein Geben und Nehmen. Wenn ich nehme sollte ich auch geben können. Zum aktiv sein kann ich nur sagen, ich schaue jeden Tag mal kurz rein.

Hagbard: Zum Thema Fragen: Gehörst du eher in die Kategorie "Erstmal-im-Forum-fragen" oder "Selbst-probieren"? Und: wie gehst du dabei vor?

BaseTH: Als erstes überlege ich mir, „was will ich machen?“ Dann probiere ich es erst mal aus. Sollte es dann nicht klappen, schaue ich in meine Bücher und die Tutorials die ich hier ausgedruckt habe.
Natürlich schau ich auch im Forum nach, bisher gab es noch keine Frage die nicht schon einmal geklärt wurde. Daher sieht man bisher auch keine Fragen von mir. Denn mit der Forensuche kommt man sehr weit was, meiner Meinung nach, nicht jeder macht.

Hagbard: Wie gehst du beim Programmieren lernen vor? Hast du größere Projekte, in denen kleinere Probleme auftreten? Oder ist es eher so,

dass du dir gezielt typische Fälle (GUI, Input-Routine, Spiel- Physik usw.) heraussuchst und dich daran versuchst?

BASETH: Ich habe hier so ein paar Projekte laufen, die meisten kennst du ja schon vom Treffen. Ich setzte mir als erstes ein Ziel – derzeit halt den Spaceshooter – und programmiere dann los. Vorher steht natürlich im Kopf das Konzept, was im Programm sein soll. Einzelne Dinge wie z.B. Levelgestaltung und so wird natürlich in kleineren Testprogrammen ausgetestet. Für Level wird ein selbstgeproggter Leveleditor verwendet (Breakout-Klon). Bei dem DVD-Verwaltungsprogramm machte ich mir erst mal Gedanken über welche Suchparameter später eine Suche möglich sein soll, erst danach habe ich begonnen die Eingabemaske zu entwerfen.

HAGBARD: Klingt sehr vernünftig. Könntest du dir vorstellen, es anders zu machen? Einfach mal drauf los programmieren, ungeplant?

BASETH: Kurz gesagt, ich Plane gerne im Voraus. Ich hatte früher die ersten Projekte ohne ein wirkliches Konzept einfach mal begonnen, musste dann aber feststellen, dass Änderungen sehr, sehr schwer sind. Daher geht nun alles nach Plan.
Es kommt da aber natürlich auch mal vor, dass ich noch eine Idee habe, die mit rein soll. Die kann dann auch mal dazu führen, dass ein größerer Teil des Quellcodes geändert werden muss. Aber nur durch solche Sachen lernt man. Nur durch Fehler werden wir und Projekte besser.

HAGBARD: Weil du gerade schon so nützliche Empfehlungen aussprichst, fünf Dinge/Tipps, die du anderen Beginnern rätst. Bitte kurz!

BASETH: 1. Fangt mit kleinen Projekten an.
2. Denkt auch mal länger als eine Minute nach und bleibt ruhig.
3. Nutzt die Forenssuche, denn dort steht sehr viel.
4. Macht nur Projekte öffentlich, die ihr auch ernst meint, sonst werden euch die User ggf. meiden / nicht antworten.
5. Und der wichtigste Punkt: Habt Spaß bei der Sache und schreibt nicht ab, denn das hilft euch nicht, euch zu entwickeln. Also Forensuche, Text lesen und verstehen und dann selber neu schreiben.

HAGBARD: Sehr gut. Wie leicht fiel/fällt dir der Einstieg selbst?

BASETH: Wenn mich eine Sache interessiert, dann lerne ich bis zum Umfallen, das ist jedoch nicht gut für den Körper.
Der Einstieg in BlitzMax war super, denn es ist meiner Meinung nach eine Programmiersprache die sehr leicht zu erlernen ist. Habe aber auch schon Kenntnisse durch BlitzBasic3D gehabt, auch leicht zu erlernen und vorher war halt QBasic.
Also ich programmiere mehr oder weniger seit der fünften Klasse. Aktiv allerdings erst seit ca. 2008. Die Blitz-Dialekte sind, wie gesagt, für Anfänger meiner Meinung nach perfekt, denn leichter geht es wohl kaum noch.

HAGBARD: Wie bist du damals auf BB gestoßen? Lag es an der sprachlichen Nähe zu Qbasic? Oder in der Schule gelernt?

BASETH: Ich hatte mal etwas für QBasic gesucht und dann im Netz BB gefunden.
Dann war dort noch ein Link zu dem bekannten Buch "Spiele programmieren mit BlitzBasic". Da dieses sehr günstig war, dachte ich mir: Los geht‘s.
Mal so nebenbei, es sollte übrigens regelmäßig solche Treffen wie in HH geben, das wäre nice.

HAGBARD: Guter Punkt, auch für die Motivation, oder? Geht mir jedenfalls so.

BASETH: Auf jeden Fall auch das. Man lernt die Leute kennen, sieht Dinge die man vielleicht nicht im Kopf hatte und

kann auch einiges lernen. Außerdem fördert das meiner Meinung nach auch die Aktivität im Forum.
Fand es übrigens recht geil, dass der Count-Doku gefahren wurde. Sehr nette Sache.

HAGBARD: Welche Leute kennst du auch in Wirklichkeit, die BB programmieren?

BASETH: Also von meinen Bekannten keine mehr, gab mal welche die es probiert haben, aber sich nicht fürs proggen begeistern konnten.

HAGBARD: Beim Treffen waren ja einige dabei, die beruflich programmieren, manche tatsächlich Spiele, manche anderes. Kannst du dir vorstellen, auch beruflich nochmal in die Richtung zu gehen?

BASETH: Denke eher nicht, dass es dazu kommt, jedenfalls nicht in einem Unternehmen in dem ich Aufgabe eins, zwei usw. umsetzen soll. Rein privat ein kommerzielles Projekt zu realisieren – denke ich schon, dass ich das mal versuchen werde. Aber derzeit muss ich ja erst mal ein paar Projekte beenden.

HAGBARD: Zum Abschluss noch ein kleines Spiel:
Fünf kurze Fragen, fünf kurze Ein-Wort-Antworten. Los geht's!
Buch oder Tutorial?

BASETH: Das ist gemein. Beides.

HAGBARD: 2D oder 3D?

BASETH: 2D.

HAGBARD: Teamwork oder Einzelkämpfer?

BASETH: Bezug aufs Forum "Einzelkämpfer" sonst gerne Aufgabenteilung.

HAGBARD: BB-Community: Forum oder Chat?

BASETH: Forum.

HAGBARD: Projekt oder "kleine Aufgaben"?

BASETH: Projekt.

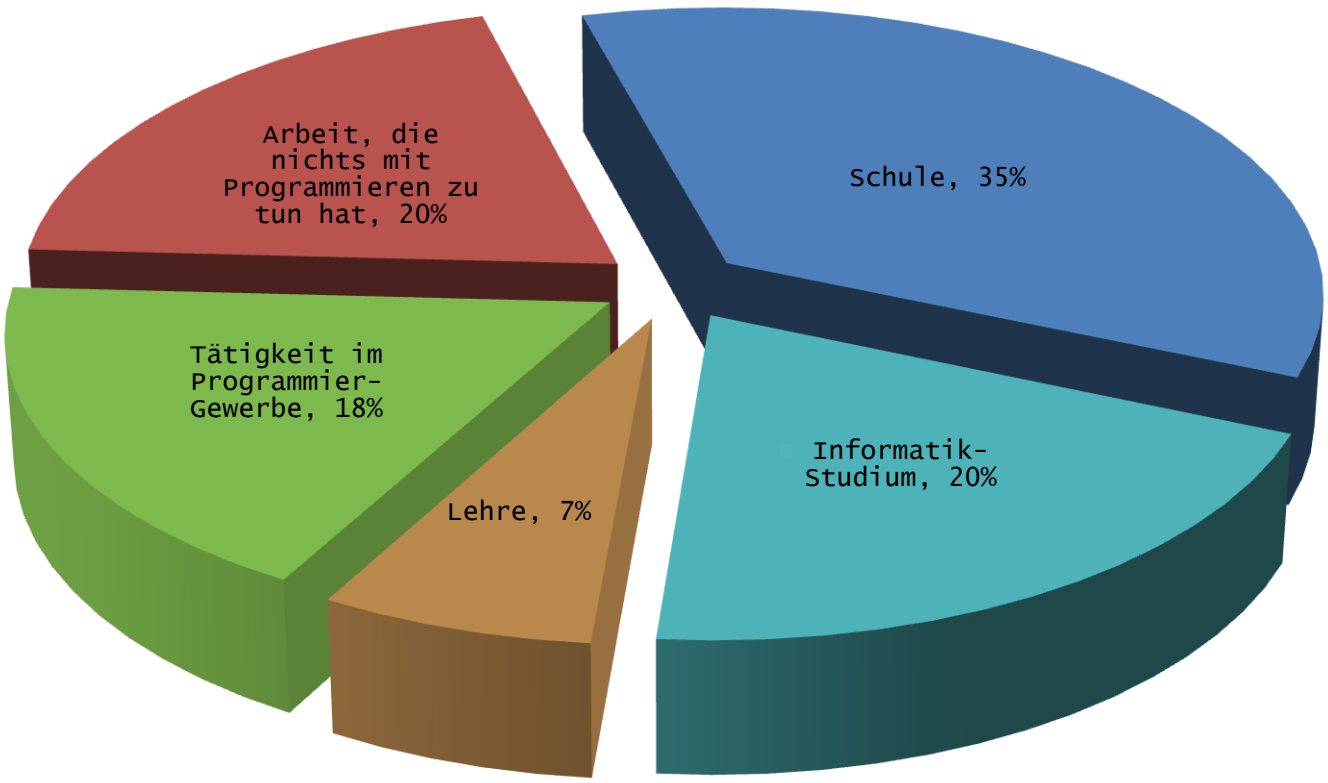

Abbildung 5: Hobbyisten & Profis.

