BLAISE PASCAL MAGAZINE 112

Databases / CSS Styles / Progressive Web Apps
Android / IOS / Mac / Windows & Linux

Blaise Pascal

Bericht Kanonenkugel-Simulation
Die Castle Game Engine, die schlechte Art, Schach zu spielen
3d-Physik-Spaß mit der Castle Game Engine
H-BOT, H-förmiger Roboter: ein simulierter Roboter
Bauen und benutzen Sie Ihren eigenen Roboter
Pythagoreische Dreiergruppen
Debugging in FPC-Lazarus Teil 3
Ausführen von Programmen auf dem Server in PAS2JS
langlaufende Prozesse auf dem Server

BLAISE PASCAL MAGAZINE 112
Multi platform /Object Pascal / Internet / JavaScript / Web Assembly / Pas2Js / Databases / CSS Styles / Progressive Web App
Android / IOS / Mac / Windows & Linux


Blaise Pascal

CONTENT

ARTICLES

ADVERTISING


Niklaus Wirth

Pascal ist eine imperative und prozedurale Programmiersprache, die Niklaus Wirth (links unten) in den Jahren 1968-69 entworfen und 1970 veröffentlicht hat. Es handelt sich dabei um eine kleine, effiziente Sprache, die gute Programmierpraktiken unter Verwendung von strukturierter Programmierung und Datenstrukturierung fördern soll. Ein Derivat, bekannt als Object Pascal, wurde 1985 für die objektorientierte Programmierung entwickelt. Der Name der Sprache wurde zu Ehren des Mathematikers und Erfinders der ersten Rechenmaschine gewählt: Blaise Pascal (siehe oben rechts).

Herausgeber: © Foundation Supporting Programming Language Pascal - mit Sitz in den Niederlanden
Registrieter Name: Stichting Ondersteuning Programmeertaal Pascal IJsselstein, Netherlands
VAT / BTW: NL814254147B01 Chamber of commerce (KVK) 30 202429 Handy: +31 6 21 23 62 68

CONTRIBUTORS

Stephen Ball
http://delphiaball.co.uk
DelphiABall

Dmitry Boyarintsev
dmitry.living @ gmail.com

Michaël Van Canneyt
,michael @ freepascal.org

Marco Cantù
www.marcocantu.com
marco.cantu @ gmail.com

David Dirkse
www.davdata.nl
mail: David @ davdata.nl

Benno Evers
b.evers @
everscustomtechnology.nl

Bruno Fierens
www.tmssoftware.com
bruno.fierens @ tmssoftware.com

Holger Flick
holger @ flixments.com

Mattias Gärtnernc-
gaertnma@netcologne.de

Max Kleiner
www.softwareschule.ch
max @ kleiner.com

John Kuiper
john_kuiper @ kpnmail.nl

Wagner R. Landgraf
wagner @ tmssoftware.com

Vsevolod Leonov
vsevolod.leonov@mail.ru

Andrea Magni
www.andreamagni.eu andrea.
magni @ gmail.com
www.andreamagni.eu/wp

Paul Nauta PLM Solution
Architect CyberNautics
paul.nauta @ cybernautics.nl

Kim Madsen
www.component4developers.com
kbmMW

Boian Mitov
mitov @ mitov.com

Jeremy North
jeremy.north @ gmail.com

Detlef Overbeek
- Editor in Chief
www.blaisepascal.eu
editor @ blaisepascal.eu

Anton Vogelaar
ajv @ vogelaar-electronics.com

Danny Wind
dwind @ delphicompany.nl

Jos Wegman
Corrector / Analyst

Siegfried Zuhr
siegfried @ zuhr.nl

Chefredakteur
Detlef D. Overbeek, Niederlande Tel.: Mobil: +31 (0)6 21.23.62.68
Nachrichten und Pressemitteilungen nur per E-Mail an editor@blaisepascal.eu

Abonnemente können online unter https://www.blaisepascalmagazine.eu/deutsche-Ausgabe/ oder per schriftlicher Bestellung abgeschlossen werden oder indem Sie eine E-Mail an office@blaisepascal.eu senden. Das Abonnement kann zu einem beliebigen Zeitpunkt beginnen. Alle Ausgaben, die im Kalenderjahr des Abonnements veröffentlicht werden, werden geschickt. Das Abonnement hat eine Laufzeit von 365 Tagen. Abonnemente werden nicht ohne Vorankündigung verlängert
Der Zahlungseingang wird per E-Mail verschickt. Sie können das Abonnement bezahlen, indem Sie die Zahlung an folgende Adresse senden: ABN AMRO Bank Konto Nr. 44 19 60 863 oder per Kreditkarte oder Paypal Name: Stiftung Pro Pascal (Stichting Programeertaal Pascal)
IBAN: NL82 ABNA 0441960863 BIC ABNANL2A Umsatzsteuer-Nr.: 81 42 54 147 (Stichting Programmeertaal Pascal)
Abonnementsabteilung Edelstenenbaan 21 / 3402 XA IJsselstein, Niederlande Mobil: + 31 (0) 6 21.23.62.68 office@blaisepascal.eu

Mitglied der **Königlich Niederländischen Bibliothek** KB

Mitglied und Spender von WIKIPEDIA

Subscriptions (2022 prices)	Internat. excl. VAT	Internat. incl. 9% VAT	Shipment	TOTAL
Printed Issue (8 per year) **±60 pages :**	**€ 200**	**€ 218**	**€ 130**	**€ 348**
Electronic Download Issue (8 per year) **±60 pages :**	**€ 64,20**	**€ 70**		

von Ihrem Redakteur

Hallo!
In dieser Ausgabe haben wir etwas ganz Besonderes zum Lesen und Benutzen:
Die **Castle Game Engine**.
Michalis Kamburelis ist der Entwickler dieser Engine und sie funktioniert sowohl unter Lazarus als auch unter Delphi. Mit dieser Engine kann man nicht nur schöne Spiele, sondern auch 3d-Objekte erstellen. Das ist eine sehr schöne Art der Entwicklung.