Das Hamburg Treffen 2011 – Wenn Programmierer aufeinander hocken

Von Mr.Hyde

Teilnehmer: Koemterion, BaseTH, count-doku, Firstdeathmaker, Hagbard, sein Bruder, jsp, sein Sohn und Mr.Hyde.

Ein Aufruf im Forum „Wer kommt aus der Nähe von Hamburg und hat Lust auf ein Treffen?“, meistens fängt es so an, natürlich auch dieses Mal, als Hagbard die entscheidenden Worte postete. Schnell fanden sich ein paar Teilnehmer und bald war beschlossen: Wir treffen uns in Hamburg!

Am 26.11.2011 trafen wir uns dann bei koemeterion, der sogar extra für uns aufgeräumt hatte. Seine Wohnung in Hamburg-Hamm sollte also für die nächsten Stunden zu so etwas wie einem Programmierer-Paradies werden.

Sofas, Sessel, ein überdimensionaler Schreibtisch und ein Präsentationsbildschirm waren vorbereitet. Koemeterion hatte sogar gelüftet, damit es nicht zu verqualmt riecht.

Für 12:00 waren wir verabredet – und tatsächlich tröpfelten nach und nach die Leute ein. Es sollte allerdings noch 2 Stunden dauern, bis wir vollzählig waren. Vollzählig bedeutete auch gleichzeitig, dass mehr nicht gegangen wäre. Selbst die Plätze auf dem Boden waren belegt. Steckdosen wurden rar und die Raumluft stieg jede Minute um gefühlte 2 Grad – obwohl alle nur Laptops mitgebracht hatten.

Trotzdem dauerte es eine Weile, bis auch die Blitzer miteinander warm geworden waren. Besonders bei Ersttreffen wie diesem braucht es besonders viel Zeit. Was wäre da besser geeignet als eine Projektvorstellung. Wie sonst, will man schließlich einen Entwickler kennenlernen?

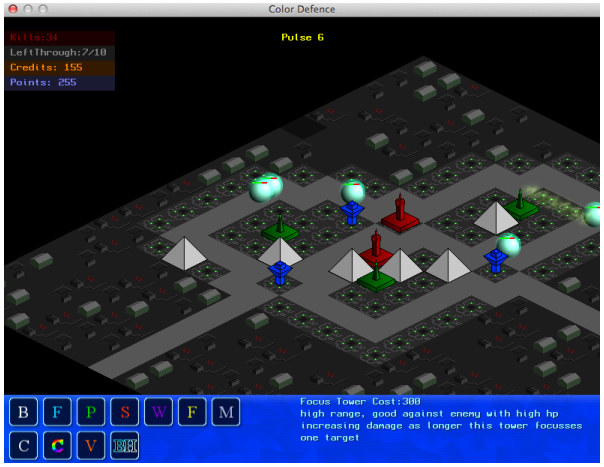

Abbildung 6: Color Denfense

Firstdeathmaker, eigens aus Berlin angereist, machte den Anfang. Sein Projekt: ein Tower Defense für den zu dieser Zeit aktuellen Code-Wettbewerb (BCC #57) . Der Clou ist, die farbigen Tower so zu platzieren, dass alle vorkommenden Gegnerfarben bekämpft werden können. Je größer die Farbdifferenz ist, desto stärker wirken die Tower. Ebenso besonders ist die Kombination aus frei zu bebauendem Maze und fester Streckenführung. Toll auch die Effekte der Tower, wobei hier zu erwähnen ist, dass diese zum Zeitpunkt des Treffens nur rudimentär implementiert waren. Auch der hier gezeigte Screenshot stammt schon aus der Veröffentlichungsversion.

Nicht nur „Color Denfense“, so der Titel, faszinierte die Blitzer, auch ein paar alte Projekte von Firstdeathmaker

wurden hervorgekramt und entstaubt. Das Game of Life mit lernenden Kulturen ist zwar schon etwas älter, machte beim Beobachten trotzdem Spaß.

BaseTH zeigte einen Breakout-Klon, der sich derzeit allerdings in einer Entwicklungspause befindet, da noch grafische Arbeiten erledigt werden müssen, die er allerdings nicht selbst erstellt und nun auf den Grafiker warten muss. Stattdessen arbeitet er an einem Spaceshooter, damit geht er den nächsten Spiele-Klassiker-Typus an.

Einen absoluten Klassiker konnte Count-Doku vorstellen. MineSweeper und den auch noch als Multiplayer in verschiedenen Modi, wie Coop oder eine Art Schiffe versenken, indem man seinem Mitspieler die Minen platziert und umgekehrt. Aber auch hier war die Entwicklung noch nicht abgeschlossen. Wir dürfen gespannt sein, wann das Projekt fertig wird.

Um 19:00 war das Treffen dann auch ohne große Aufreger wieder vorbei und nach und nach tröpfelten die Blitzer wieder heim.

Bis zum nächsten Treffen.

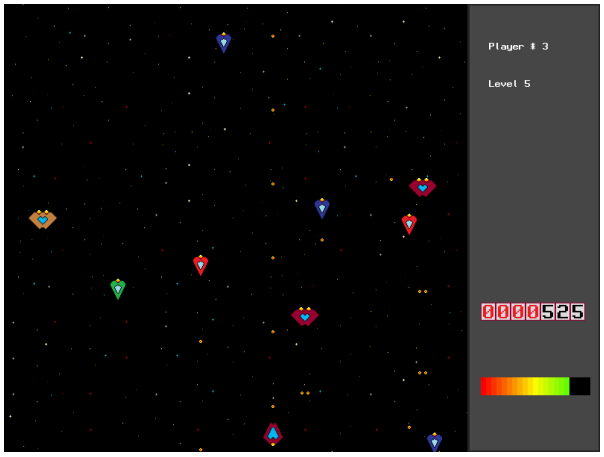

Abbildung 8: Spaceminator

Abbildung 7: MineSweeper MP

WAS SICH IM SHOWCASE VERBIRGT

VON BASETH

Was verbirgt sich eigentlich hinter dem Showcase?

Nun, der Showcasebereich ist ein Sammelwerk verschiedener Projekte, die entweder sehr weit fortgeschritten oder bereits komplett abgeschlossen sind. Derzeit werden dort über 250 Beiträge präsentiert. Auch wenn dort einige Anwendungsprogramme zu finden sind, beschäftigen sich die meisten Einträge mit etwas spielbarem. Diese Spiele verteilen sich auf unterschiedliche Genres.

Im Bereich der Geschicklichkeitsspiele finden wir u.a. die Snake-Klone „BIKILA“ von Lucius, „SN4KE“ von Cornelius und „SimpleSnake“ von MasterSolaris. Aber

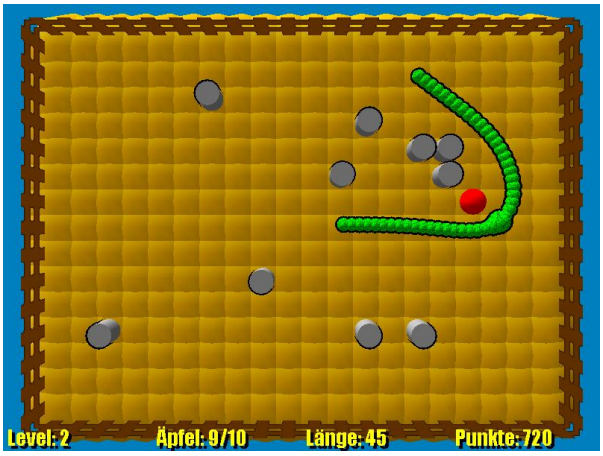

Abbildung 9: SN4KE – fressen im Comiclook.

auch andere Spieleklassiker wurden von den Mitgliedern der Community neu umgesetzt. So finden wir die Spiele „Bial – Bricks in a Line“ von DaysShadow, welches einen Tetris-Klone darstellt und „Pac The Man“ von coolo – einen Vertreter der Pac-Man-Klone.

Wer lieber läuft und springt kann eines der Jump & Run Spiele spielen. Ein sehr interessantes Konzept hat dabei das Spiel „Bob the Blob“ von Pousup. Hier spielt man „Bob“, eine farblose Kugel, die durch eine teilweise farblose Welt zu seiner Freundin

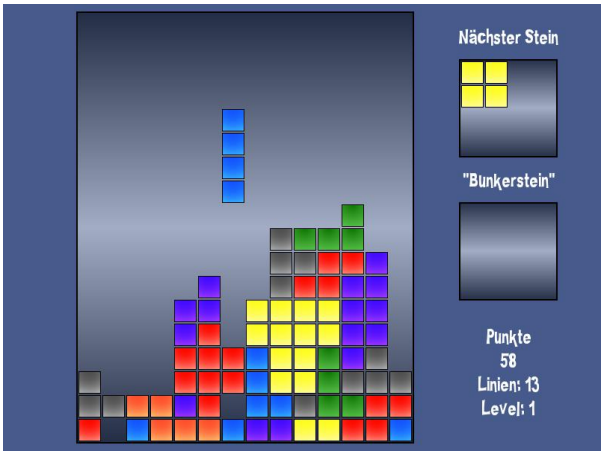

Abbildung 11: Bricks in a Line.

gelangen muss. Das Problem für Bob ist dabei, dass er im farblosen Zustand die farblosen / schwarzen Bereiche der Welt nicht berühren kann. Erst nach dem Bob etwas Farbe eingesammelt hat, ist es ihm möglich diese Bereiche zu betreten. Dabei muss der Spieler jedoch darauf achten, dass er nicht zu viel der farblosen Welt betritt, denn immer wenn Bob das macht, verliert er einen Teil seiner Farbe, um die Welt zu färben. Zusätzlich gibt es auch Stacheln

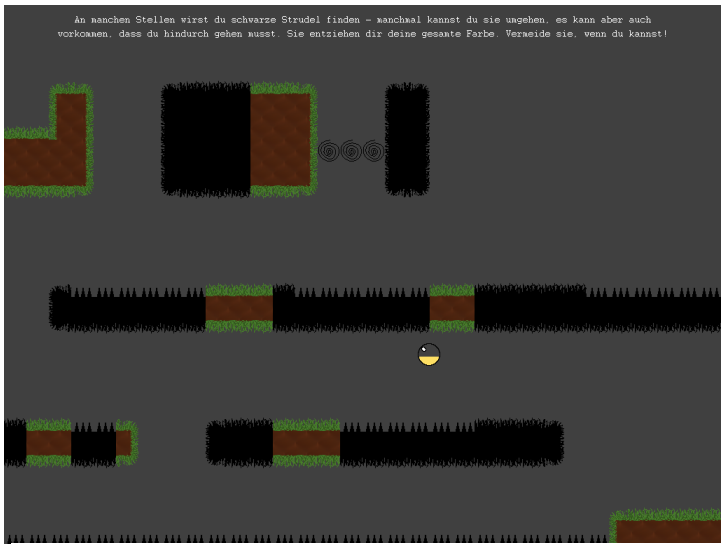

Abbildung 10: Bob the Blob in Aktion.

die Bob lieber nicht berühren sollte und auch ein Symbol, durch das er seinen gesamten Farbvorrat verliert. Es ist also auch hier Vorsicht für unsere Kugel geboten. Da es dem Entwickler hier primär um die Ausarbeitung eines interessanten Konzeptes ging, ist dieses

Spiel extrem kurz, sodass wir unsere Freundin bereits in unter 5 Minuten erreichen können. Ob es von diesem Konzept auch einmal ein größeres Spiel mit mehreren Level und auch Gegner geben wird, ist bisher ungewiss. Meiner Meinung nach ist das jedoch wünschenswert, denn das Konzept mit der farblosen Welt ist sehr gelungen.

Wenn ihr etwas mehr Action braucht, dann ist der Bereich der Actionspiele genau das richtige für euch. Ein schöner Vertreter eines 2D-Spaceshooter im Stil von R-Type bildet dabei das Spiel „Argon“ von Sirius7. Dieses Spiel beinhaltet vier Level, in denen viele Gegner beseitigt werden müssen, bevor man am Ende der jeweiligen Level Bekanntschaft mit einem Endgegner macht. Auch wenn in diesem Spiel so manches Upgrade an Waffen durch einsammeln zur Verfügung gestellt wird, finde ich es recht schwer. Sollte man getroffen

Abbildung 12: Argon

werden, verliert man eine Upgradestufe und die vorhandene Feuerkraft wird wieder schlechter. Optisch und akustisch finde ich dieses Spiel recht stimmig gemacht, sodass bei mir Erinnerungen an alte Spieltage wach werden.

Natürlich sind auch andere Spielgenres wie z.B. Adventure, Sport und Strategie vertreten, sodass sich ein Besuch im Showcasebereich durchaus lohnt, denn dort gibt es noch vieles zu entdecken.

Die Magie von Git und GitHub

Von Propellator

Was ist Git?

Git ist ein Versionskontrollsystem, entwickelt von Linus Torvalds. Es merkt sich Änderungen am Code, ermöglicht das Verfolgen verschiedener Versionen und das Zusammenführen derer, sowie das Rückgängigmachen einer Änderung und vieles mehr.

Was ist GitHub?

GitHub ist eine Website, welche das Hosten von Git-Repositories ermöglicht und zusätzliche Features wie einen Bugtracker (Service zum Melden und Kategorisieren von Problemen in der Software, oder Anregungen), ein Wiki usw. anbietet. Ebenfalls dient es als eine Art „Soziales Netzwerk“ für Programmierer, in welchem ihr andere Projekte und Coder verfolgen könnt, um zu sehen was sie so treiben. Das Beste am ganzen Spaß ist, dass er gratis ist, sofern der eigene Projektsourcecode für jeden öffentlich einsehbar und herunterladbar sein darf.

Was ist ein Distributed Version Control System?

Git ist ein sogenanntes Distributed Version Control System, das heißt: Jeder, der sich das Repository (Eine „Lagerstelle“ des vom Version Control System überwachten Codes) kopiert (also, z.B. herunterlädt) hat automatisch ein eigenes Repository. Somit können viele Leute gleichzeitig an einem Projekt arbeiten, ohne dass sie groß Rücksicht auf andere Mitwirkende nehmen müssen.

Wer ist der Autor dieses Artikels, und hat er überhaupt praktische Erfahrung?

Ich beobachte schon seit einiger Zeit kleine bis mittelgroße Open Source Projekte, und habe teils sogar schon zu ihnen beigetragen. Als Versionskontrollsystem wurde in vielen dieser Projekte Git eingesetzt, und teils wurde auch GitHub als Ergänzung verwendet, um den Arbeitsprozess einfacher zu machen, und eine reibungslose Kommunikation zwischen Benutzern und Entwicklern zu ermöglichen.

Über diesen Artikel

Der Artikel besteht (leider) größtenteils aus Theorie. Dies ist hauptsächlich aus dem Grund, dass es schon haufenweise praxisbezogene Guides im Internet gibt, welche ich auch verlinkt habe. Diese sind meist besser geschrieben als ich es könnte, von da her dient dieser Artikel lediglich als „Appetit-Macher“.

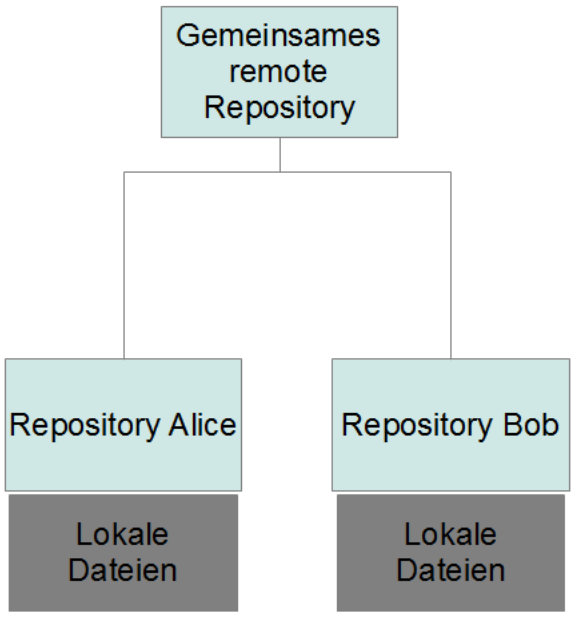

Abbildung 13: Alice und Bob haben jeweils ein Repository, welches verbunden ist mit dem remote Repository.

Der Aufbau von Repositories mit Remotes, Commits und Branches

Ein Repository ist nichts anderes als euer Projekt, und zusätzlich die Daten zu allen Änderungen, die ins Repository

eingereicht wurden. Diese werden „Commit“ genannt.

Wenn jemand mit einer Änderung des Codes fertig ist, kann er diese mit einem Commit im Repository vermerken. Git vergleicht anschließend die zwei Versionen, merkt sich die Änderungen und generiert für den Commit eine Prüfsumme. Durch diese Prüfsumme kann auch niemand etwas in eine Änderung schmuggeln, ohne dass es Git bemerken würde.

Der Commit selbst zeigt auch immer auf den (oder die) übergeordneten Commit, kennt jedoch nicht den nachfolgenden Commit. Dies ist wichtig wenn wir zum Thema der Versionskonflikte kommen.

Anschließend kann der Programmierer seine Änderungen (falls er dies möchte!) in ein anderes Repository einreichen (pushen), sofern er Schreibzugriff darauf hat. Die „anderen Repositories“ werden „Remotes“ genannt, da sie sich (meistens) auf einem entfernten Rechner befinden.

Möchte der Programmierer nun sicherstellen, dass er die neueste Version des Codes hat, so kann er sich die neuen Änderungen von einem beliebigen Remote holen (pullen).

Das Schöne an der Sache ist auch, dass ein Git Repository problemlos auch auf mehrere Remotes zeigen kann. Ebenfalls ist es nicht zwingend, einen Remote zu haben. Man kann also auch sein eigenes Süppchen kochen.

Will man nun z.B. ein neues Feature in das Projekt einbauen, aber dabei sicherstellen, dass währenddessen eine stabile Hauptversion während der Entwicklungszeit bestehen bleibt, so kann man dies mit sogenannten „Branches“ (zu deutsch: Äste) erreichen.