Wir sind damit beschäftigt, dieses in **WebAssembly** und **PAS2JS** zu integrieren, so dass wir es schließlich sogar im Web verwenden können.
Ich habe bereits angefangen, über eine normale Desktop-Anwendung nachzudenken, die einige Elemente der Game Engine integriert. Besonders für die Unterstützung des Benutzers (User-Interface) könnte das sehr interessant werden.

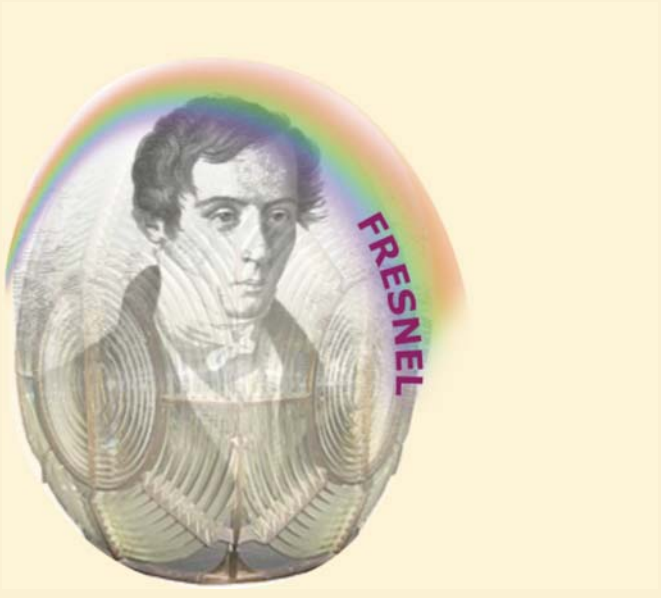
Im Moment sind wir sehr tief in der Entwicklung von **Fresnel** bezogen (*die Alternative für die LCL von Lazarus*) **Michael van Canneyt** und **Mattias Gärtner** arbeiten sehr hart daran.
Wir versuchen, das bis zu meinem Besuch in **Backnang** -Stuttgart / Heidelberg- am 22. und 24. September dieses Jahres einen ersten Anfang fertig zu stellen.
(**`https://www.blaisepascalmagazine.eu/lazarus-konferenz-2023-in-backnang-22-09-2023-24-09-2023/`**)
Mattias versucht, etwas für Skia zu bauen und Michael wird das die Mouse-events für die Bibliothek erstellen.
Das ist schon ein enormer Schritt, um **Lazarus** Farben fähig zu machen.
Schließlich wollen wir etwas machen, das keinen Multi-OS-Compiler zur Verfügung hat:
Farbeinstellung nach Belieben - auf welcher Plattform auch immer,
unabhängig vom Farbschema des zu verwendenden Betriebssystems.

Martin Friebe führt den stark verbesserten **Debugger** für **Lazarus** weiter.
Bitte versuchen Sie es, es ist sehr interessant und hilfreich. Es werden noch etwa sechs oder sieben weitere Artikel folgen, so daß es wie ein Kurs im Debuggen ist.

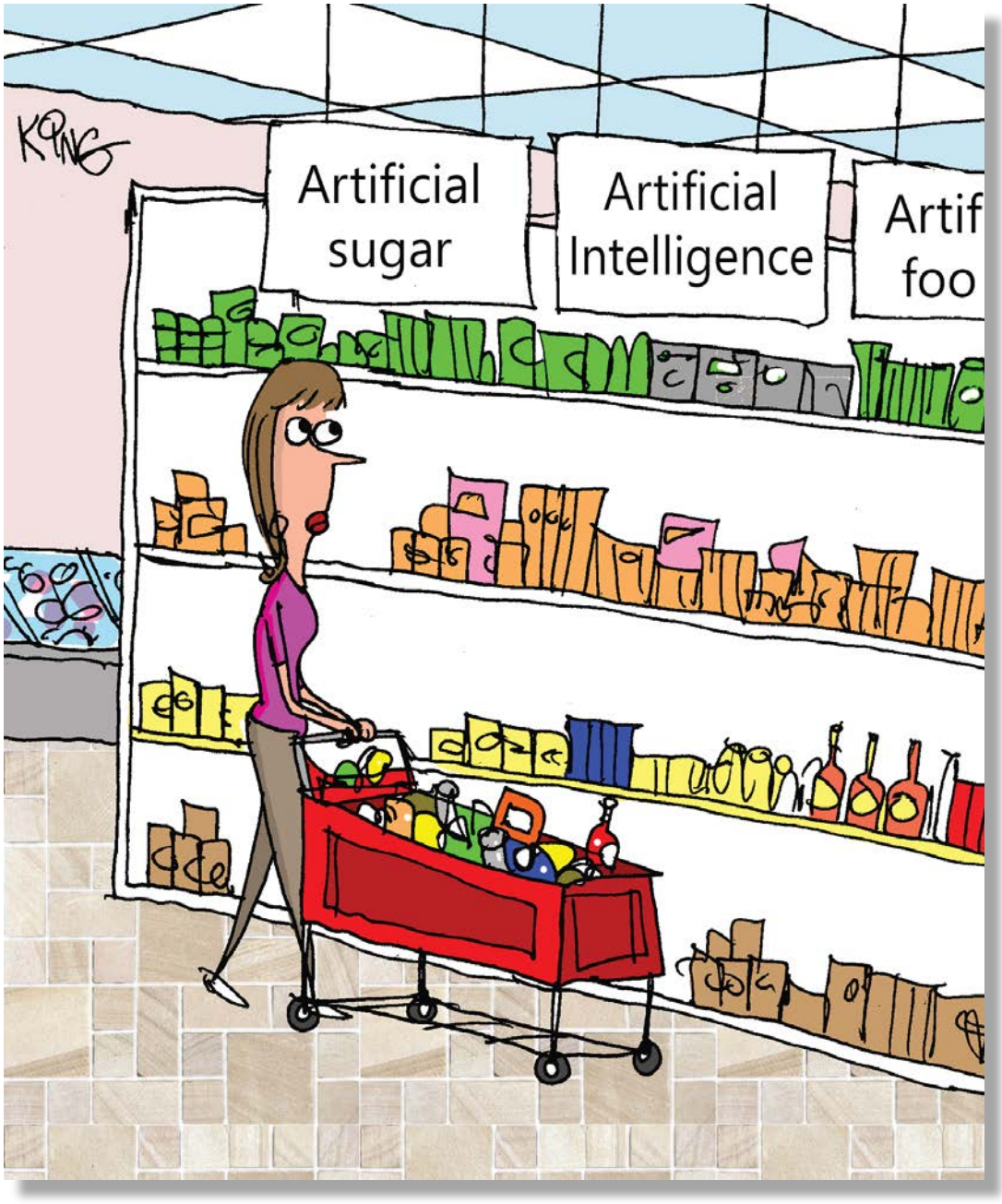
Diese neuen Entwicklungen werden auch in das nächste **Lazarus-Handbuch** integriert werden.
Apropos Bücher: Wir werden zwei neue Bücher veröffentlichen:

1. **Learning to program with FreePascal and Lazarus** (Englisch) - es ist ein völlig neu geschriebenes Buch mit vielen einfachen Lektionen, so dass Sie in der Lage sein werden, jedes Programm zu erstellen, das Sie wollen.
2. **Creating Pas2Js Apps, in Pascal.** (Englisch)

Wir werden über Details in der nächsten Ausgabe des Blaise Pascal Magazins schreiben.



Von unserem technischen Ratgeber Jerry King



POCKET PACKAGE (2BOOKS)

EXCLUDING VAT AND SHIPPING

LAZARUS HANDBOOK PRICE: € 25,00

https://www.blaisepascalmagazine.eu/product-category/books/

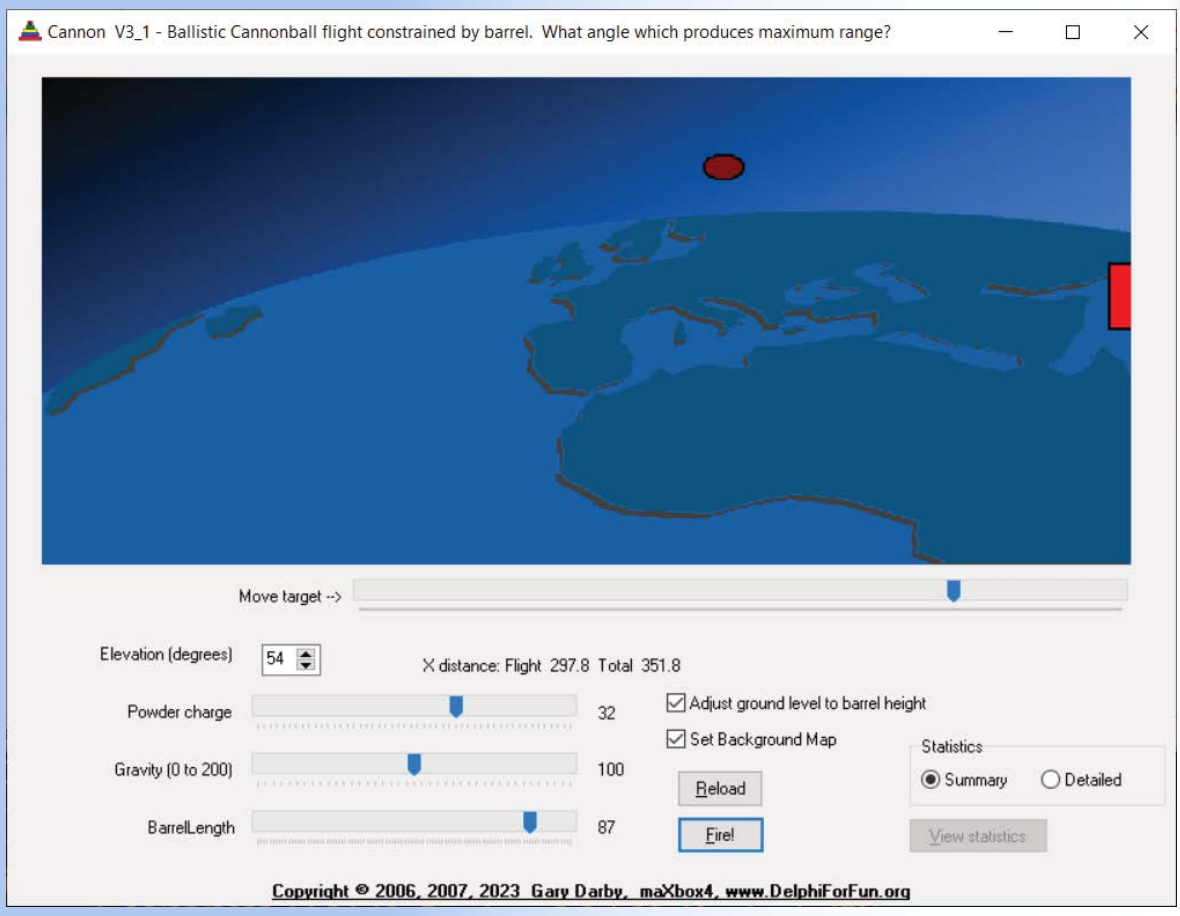
Bericht Kanonenkugel-Simulation

maXbox Starter 112

Starter Expert

Heute machen wir einen Abstecher in die Welt der Ballistik, Simulation & Training.