In jedem Repository existiert mindestens ein Branch, der master-Branch. Will man nun eine größere Änderung am Projekt vornehmen, ohne befürchten zu müssen dass der Rest des Projekts nicht mehr voran geht, so wird sich ein Programmierer einen neuen Branch erstellen. Der Branch enthält dann eine eigene Versionsgeschichte, welche aber auf einem gewissen Commit basiert.

Änderungen, welche an anderen Branches vorgenommen werden, können (meist) problemlos für andere Branches übernommen werden. Die Magie dahinter übernimmt Git. Hat man die Arbeit in einem Branch abgeschlossen, so kann der Branch mit Haupt-Branch (master) zusammen gefügt werden. Wie das funktioniert, erklärt der nächste Abschnitt.

Versionskonflikte und andere Magie

Nehmen wir an, Alice hat am 1.3.2011 ihr lokales Repository mit dem Remote-Repository eines kleineren OpenSource-Projekts abgestimmt. Alice hat nun ihre Änderungen vollbracht und will diese natürlich einreichen, jedoch hat in der Zwischenzeit bereits Bob andere Änderungen vorgenommen. Dies wird Alice dadurch gemeldet, dass Git sie beim Pushen warnt, dass es weitere Commits gibt, welche auf den letzten gemeinsamen Commit zeigen.

Anstatt Bob zu erwürgen, pullt Alice nun einfach die Änderungen des Remotes in ihr lokales Repository, genauer ihren lokalen Branch in ihrem lokalen Repository. Git „merged“ dann die beiden Versionsgeschichten, fügt sie also zusammen. Dies tut es, indem es einen neuen Commit erstellt (Den sogenannten „Merge-Commit“), dieser hat dann die übergeordneten Commits (die sogenannten „parents“) den letzten Commit von Alice und den letzten Commit von Bob. Nun kann Alice ohne Versionskonflikte ihre Änderungen auf dem Remote pushen, und Bobs Luftröhre bleibt intakt.

Dieses Szenario trifft auch ein wenn ich in einem Repository 2 Branches

habe, und diese zusammenführen will. Anstatt zwei Repositories mit je einem Branch habe ich dann ein Repository mit 2 Branches. Dabei muss ich dann die Änderungen aber nicht noch Pullen, sondern die beiden Branches werden direkt lokal gemerged.

Das Mergen in Sachen Dateiänderungen betrachtet, welches Git für uns übernimmt, ist natürlich komplexer, jedoch benötigen wir dieses Wissen nicht um Git zu benutzen, also lassen wir Magie einfach mal Magie sein.

Git schön und gut, aber wie und woher?

Um Git zu benutzen braucht ihr nicht zwingend irgendeinen Server, der als Remote agiert. Ihr könnt auch einfach – solltet ihr alleine arbeiten – Git einfach nur lokal benutzen. Dies hat den Vorteil, dass man immer eine komplette Versionsgeschichte hat.

Git gibt es für praktisch alle Plattformen; für Linux findet man Git in praktisch jedem Paketmanager. Für andere Plattformen, ist die folgende Seite sehr hilfreich: *git-scm.com*

Obwohl man als Git Remote im Prinzip jeden Host benutzen kann, welcher einen SSH-Daemon aufgesetzt hat, werden wir hier konkret mit GitHub arbeiten.

Der Einstieg in GitHub

Ich könnte hier mehrere Paragraphen schreiben mit Erklärungen, jedoch hat GitHub schon sehr gute Guides zum Thema geschrieben. Nach dem erstellen eines Accounts auf *github.com* könnt ihr einfach mal hier vorbeischauen: *Set Up Git*

Zusätzliche Features von GitHub

Neben der Möglichkeit Git Repositories zu hosten, bietet GitHub natürlich eine ganze Menge cooler Features, um OpenSource-Entwicklung zu vereinfachen.

Das 'Social Coding'-Netzwerk

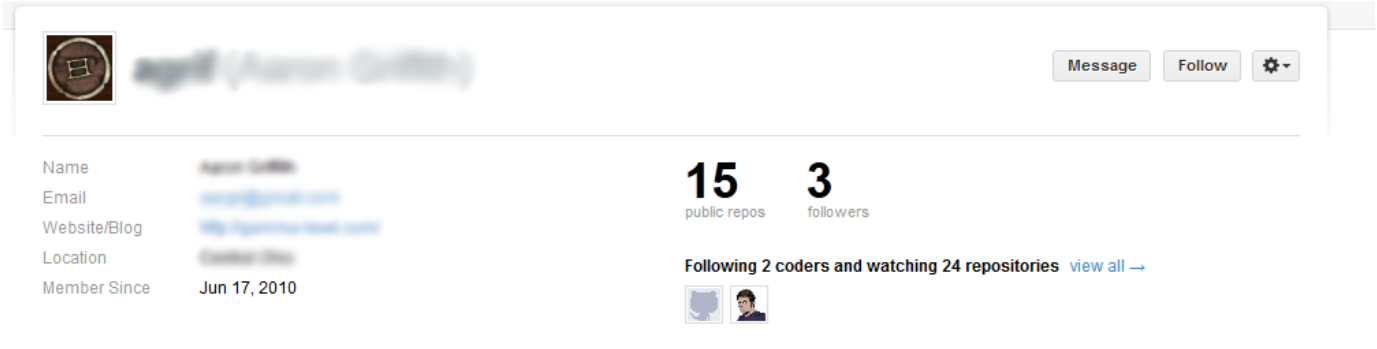

Abbildung 14: Das Profil eines GitHub Users.

GitHubs soziales Netzwerk ermöglicht es dem Benutzer, anderen Benutzern oder Projekten zu folgen (d.h. sich immer über ihre neusten Aktivitäten informieren). Natürlich kann man auch diesen Benutzern Nachrichten senden, und so weiter. Glücklicherweise ist keine „Anstupsen"-Funktion vorhanden.

Das Dashboard

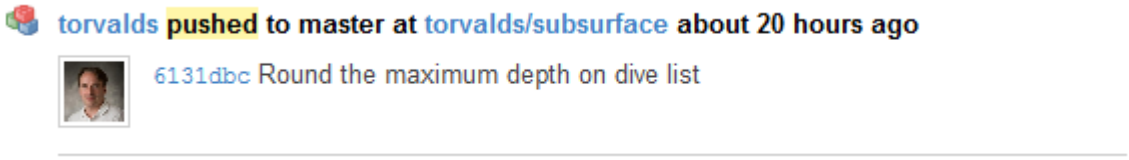

Abbildung 15: Linus pusht in seinen master-branch.

Das GitHub Dashboard zeigt euch die neuste Aktivität der Coder und Repositories an, welche ihr verfolgt. Wenn ihr auf GitHub angemeldet seid, ist das Dashboard automatisch eure Startseite.

Graphen und Statistiken

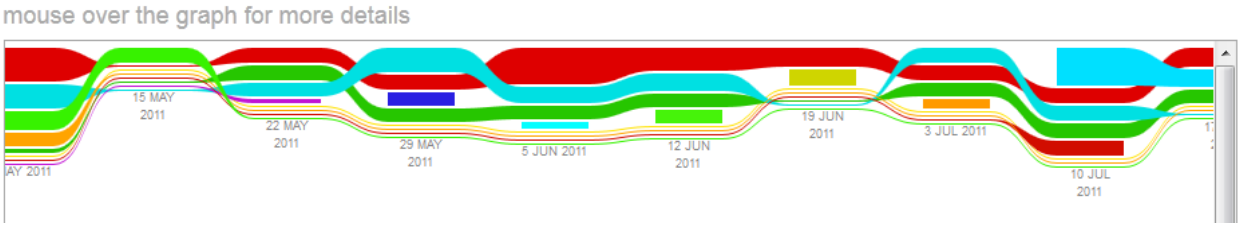

Abbildung 16: Der sogenannte "Impact"-Graph.

GitHub bietet eine Vielzahl an Graphen und Statistiken zu euren Repositories.

Einfache Administration

Mit GitHub können Repositories einfach verwaltet werden; dazu gehört das Hinzufügen von schreibberechtigten Benutzern, Umbenennen des Repositories, verwalten von Pull-Requests (eingereichte Änderungen von nicht schreibberechtigten, welche ins Repo gepullt werden können), Service Hooks zum Ausführen von bestimmten Aktionen wenn neue Commits auftreten und so weiter.