Einer meiner Freunde bei **Delphi for Fun** hat mich auf die Idee gebracht, das Executable in ein Skript für Windows 11 zu portieren, als Vorbereitung für die 64bitbox.
Ballistik ist also die Lehre von der Bewegung von Projektilen wie Kugeln, Granaten und Raketen, in unserem Skript beschäftigen wir uns nur mit Kugeln. Sie ist ein Teilgebiet der Mechanik, das sich mit dem Verhalten von Objekten in Bewegung beschäftigt. Die Ballistik kann in drei Hauptkategorien unterteilt werden: Innenballistik, Außenballistik und Endballistik. Also habe ich das Delphi-Programm mit einigen Verbesserungen in dieses Skript übersetzt:

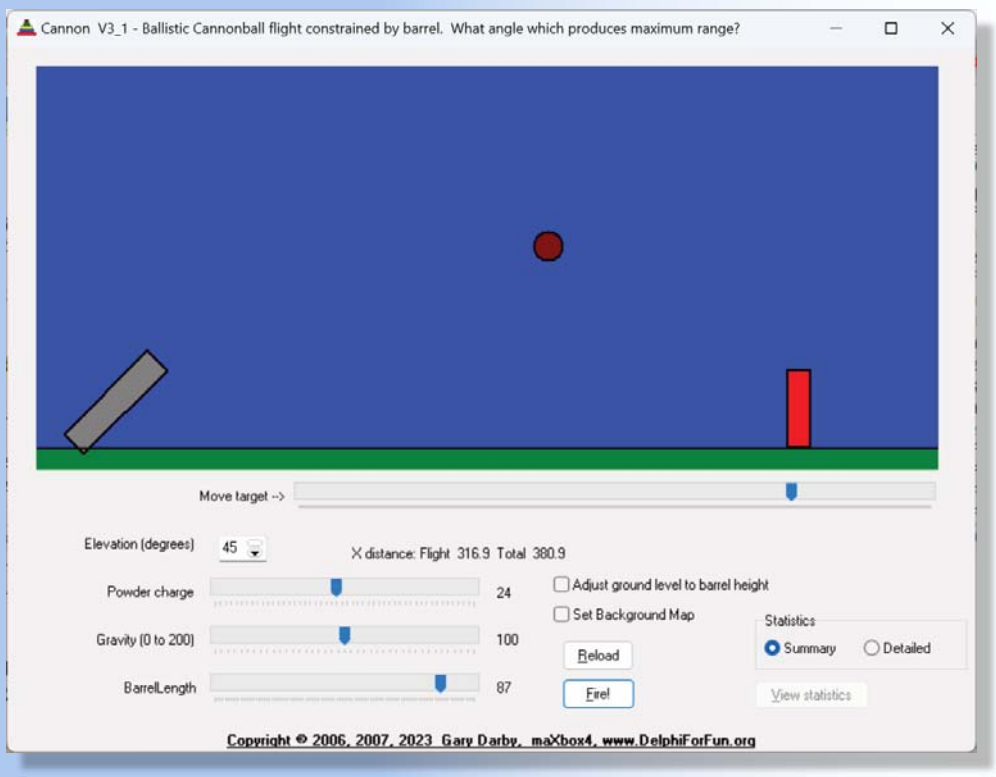


Natürlich können wir eine Kanonenkugel mit einer Physiksimulationssoftware simulieren oder in unserem Fall dieses Modell in Pascal integrieren. Mit dieser Simulation können Sie eine Kugel aus einer Kanone abschießen und sich selbst herausfordern, ein bewegliches Ziel zu treffen. Sie können Parameter wie den Winkel (*Elevation*), die Anfangsgeschwindigkeit (*Pulverladung*) und die Masse (*Schwerkraft*) einstellen und die Vektordarstellungen untersuchen.

Auch eine erklärbare Statistik ist Teil des Skripts,
als Zusammenfassung oder ausführlich (*unser Standardfall, der das Ziel traf*):

Zusammenfassung der Fallstudie

Lauflänge 87, Winkel 45,0, Anfangs-V 24,0, Schwerkraft 1,0
Zeit im Lauf 3,8 Sekunden
X-Abstand am Ende des Laufs 61,5
Y-Abstand am Ende des Laufs 61,5
Zeit bis zur Spitze des Freiflugbogens 15,1, insgesamt 18,9
X-Entfernung bis zur Spitze des Freiflugbogens 226,5, 288,1 insgesamt
Höhe über dem Lauf bis zum oberen Ende des Freiflugbogens 113,3, 174,8 insgesamt
Zeit bis zum Erreichen des Bodens aus maximaler Höhe 18,7, 37,6
total X Entfernung vom oberen Ende des Freiflugbogens bis zum Ende 281,4, 569,5 gesamt