Issues

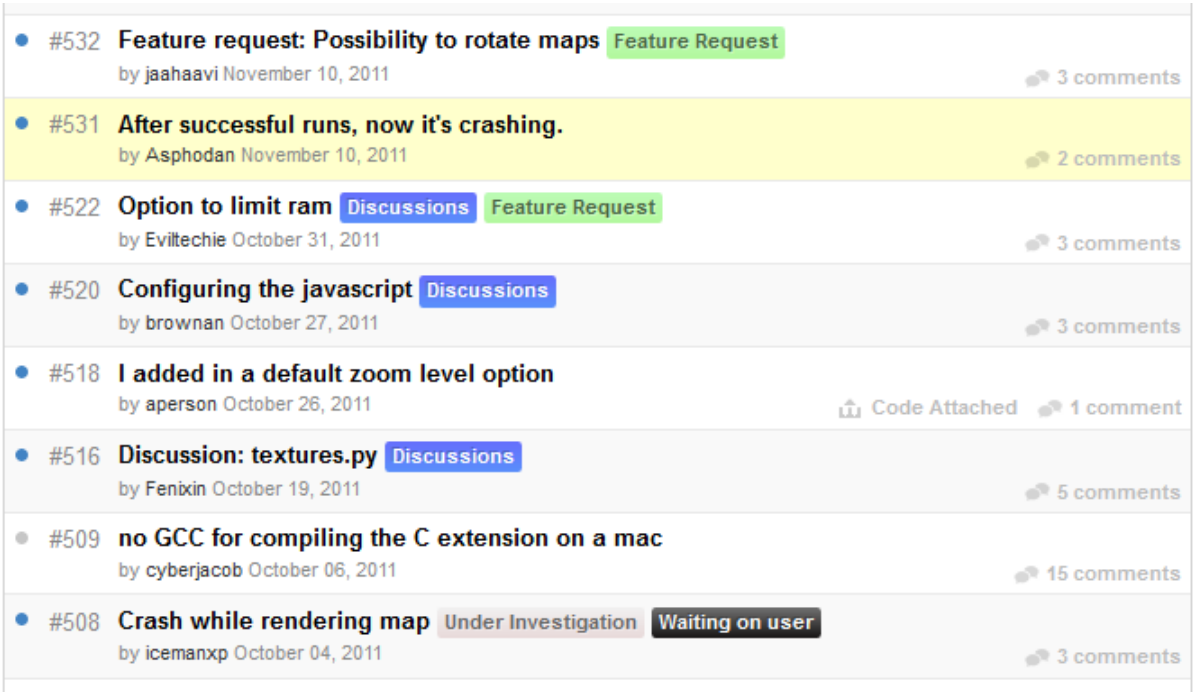

Abbildung 17: Offene Issues

Mit „Issues“ können Bug-Reports und Diskussionen einfach abgehandelt und geschlossen werden.

Issues können sogar per Commits geschlossen oder erwähnt werden; dies wird gemacht in dem man einfach die Issue-Nummer mit einem entsprechenden Begriff in der Commit-Nachricht erwähnt. z.B.:

Abbildung 18: Issue #496 wurde per Commit geschlossen.

Wiki

Jedes Repository auf GitHub hat (falls gewünscht) auch ein eigenes Wiki zur Verfügung, welches sich besonders gut eignet um Anleitungen für die Software bereitzustellen.

Neben diesen oben erwähnten Features bietet GitHub noch einige weitere, welche sehr nützlich sind.

Die Macht der Rekursion

Von Thunder

Definition: Rekursion ist ein Mittel in der Programmierung, das angewandt werden kann, wenn dieselbe Operation mit verschiedenen Parametern durchgeführt werden soll. Die Basis für Rekursion sind Funktionen, sie funktioniert also in allen Programmiersprachen, die Funktionen unterstützen[2].

Gegenteil: Iteration[3] bzw. iterativ.

Aber was ist denn Rekursion jetzt?

Eine Funktion, die sich selbst mit geänderten Parametern aufruft, aber Abbruchbedingungen eingebaut hat, sodass diese Aufrufe auch einmal ihr Ende haben und die Funktionen nacheinander terminieren, nennt man rekursiv. Man spaltet eine Aufgabe in mehrere gleiche Teilaufgaben auf und lässt ein und dieselbe Funktion die ganze Aufgabe durch Selbstaufrufe lösen.

Ich möchte zwei Beispielfunktionen zeigen, die beide die n-te Fibonacci-Zahl berechnen können (ausgehend davon, dass sie so beginnt: 0, 1, 1, 2, 3 ...):

```
'REKURSIV
Function fibo:Int(n:Int)
    If n<=2 Then    ' Abbruchbedingung
        Return n ' kein Selbstaufruf
    Else
        Return fibo(n-1)+fibo(n-2)
    EndIf
EndFunction
```

```
'ITERATIV
Function fibo:Int(n:Int)
    Local a:Int=1, b:Int, tmp:Int
    Repeat
        tmp=a
        a:+b
        b=tmp
        n:-1
    Until n=0
    Return a
EndFunction
```

Die rekursive Funktion sieht einfacher (sie hat eine gewisse Ähnlichkeit mit der mathematischen Beschreibung der Fibonacci-Folge $a_n = a_{n-1} + a_{n-2},\ a_0 = 0,\ a_1 = 1$) und für den Laien eventuell auch effizienter aus, jedoch wird die iterative Funktion mit steigendem n nur linear ineffizienter, während die rekursive Variante exponentiell ineffizienter wird. Außerdem braucht letztere, pro Instanz von `fibo`, 20 Byte zusätzlichen Speicher (auf dem Stack[4]; man glaube mir oder lese selbst im Assemblercode nach).

Dieses Beispiel einer rekursiven Funktion gibt einen Wert zurück, indem sie sich selbst benutzt, um ihn zu errechnen. Eine Funktion muss aber nicht einen Wert zurückgeben, um als *rekursiv* zu gelten.

---

[2] Es gibt Programmiersprachen (wie Fortran), in denen Rekursion nicht zulässig ist.

[3] Iterative Programmierung verwendet, statt Selbstaufrufe, Schleifen.

[4] Der Stack ist der Ort, an dem Funktionsparameter, lokale Variablen und Rücksprungadressen gespeichert werden.

FLOODFILL

Floodfill ist ein Verfahren, das beispielsweise in der Bildbearbeitung als Eimer- oder Füll-Funktion bekannt ist. Genau dasselbe Prinzip kann aber auch in anderen Kontexten verwendet werden. Ein Spieleklassiker, bei dem das Verfahren gebraucht wird, ist Minesweeper. Wenn man ein blankes Feld öffnet, werden auch die umliegenden blanken Felder geöffnet. Hier einmal der relevante Teil des Codes (der Rest ist verlinkt):

```
Function oeffne_feld(spielfeld:TFeld[,], x:Int, y:Int)
    If spielfeld[x,y].offen Then Return
    If mine_benachbart(spielfeld, x, y)=0 And spielfeld[x,y].mine=0 Then
        oeffne_blankesfeld spielfeld, x, y
    Else
        spielfeld[x,y].offen=1
    EndIf
EndFunction

Function oeffne_blankesfeld(spielfeld:TFeld[,], x:Int, y:Int)
    If x<0 Or y<0 Or x>=FELD_B Or y>=FELD_H Or spielfeld[x,y].offen Then
Return
    spielfeld[x,y].offen=1
    If spielfeld[x,y].benachbart=0 Then
        oeffne_blankesfeld spielfeld, x-1, y
        oeffne_blankesfeld spielfeld, x+1, y
        oeffne_blankesfeld spielfeld, x, y-1
        oeffne_blankesfeld spielfeld, x, y+1
        oeffne_blankesfeld spielfeld, x-1, y-1
        oeffne_blankesfeld spielfeld, x-1, y+1
        oeffne_blankesfeld spielfeld, x+1, y-1
        oeffne_blankesfeld spielfeld, x+1, y+1
    EndIf
EndFunction
```

Anmerkung: Wenn der Spieler auf ein geschlossenes Feld klickt wird `oeffne_feld` mit den entsprechenden Parametern aufgerufen. `spielfeld[x,y].benachbart` gibt die Zahl der benachbarten Minen an.

Download minesweeper.bmx (docs.google.com)

Floodfill lässt sich auch iterativ implementieren (in Wirklichkeit kann man alles so oder so implementieren), ist aber in iterativer Variante auf einen Stack (Datenstruktur) angewiesen. Man muss immer genau abwägen, welches Verfahren effizienter ist, welches kürzer ist und, ob Effizienz oder Kürze wichtiger ist. Wenn es zum Beispiel um eine einmalige Laderoutine geht, dann darf sich diese auch etwas mehr Zeit nehmen, wenn man dafür eine ordentliche Quelltexteinsparung macht.

In diesem Fall ist die rekursive Variante schnell genug – in der Größenordnung, in der diese Funktion langsam wird, spielt niemand Minesweeper.

REKURSIVER ABSTIEG

Oft wird Rekursion auch (als sogenannter rekursiver Abstieg) im Compilerbau[5] verwendet, weil sich bestimmte Syntaxen[6] so besser parsen[7] lassen. In diesem Beispiel möchte ich einen kleinen Teil eines Interpreters[8] besprechen – einen vereinfachten Matheparser. Er soll Zahlen, die vier Grundrechnungsarten und Klammern beherrschen und das Endergebnis ausgeben.

[5] Compiler: wandelt Quelltext in Maschinencode um (eventuell über Umwege).
[6] "Grammatik" einer Programmiersprache.
[7] Verfahren, das ein Stück Quelltext nach den Regeln der Syntax zerlegt.
[8] Interpreter: interpretiert Quelltext, statt ihn in Maschinencode umzuwandeln.

Man könnte nun in EBNF[9] beschreiben, wie ein mathematischer Ausdruck aufgebaut ist, der genau den Vorgaben entspricht. Das sähe dann zum Beispiel so aus:

```
AUSDRUCK1   =   AUSDRUCK2 ( "+" | "-" ) AUSDRUCK2
AUSDRUCK2   =   WERT ( "*" | "/" ) WERT
WERT        =   [ "-" ] ZAHL | "(" AUSDRUCK1 ")"
```

Man könnte sich AUSDRUCK1, AUSDRUCK2 und WERT als Funktionen vorstellen, die auch rekursiv wirken. Bevor also Zahlen, wie in AUSDRUCK1 beschrieben, addiert oder subtrahiert werden können, muss durch AUSDRUCK2 überprüft werden, ob es da keine Punktrechnungen gibt, die zu beachten wären. Und bevor AUSDRUCK2 das machen kann, müssen Klammern beachtet werden. Zahlen haben dieselbe Priorität wie Klammern und sind daher auch in WERT zu finden. Zu beachten in WERT: das optionale Minus vor der Zahl, ohne dieses könnte man keine negativen Zahlen multiplizieren oder den Term mit einer negativen Zahl beginnen.

Ich habe einige Funktionen (Kommazahlen) und Sicherungen (Division durch 0, geschlossene Klammern) ausgelassen, weil es im Prinzip nur um die Rekursion geht, die dahinter steckt. Die Stellen, an denen etwas fehlt, sind durch Kommentare gekennzeichnet.

```
SuperStrict
Framework brl.blitz
Import brl.standardio
Import pub.stdc

' Globale Zeichenvariable - enthält das aktuell
' gelesene Zeichen.
Global look:Byte

getnextbyte
Print "= "+ausdruck1()
End

' AUSDRUCK1
Function ausdruck1:Int()
    Local a:Int
    a=ausdruck2() ' >>TERM1<< + TERM2
    While look=Asc "+" Or look=Asc "-"
        If look=Asc "+" Then
            getnextbyte
            a:+ausdruck2() ' a + >>TERM2<<
        ElseIf look=Asc "-" Then
            getnextbyte
            a=a-ausdruck2()' a - >>TERM2<<
        EndIf
    Wend
    Return a
EndFunction

' AUSDRUCK2
Function ausdruck2:Int()
    Local a:Int
    a=wert() ' >>ZAHL1<< * ZAHL2
    While look=Asc "*" Or look=Asc "/"
        If look=Asc "*" Then
            getnextbyte
            a:*wert() ' a * >>ZAHL2<<
        ElseIf look=Asc "/" Then
            getnextbyte
            a:/wert() ' fehlende Prüfung
        EndIf
    Wend
```

[9] Erweiterte Backus-Naur-Form: Schreibweise zum Festhalten einer Syntax.

```
	Return a
EndFunction

' WERT
Function wert:Int()
	Local x:Int, vorzeichen:Int=1
	If look=Asc "-" Then vorzeichen=-1 ; getnextbyte
	If look>=Asc "0" And look<=Asc "9" Then
		Return getnumber()*vorzeichen
	ElseIf look=Asc "(" Then
		getnextbyte
		x=ausdruck1()*vorzeichen ' ( >>AUSDRUCK1<< )
		getnextbyte ' fehlende Prüfung
		Return x
	EndIf
EndFunction

' Liest eine Zahl ein.
Function getnumber:Int()
	Local x:Int, negativ:Int
	While look>=Asc "0" And look<=Asc "9"
		x=10*x+look-Asc "0"
		getnextbyte
	Wend
	If negativ Then x=-x
	Return x
EndFunction

' Liest das nächste Zeichen ein.
Function getnextbyte()
	look=getchar_()
EndFunction
```

Download parser.bmx (docs.google.com)

ANALYSE

Hier noch eine kurze Analysephase zum obigen Code.

- AUSDRUCK1 wird aufgerufen.
- Die Funktion ruft AUSDRUCK2 auf, um die erste Zahl für eine mögliche Addition zu liefern. Das ist sinnvoll, weil AUSDRUCK2 nach möglichen Punktrechnungen sucht.
- AUSDRUCK2 ruft WERT auf. WERT gibt im einfachsten Fall eine Zahl zurück, die es ausgelesen hat. Damit wäre die Welt schon im Lot und es gäbe keine Rekursion in dem Beispiel, doch:
- WERT kann auch einen ganzen Klammerausdruck ausrechnen. Wie das? Sobald es eine Klammer liest, ruft es einfach AUSDRUCK1 auf. Die kann nämlich schon die 4 Grundrechenarten (zusammen mit AUSDRUCK2). Dass AUSDRUCK1 und AUSDRUCK2 zusammen noch keine Klammern beherrschen, stört nicht im Geringsten, denn WERT behandelt jede Klammer als Teilaufgabe und ruft, um sie zu lösen, AUSDRUCK1 auf.

NACHWORT

Es gibt noch einige andere Verwendungszwecke von Rekursion: Bäume und Graphen allgemein, Sortieren (z.B. Quicksort), Erzeugen von Fraktalen und natürlich Rekursion als Floodfill und rekursiver Abstieg. Außerdem gibt es Programmiersprachen, die auf Rekursion basieren: LISP und Haskell beispielsweise.

Ich hoffe, ich konnte mit diesem kleinen Artikel einen Denkanstoß liefern, wie dies und jenes zu lösen ist, und ein paar Neulingen diese Technik näherbringen.

Die Begriffe Rekursion/rekursiv und Iteration/iterativ sind auf die Informatik bezogen. Es sollte erwähnt werden, dass diese Begriffe in anderen Wissenschaften eventuell andere Bedeutungen haben.

Tipps zur Projektplanung

Von Xeres

Wie fängt man ein Spiel an? Besser noch: wie fängt man ein Spiel so an, dass man es auch beendet? Wie sieht ein „richtiges“ Konzept aus?

Ich hab nicht das Rezept für ein erfolgreiches Projekt, aber ich kann euch meins vorstellen.

Eine Idee muss her

Ganz am Anfang steht die Idee von einem Spiel. Wann ein kreativer Funke entsteht, mag für jeden unterschiedlich sein: Träume, Drogen[10], beim Bahn fahren aus dem Fenster starren – in jedem Fall solltet ihr Vorbereitet sein. Ich trage immer Stift und Papier mit mir herum – kein Einfall geht verloren.

Hier ist auch schon der erste Filter bei der Spieleentwicklung: Riesige Königreiche, famose Dungeons und knuffige Monster - solange es nur in eurem Kopf ist, ist es wunderbar. Der Akt des Aufschreibens reduziert die Idee auf ein Stück Papier.

Und plötzlich liest man, was man vorhat und sagt sich „Woa, das ist ‘ne dumme Idee!“.

Am besten ist es, wenn man seine Idee detailliert aufgeschrieben hat und nach ein paar Tagen nochmal sein Zettel zu Rate zieht. Es klingt immer noch machbar und spaßig? Dann folgt ein ausführlicheres Konzept.

Die Konzeption

Jedes Spiel – jedenfalls ist das bei mir so – beginnt mit einer wunderbaren Kernidee: „Es ist ein Tower-Defense, aber man spielt im Angriff!“, „Der Name des Spiels, ist die Lösung für das erste Rätsel!“, „Hey, die Idee sieht gut aus, ich glaub‘, die klaue ich!“.

Um diese Kernidee herum muss man ein Spiel entwickeln. Wie sieht die Karte/Umgebung aus? Welche Einheiten/Gegner muss es mindestens geben? Wie steuert man das Ganze?

Alles wird zu Papier gebracht.

In einem kleinen Quadrat wird der Bildschirm skizziert. Wie groß muss das Fenster sein, um genug Platz für Spieler, Objekte und UI zu lassen?

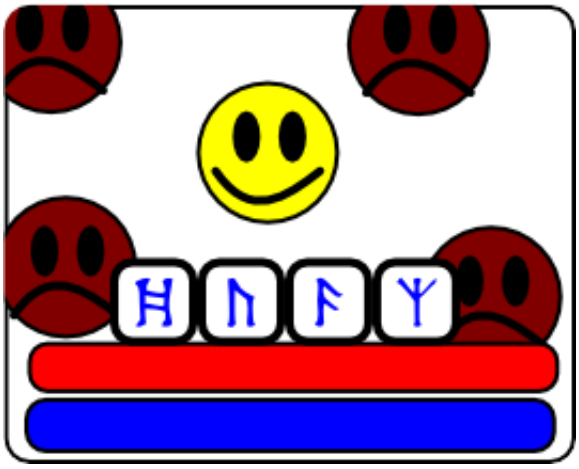

Abbildung 19: Die super designten Charaktere & UI sind gut zu erkennen!

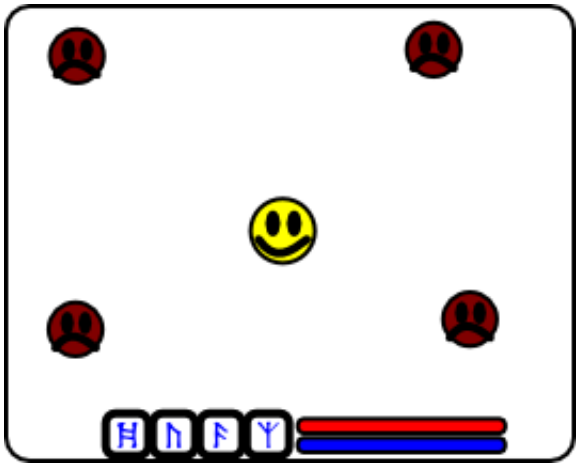

Abbildung 20: Der Spieler hat tatsächlich einen Überblick über seine Situation. Zeigt dem Spieler was er sehen muss, nicht, was ihr für tolle Grafik habt.

Dann folgen ggf. ein paar Kalkulationen der Spielmechanik: Wenn der Schaden so in die Formel eingeht, wie lange dauert es, das Monster um zu hauen? Wie sieht eigentlich die Formel zur Beschleunigung des Rennwagens aus? Wenn die fliegenden Schweine mit 66% Wahrscheinlichkeit 2x Ohren droppen, wie lange muss man durchschnittlich

[10] Drugs are bad, mkay?

für einen Ohrensuppenquest farmen gehen?