Das Interessante an dieser Simulation ist, dass sie zeigt, dass die Bewegung eines Projektils wie einer Kanonenkugel im Grunde dasselbe ist wie die Umlaufbahn eines Himmelskörpers wie des Mondes!
Die entwickelten Rotations- und Translationsroutinen werden hier verwendet, um die Kanone zu heben.
Die Bewegungsschleife des Balls ähnelt einem Bouncing-Ball-Programm mit dem Zusatz einer horizontalen Komponente.
Die Anfangsgeschwindigkeiten in X- und Y-Richtung sind proportional zum Kosinus bzw. Sinus des Elevationswinkels beziehungsweise.
Der Lauf ist etwas knifflig; wir gehen davon aus, dass die Kanonenkugel im Inneren des Laufs "eine Rampe hinaufrollt", wobei die Komponente der Schwerkraft, die parallel zum Rohr wirkt, ist die Kraft, die die Geschwindigkeit der Kanonenkugel sowohl in x- als auch in y-Richtung verringert, so dass wir die Abstandsfunktion im Auge behalten:

http://www.softwareschule.ch/examples/cannonball.txt

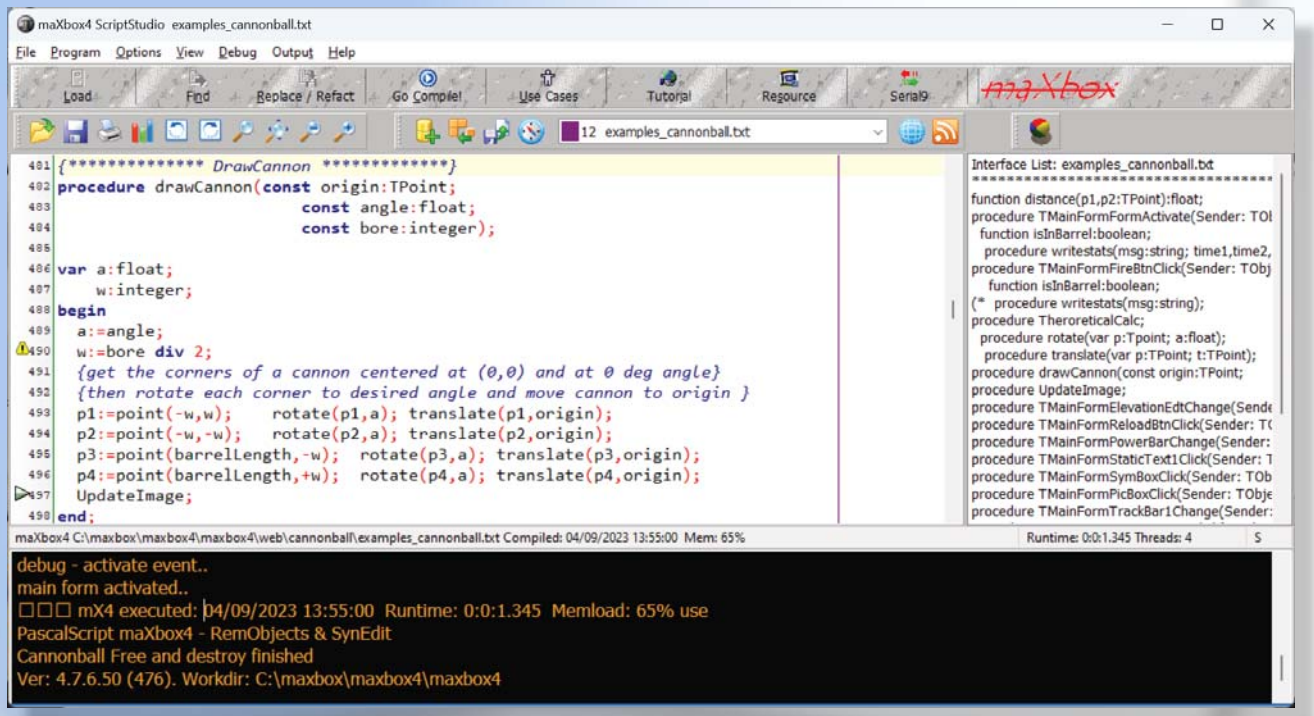
```
function distance(p1,p2:TPoint):float;
begin
 result:= sqrt(sqr(p1.x-p2.x)+sqr(p1.y-p2.y));
end;
```

Zwei Prozeduren, `Rotate` und `Translate`, sorgen für die Rotation von Punkten.
Die Drehung um einen Ursprungspunkt (0,0) ist recht einfach, wie der folgende Code zeigt:

```
 procedure rotate(var p:Tpoint; a:float);
{rotate a point to angle a from horizontal}
var t:TPoint;
begin
  t:=P;
  p.x:=trunc(t.x*cos(a)-t.y*sin(a));
  p.y:=trunc(t.x*sin(a)+t.y*cos(a));
end;

procedure translate(var p:TPoint; t:TPoint);
{translate a point by t.x and t.y}
Begin
  p.x:=p.x+t.x;
  p.y:=p.y+t.y;
end;
```

Sobald wir den Punkt um den gewünschten Winkel relativ zum Ursprung gedreht haben, kann `Translate()` den Punkt verschieben, indem die neuen x- und y-Ursprungskoordinaten zu den x- und y-Werten des Punktes vom Typ `TPoint` addiert.
Die andere Logik besteht darin, festzustellen, ob die Kanonenkugel das Ziel getroffen hat, das durch eine Leiste bewegt wird. Die "Kollisionserkennung" ist ein häufiges (*und auch kompliziertes*) Problem in den meisten animierten Grafik-Anwendungen.
Die Implementierung prüft, ob der Abstand vom Zentrum der Kanonenkugel kleiner ist als ihr Radius vom linken oder oberen Rand des Ziels nach jeder Bewegung oder jedem Treffer.
Das Problem besteht darin, dass bei kleinen Winkeln eine horizontale Bewegung die Kugel in einer Schleife von einer Seite des Ziels auf die andere bringen kann, so dass wir nie wissen, dass wir genau durch das Ziel hindurchgegangen sind! Eine lustige Sache ist die Speicherung von Kanonenkugeln; Kugelförmige Objekte, wie z. B. Kanonenkugeln, können zu einer Pyramide gestapelt werden, mit einer Kanonenkugel an der Spitze, die auf einem Quadrat aus vier Kanonenkugeln sitzt, das wiederum auf einem Quadrat bestehend aus neun Kanonenkugeln, und so weiter.