Das Ziel besteht darin, wichtige Teile auszutesten – auch mit einem schnell zusammengewürfelten Code. Wenn ihr die Fähigkeiten habt, die ihr braucht, die Funktion schnell genug ist oder andere kritische Teile passen, folgt nun endlich…

Die Alphaphase

Ihr habt mit Tinte, Tusche und Wachsmalstiften vollgeschriebene und bemalte Zettel. Im Projektordner liegen ein paar Bilder, um den passenden Stil zu finden. Ein paar Codes demonstrieren dies und jenes.

Schritt 1: Schaufelt das alles in einen Unterordner „Entwicklung“ oder „Design“ oder was auch immer.

Schritt 2: Einen Unterordner für die Medien, einer für Includes und ein File für den Code erstellen.

Schritt 3: Anfangen zu programmieren!

Oft zitierter Rat: „Fang nicht mit dem Menü an!“ Nun – wenn man die ganze letzte Seite ignoriert hat, klar – sobald das Menü fertig ist, geht einem auf: die Idee ist zu schwierig umzusetzen / Mist / Ich habe eine noch viel bessere Idee!

Wenn der Plan steht, macht eine Strukturierung des Codes schon Sinn. Ein fertiges Menü kann auch ein kleines Erfolgserlebnis sein oder – wenn ihr erfahrener seid – einfach in 5 Minuten stehen, ohne das ihr viel Zeit gebraucht hättet. Ich jedenfalls fange für einen BCC seit einiger Zeit immer mit dem Menü an.

Programmieren ist auch ein Handwerk: Mit jedem Neuanfang bekommt ihr mehr Übung, bis gewisse Konstruktionen eins fix drei fertig sind (und ihr auf eure Funktionssammlung zurück greift).

Macht dabei aber nicht den Fehler, eure eigene Engine zu programmieren, wenn ihr nicht unbedingt müsst. Userlibs oder Module gibt es in Fülle. So gern man auch 100%ige Kontrolle hätte: Das Spiel beginnt ihr damit nicht! Wenn ihr jede Zeile eigenhändig in Assembler geschrieben habt, weil ihr meint, nur so „richtig“ programmiert zu haben, werden andere ein paar Dutzend Spiele mit einem GameMaker beendet haben. Schreibt Spiele, keine Engines!

Wichtig sind zwei Dinge: Spaß & sichtbare Ergebnisse. Benutzt eine Testkarte/Umgebung, um alle Spielelemente nach der Implementation sofort zu testen.

Die Betaphase

Die Spielmechanik steht (größtenteils)? Dann wird es langsam Zeit, etwas Spielbares daraus zu machen. Das Wichtigste hierbei ist das Verlieren. Schickt den Spieler nur mit dem absoluten Minimum an Leben / Gold / Fähigkeiten etc. auf die Reise und lasst ihn eher schneller als langsamer ins Gras beißen. Der Grund ist simpel: Einfacher geht immer aber ein Spiel ohne Herausforderungen ist ein langweiliges Spiel.

Wenn man verlieren kann, macht eine Speicher/Lade Routine, Passwordsystem oder andere Einrichtung Sinn. Da das Spiel Grundsätzlich steht, sollte da in Zukunft nicht zu viel geändert werden müssen.

Wenn ihr ein paar Karten / Rätsel / Quests eingebaut habt, ist es Zeit zum Testen!

Passende Testsubjekte sind nicht zu schwer zu finden, aber achtet auf ein paar Kleinigkeiten:

Mütter / Väter sind nicht unbedingt eine gute Wahl, wenn sie bedingungslos toll finden, was ihr produziert – ihr braucht sachliche Kritik, am besten von der Zielgruppe.

Für die Tutoriallevel von WhiteWorld habe ich mir einen Freund/Freundin geschnappt, sie ohne weitere Hinweise vor den Rechner

gesetzt, und mit Zettel & Stift bewaffnet ihre Reaktionen dokumentiert. Die bitte, laut zu denken kann dabei auch hilfreich sein.

Wenn ihr nun Verbesserungswünsche, Bugs usw. gesammelt habt, tweaked ihr das Spiel entsprechend.

Ihr müsst an diesem Punkt stark sein und diesen Gedanken unterdrücken: „Ich habe so viel gelernt, ich könnte den Code nochmal komplett neu und besser schreiben!“

Nein! Ihr werdet das nächste Mal wieder an dieser Stelle ankommen. Und das Mal darauf. Euer Code wird nie perfekt sein. Hackt einen Workaround zusammen wenn es sein muss, aber bringt das Spiel jetzt zu Ende.

Nun wird es Zeit für das Foolproofing: Selbst der dümmste anzunehmende User darf euer Spiel nicht zum Absturz bringen. Fehleingaben müssen erkannt und korrigiert oder entschärft werden. Mein Vater hat immer bewusst unsinnige Dinge ausprobiert und mich damit echt genervt – aber der Punkt ist: Leute werden das probieren. ReadMe und Handbücher werden nicht gelesen, die Anweisung „in dieses Feld nur Zahlen eingeben!“ wird missachtet und Dateiauswahldialoge werden abgebrochen. Versucht alles falsch zu machen, was nur irgendwie geht und sichert das Programm ab. Vermutlich finden die Tester noch etwas, was ihr übersehen habt.

Die Releaseversion

Alle Aspekte des Spiels wurden kontrolliert, alles läuft fehlerfrei und problemlos.

Jetzt kommt der schwierige Teil, die letzten 5%. Das Erstellen des (restlichen) Inhalts.

Achtet darauf, dem Spieler die Spielmechanik vorzuführen, so dass er sie versteht und anwenden kann. Konzentriert euch auf ein neues Element, bevor ihr es in Verbindung mit dem Rest benutzt. Sobald ihr nichts Neues habt, liegt es an euch den Endpunkt zu bestimmen. Rätsellastige Spiele können enden, sobald das neuste Element vom Spieler bezwungen wurde, in missions- und storygetriebenen Spielen geht es dann meist erst richtig los.

Abschließend solltet ihr ein Handbuch, ReadMe- oder Infofile verfassen, das nochmal die Steuerung, Geschichte oder zusätzliche Informationen enthält, ob das Spiel weitergegeben werden darf und natürlich, wie ihr zu erreichen seid.

Nehmt euch auch Zeit für Screenshots und die Texte, mit denen ihr euer Spiel vorstellt (und bewerbt) – die Qualität eines Showcase-Eintrags spiegelt die des Spiels wieder.

Tipps & Tricks

Vom BPS Team

Blitz3D

```
; Mit dem Befehl AppTitle kann die Titelleiste eines Graphics-Fensters
beeinflusst werden.

AppTitle "Programmname", "Wirklich Beenden?"

; Optional und nicht empfohlen, trotzdem einer Erwähnung wert: bei BB und B3D
kann auch noch eine Sicherheitsabfrage ergänzt werden, die angezeigt wird,
sobald das Programm über das X geschlossen werden soll.
```

```
; In BlitzBasic /Blitz3D kann man ein Programm im Vollbildmodus automatisch in
der Desktop-Auflösung erstellen.
; Wenn man die Parameter für Breite und Höhe bei Graphics auf 0 stellt,
Graphics(0, 0, 0, 1)
; dann startet das Programm im Vollbild mit Desktop-Auflösung
Print("Das Fenster ist " + GraphicsWidth() + "px breit")
Print("und " + GraphicsHeight() + "px hoch.")

WaitKey()
```

```
; Man kann mit % und $ Binär- resp. Hex-Werte definieren
Local acht = %1000
Local mio = $f4240
Print(acht)
Print(mio)
```

```
; Mit logischen Bedingungen lässt sich auch rechnen.
Local a=42
Print(3*(a=42))
; a=42 wird hier 1.
```

```
; Benutze Konstanten um den Code lesbarer und flexibler zu machen.
Const ENDECODE = 112 ; ASCII-Code von p
; Aufforderung ausgeben
Print("Drücke " + Chr(ENDECODE) + " um das Programm zu beenden")
; Warten bis p gedrückt wurde
Repeat
Until WaitKey() = ENDECODE
```

```
; In RuntimeError und Notify-Meldungen kann man mittels Chr(10) Zeilenumbrüche erstellen
RuntimeError("Dieser" +Chr(10)+ "Text" +Chr(10)+ "ist" +Chr(10)+ "hoch")
```

```
; Wenn etwas Mal nicht so klappt wie es sein sollte oder auch einfach nur zur Kontrolle eignet sich der 'Debug'-Modus (einstellbar in der IDE) um Fehler zu finden.
; Funktioniert wie Print, leitet die Ausgabe jedoch in die Konsole und nur bei aktiviertem Debugmodus
DebugLog("Fehlertext " + Fehlervariable)
; Manchmal möchte man ein Programm auch gezielt untersuchen. Dafür eignet sich
Stop
; Das Programm wird im Debugmodus angehalten und kann Schritt für Schritt fortgesetzt werden (durch die Symbole in der IDE).
```

```
; Color kann man auch eine Hex-Farbangabe übergeben, wenn die ersten beiden Parameter 0 sind.
Color(0, 0, $99CCFF)
Text(0, 0, "Hallo")
```

```
; Formatiere Strukturen mit Tabulatoren um den Code übersichtlich zu halten.
For y = 0 To 15
    For x = 0 To 15
        ; wenn das Produkt x*y > 45 ist,
        ; wird ein Pixel gezeichnet
        If x*y > 45 Then
            Plot x, y
        EndIf
    Next
Next
```

```
; Oft können IF-Konstrukte auch mathematisch umgesetzt und somit vereinfacht werden.
; z.B. ist
x = x + KeyDown(205) - KeyDown(203)
y = y + KeyDown(208) - KeyDown(200)
; dasselbe wie
If KeyDown(205) Then x = x +1
If KeyDown(203) Then x = x -1
If KeyDown(208) Then y = y +1
If KeyDown(200) Then y = y -1
```

```
; Benutze Include, um Teile deines Projektes (Funktionen, Deklarationen, Routinen...) in andere Dateien auszulagern.
Include "source\Consts.bb"
Include "source\Functions.bb"
; Das kann auch hilfreich sein, um Projekt-unabhängige Funktionen einzubinden
Include "D:\BlitzBasic\Mathefunktionen.bb"
```

```
; DIM-Arrays können auch einen selbst geschrieben Type als Datentyp haben.
Type TStein
     Field farbe%
End Type
Dim map.TStein(8,8)
; Damit kannst du in Tilemap-basierten Spielen die Objekte eleganter
durchiterieren, als mit der Type-Liste.
Local x%,y%
For y%=0 To 7
     For x%=0 To 7
          If(map(x,y) <> Null) Then
               ; hier verschieden Operationen ausführen
               map(x,y)\farbe = Rand(1,4)
          EndIf
     Next
Next
```

```
; Benutze beim Laden deiner Medien-Dateien (Bilder, Sounds usw.) immer relative
Pfade:
Global imgFigur = LoadImage("Grafiken/Typ.png")
Global sndSchuss = LoadSound("Sounds/Schuss.wav")
; So erlaubst du den Anwendern deiner Programme, diese dort zu speichern, wo
sie wollen. Also ganz egal, ob in C:\Programme oder F:\Games, das, was zählt,
ist dann nur noch die Ordner-Struktur. Aber die liefert man ja idR. im .zip-
File mit.
; Relative Pfade beziehen sich immer auf den Pfad, in dem die .exe-Datei, oder
auch die .bb-Datei an der du arbeitest, liegt. Beim Beispiel oben wäre also die
Hauptdatei in einem Verzeichnis (sagen wir mal D:\...\MeinSpiel) gespeichert
und die Grafiken resp. Sounds in den gleichnamigen Unterordnern.
```

```
; Explizites casten (das Umwandeln von einem in einen anderen Variablentyp)
eines Types zu String ergibt den Inhalt aller Felder in eckigen Klammern in der
Reihenfolge ihrer Definition:
Type TTest
     Field n%, buchstabe$, f#
End Type

Local t.TTest= New TTest
t\n = 42
t\buchstabe = "Q"
t\f = Pi
Print(Str(t))
WaitKey()
```

```
SuperStrict 'Zwingt den Programmierer "sauber" zu arbeiten und ist deshalb absolut zu empfehlen.

'Deklarationspflicht für Variablen
'Typen müssen streng definiert werden (mit :Int, etc.)
'Es wird mit Objekten statt Integer Handles gearbeitet, was den Ablauf beschleunigt.
'Goto ist deaktivert
'eigene Gültigkeitsräume auch für Schleifen

'SuperStrict hilft also später kaum auffindbare Fehler, wie falsche Typenbezeichnung etc. von vorne herein zu vermeiden.

'In abgeschwächter Form kann auch Strict eingesetzt werden, das erzwingt lediglich eine Variablendeklaration. Daher ist es nur bedingt zu empfehlen.

'Strict oder Superstrict werden als erster Befehl geschrieben.
```

```
Include "Dir/File"
Import "Dir/File"
'Die Anwendung dieser beiden Befehle ist ähnlich, die Unterschiede bedeutend.
'Include bietet die Möglichkeit Code in verschiedene Dateien aufzuteilen.
Include fügt den Code an exakt der Stelle ein, an der es aufgerufen wird.
'Import dagegen kompiliert die Auslagerungen vor, womit sie nur bei Änderungen erneut kompiliert werden müssen. Import eignet sich daher dafür um Klassen auszulagern, oder sogar um ganze Module verfügbar zu machen.
```

```
'Der folgende Befehl funktioniert wie Print, leitet die Ausgabe auch in die Konsole allerdings nur bei aktiviertem Debugmodus
DebugLog("Fehlertext" + Fehlervariable)

'Manchmal möchte man ein Programm auch gezielt untersuchen. Dafür eignet sich
DebugStop
'Das Programm wird im Debugmodus angehalten und kann Schritt für Schritt fortgesetzt werden.
```

```
' In For-Schleifen kann es manchmal ganz nützlich sein,
' Until statt To zu verwenden, wenn das letzte Element
' nicht mitgezählt werden soll.
For Local i:Int = 0 Until 10
     Print i
Next
```

```
'BlitzMax kennt gekürzte Schreibweisen für bestimmte Operationen. Diese werden
bei der Variable angewendet ohne den Wert nochmal explizit zuzuweisen (wie bei
"a = a + 2").
a:+ 2     ' Addition
a:- 2     ' Subtraktion
a:* 2     ' Multiplikation
a:/ 2     ' Division
a:Shl 2   ' Bitweise Linksverschiebung
a:Shr 1   ' Bitweise Rechtsverschiebung
a:Sar 1   ' Arithmetische Rechtsverschiebung (leere Bits = Kopie des Originals)
a:Mod 2   ' Modulo
a:& 3     ' Bitweises Und
a:| 3     ' Bitweises Oder
a:~ 7     ' Bitweises Exklusiv-Oder
```

```
'Die Länge von Arrays ist im Feld .length gespeichert.
Local array:Int[] = New Int[100]
Print array.length
```

```
'Mit Slices lassen sich ganz einfach ganze Arrays kopieren.
' kopiert den Inhalt von altes_array in neues_array
neues_array = altes_array[..]
```

```
'Ähnlich einfach kann man Arrays in BlitzMax um Elemente erweitern. Diese
werden an das Ende des Felds rangepusht.
Local meinArray:Int[] = [5, 10, 15]
meinArray :+ [20] ' fügt ein Element dem Array hinzu
meinArray :+ [25, 30, 35, 40] 'analog für mehrere Elemente auf einmal
```

```
'In BlitzMax können globale Variablen in Funktionen ausgelagert werden, um den
Code sauber zu halten.
Print(Count())
Print(Count())
Print(Count())

Function Count:Int()
     ' n wird wie eine globale Variable behandelt, ist aber nur
     ' in der Funktion Count sichtbar
     Global n:Int = 0
     n :+ 1
     Return n
End Function
```

```
'Der Compiler optimiert Bedingungen, was sich ausnutzen lässt. Wenn myType Null
ist, würde folgende Abfrage zu einer Ausnahme führen:
If myType.getValue() > 2 Then
'Diese kann einfach abgefangen werden:
If myType:Double = Null And myType.getValue() > 2 Then
'Hintergrund: Eine AND-Abfrage kann nur wahr zurückliefern, wenn alle
Bestandteile wahr sind. Folglich kann die Bedingung nicht mehr wahr werden, was
der Compiler erkennt und deswegen die zweite Bedingungsprüfung nicht durchführt
(und damit getValue() nicht auf NULL ausführt).
'Analog dazu werden ODER-Abfragen übersprungen, wenn der erste Teil bereits
wahr ist.
```

```
'Manchmal verdeckt der lokale Scope Variablen in einem äußeren Scope. Wenn man
die Variablen im äußeren Scope ansprechen will, reicht ein "." vor dem
Variablennamen.
Global id:Int = 10

Type tPlayer
     Field id:Int

     Method AendereGlobaleId(new_id:Int)
          ' ändert die globale Variable id:
          .id = new_id
     End Method
End Type

Local player:tPlayer = New tPlayer

Print(id)
player.AendereGlobaleId(7)
Print(id)
```

CREDITS

	Autoren & Beteiligte	Quellen & weiterführende Infos
Cover	Tobchen	
Vorwort	Joshmami	
Nicknames – Wie es dazu kam	Abrexxes BladeRunner bruZard Count-Doku DarkCorner hamZta Mr.Hyde Joshmami Klepto2 Lobby Noobody Ohaz Propellator Pulverfass Tagirijus Xeres	*IRC: blitzforum.de*
Interview mit Noobody	Joshmami Noobody	*The Stanford Bunny*
Interview mit Midimaster	Joshmami Midimaster	*Tatiti*
Beginner- Interview	BaseTH Hagbard	
Hamburg Treffen	Mr.Hyde	
Was sich im Showcase verbirgt	BaseTH	*Argon* von Sirius7 *BiaL* von DaysShadow *BIKILA* von Lucius *Bob the Blob* von Pousup *Pac The Man* von coolo *SimpleSnake* von MasterSolaris *SN4KE* von Cornelius
Die Magie von Git und GitHub	Propellator	*git-scm.com* *github.com* *Set Up Git*

Die Macht der Rekursion	Thunder	*WP: Rekursion* *Rekursion in der Informatik* *Minesweeper.bmx* *Parser.bmx*
Tipps zur Projektplanung	Xeres	*Complexity vs. Depth* *Finishing a Game* *Subtractive design*
Tipps & Tricks	D2006 Holzchopf maximilian Mr.Hyde	Zusammengestellt zur Einführung der *Beginner's Practice Series*. Die Formatierung des Codes erfolgte mit *Highlight*.
Korrekturleser	Holzchopf Tennisball	
Layout & zusätzliche Grafiken	Xeres	
Projektleitung	Joshmami	