http://delphiforfun.org/Programs/bouncing_ball.htm

In **PyGame** for example,
collision detection is done using Rect objects. The Rect object offers various methods for detecting collisions between objects. Even the collision between a rectangular and circular object such as a paddle and a ball can be detected by a collision between two rectangular objects, the paddle and the bounding rectangle of the ball. Now we can summarize the theoretic results in a procedure of our statistic:

```
{************ TheoreticalCalc **********}
procedure TheroreticalCalc;
var
 root,T1, Vf, Vxf, Vyf, X1,Y1 : float;
 TTop, Xtop, Ytop, Tlast, VyLast, Xlast, floor : float;
begin
 with {stats.}amemo1.lines do begin
  clear;
  add(format('Barrel Len %d, Angle %6.1f, Initial V %6.1f, gravity %6.1f',
                          [barrellength,180*theta/pi,v1,g]));
  if g  = 0 then g := 0.001;
  root := v1*v1 - 2*g*sin(theta)*Barrellength;
  if root>=0 then begin
   T1  :=(v1 - sqrt(root))/(g*sin(theta+0.001));
   Vf  := v1 - g*sin(theta)*T1;
   Vxf :=Vf*cos(theta);
   Vyf :=Vf*sin(theta);
   X1  :=Barrellength*cos(theta);
   Y1  :=Barrellength*sin(Theta);
   floor:=(origin.y+ballradius)-groundlevel;
   {out of barrel, Vx remains constant, Vy := Vyf- g*DeltaT}
   {Vy=0 then Vyf-g*Ttop=0 or Ttop=Vyf/g}
   Ttop:=Vyf/g;
   {x distance at top} Xtop:=Vxf*Ttop;
   {height at top = average y velocity+ time}  Ytop:=(Vyf + 0)/2*TTop;
   {Time to fall from ytop to groundlevel, descending part of projectiles path}
   {speed when ball hits ground}
   TLast:=sqrt(2*(Y1+YTop-floor)/g );
   Xlast:=Vxf*TLast;
   add(format('Time in barrel %6.1f seconds',[T1]));
   add(format('X distance at end of barrel %6.1f',[X1]));
   add(format('Y distance at end of barrel %6.1f',[Y1]));
   add(format('Time to top of freeflight arc %6.1f, %6.1f total',[Ttop,T1+Ttop]));
   add(format('X distance top of freeflight arc %6.1f, %6.1f total',[Xtop,X1+Xtop]));
   add(format('Height above barrel to top of freeflight arc %6.1f, %6.1f total',
                                          [Ytop,Y1+Ytop]));
   add(format('Time to reach ground from max height %6.1f, %6.1f total',
                                          [TLast,T1+Ttop+TLast]));
   add(format('X distance from top of freeflight arc to end %6.1f, %6.1f total',
                                          [XLast,X1+Xtop+XLast]));
  end else add('Velocity too low, cannonball does not exit barrel');
 end;
end;
```

Übrigens habe ich ChatGPT gefragt, wie ich eine Kanonenkugel in Pascal programmieren kann, und die Antwort ist:
Um eine Kanonenkugel in Pascal zu programmieren, können Sie die folgenden Schritte verwenden:

1. Bestimmen Sie die Ausgangslage und die Geschwindigkeit der Kanonenkugel.
2. Berechnen Sie die Beschleunigung der Kanonenkugel aufgrund der Schwerkraft.
3. Aktualisiere die Geschwindigkeit und Position der Kanonenkugel mit Hilfe der berechneten Beschleunigung.
4. Wiederhole Schritt 3, bis die Kanonenkugel mit einem Objekt kollidiert oder eine bestimmte Höhe erreicht.

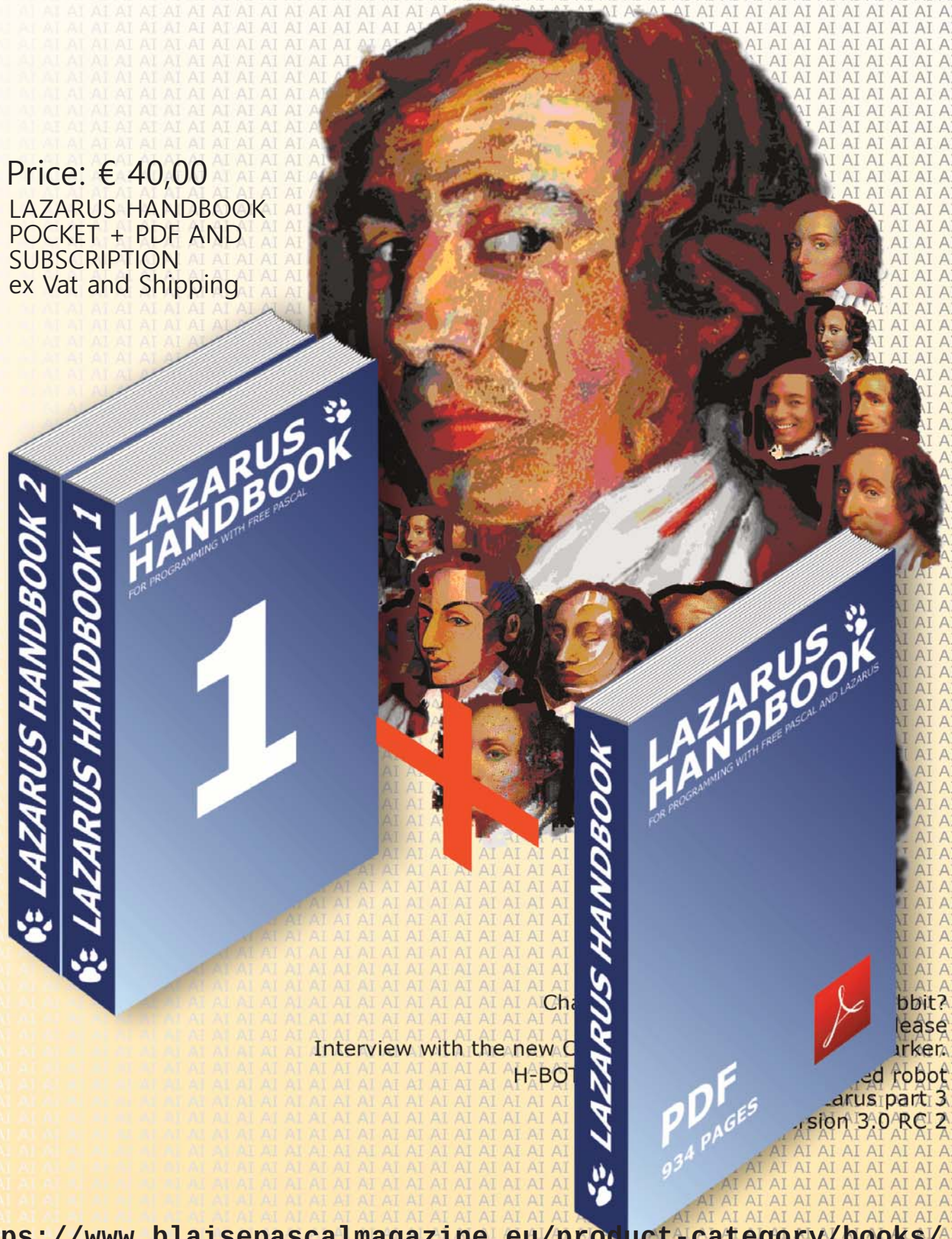
In diesem Beispielcodeausschnitt ist `CircleRectCollision()` eine benutzerdefinierte Funktion, die eine Kollision zwischen einem Kreis und einem Rechteck erkennt. Sie können diese Funktion an Ihre Bedürfnisse anpassen; der Hauptteil des Skripts hat nur 4 Prozeduren:

```
processmessagesOFF;  loadStatForm();  loadmainForm();  UpdateImage();
```

https://en.wikipedia.org/wiki/Ballistics **http://www.softwareschule.ch/examples/cannonball64.txt**

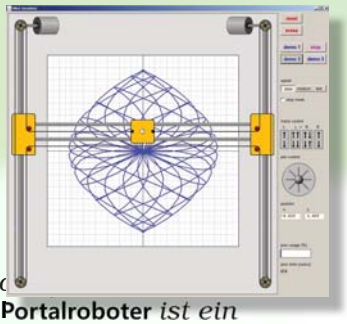
BLAISE PASCAL MAGAZINE 112

Databases / CSS Styles / Progressive Web Apps
Android / IOS / Mac / Windows & Linux



Cha bbit?
Interview with the new C lease
H-BOT arker.
ed robot
arus part 3
sion 3.0 RC 2

H-BOT, H-FÖRMIGER ROBOTER: EIN SIMULIERTER ROBOTER



EINFÜHRUNG:

Die Abbildung unten zeigt einen **HBot** (*H-förmiger Robo*
Diese Roboter werden häufig in Portalen eingesetzt (*ein* **Portalroboter** *ist ein automatisiertes industrielles System, das auch als kartesischer Roboter oder Linearroboter bezeichnet werden kann*), z. B. bei der SMD-Bestückung, der *Verpackung* und dem Laserschneidens. Aber auch unerwartete Anwendungen wurden gefunden: Falten von T-Shirts, Schachspielen, Tätowiermaschinen. Die Vorteile dieses Robotertyps sind Kosteneffizienz, Tragbarkeit und einfache Steuerung.

Abbildung 1: HBot

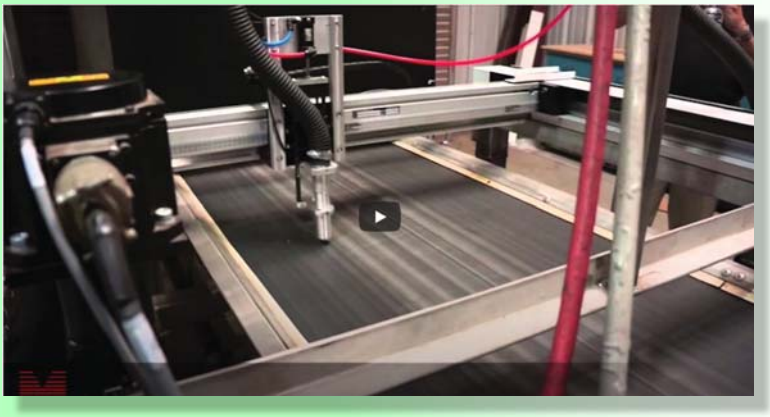


Abbildung 2: The video cover

Bitte sehen Sie sich das Youtube-Video an (siehe Abbildung 2)
`https://www.youtube.com/watch?app=desktop&v=NgGiu_0x7tg`

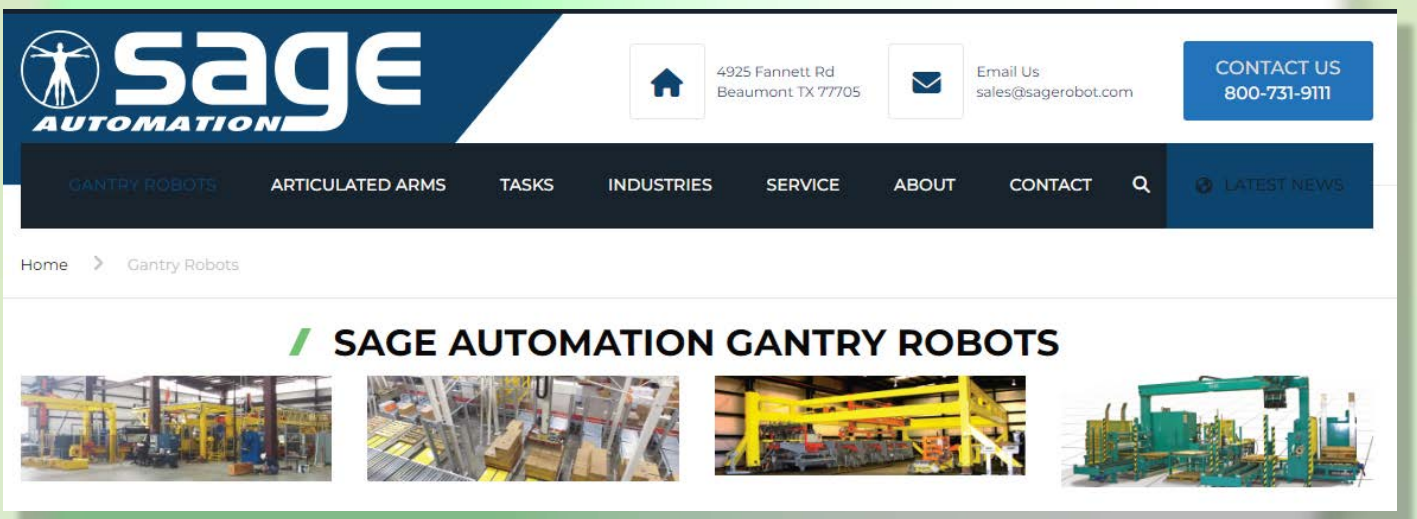
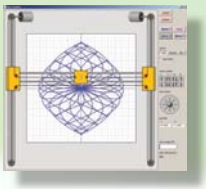


Abbildung 3: eine sehr gute Beispiel-Website
Hier erhalten Sie weitere Informationen zu den H-förmigen Robotern
`https://www.sagerobot.com/gantry-robots/`



Here is a video about a totally different application:
https://www.youtube.com/watch?v=Ztm-PrCzxos

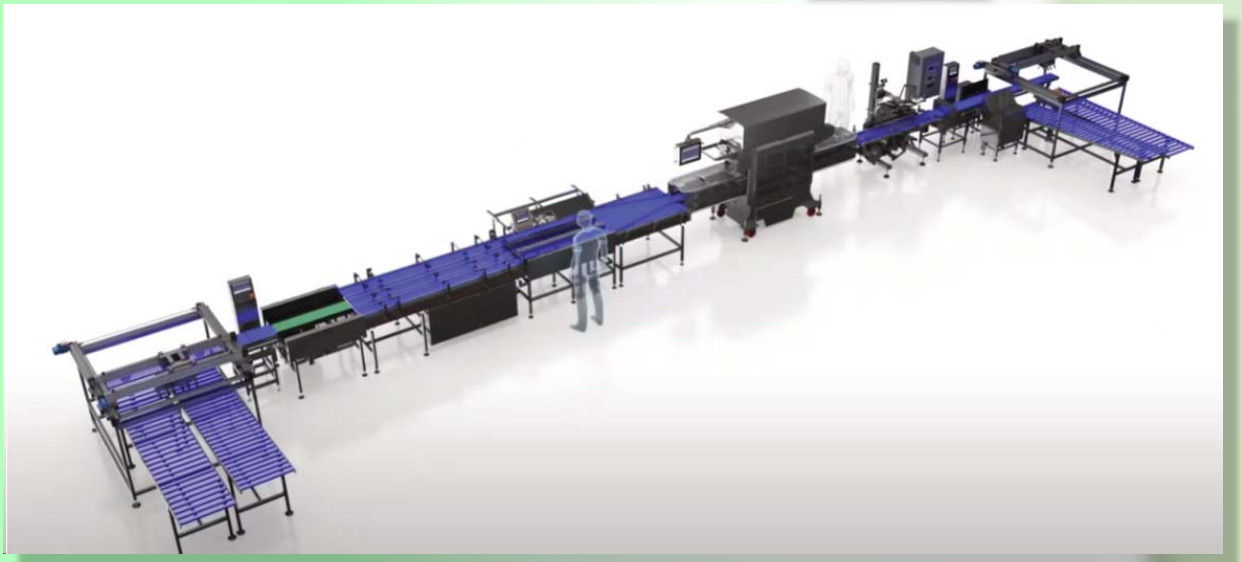


Abbildung 3: Handhabung von Pilzen

Die Abbildung unten (*verkleinert*) zeigt den simulierten Roboter, der als Plotter ausgeführt ist. Links und rechts oben befinden sich zwei feststehende Motoren, die unabhängig voneinander arbeiten und einen einzelnen Riemen antreiben.
Wenn sich die Motoren im oder gegen den Uhrzeigersinn drehen, bewegt sich der Stift in horizontaler, diagonaler oder vertikaler Richtung.

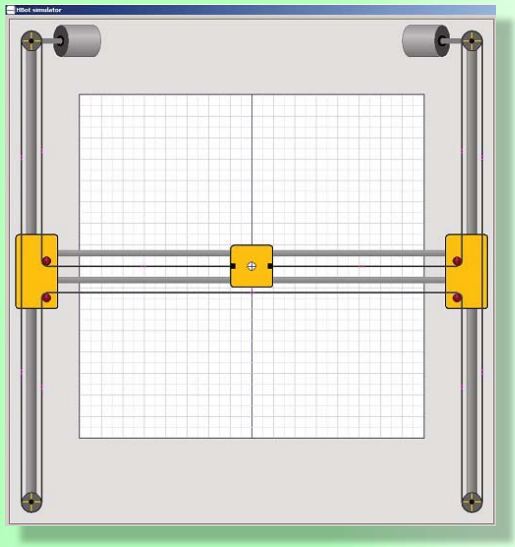
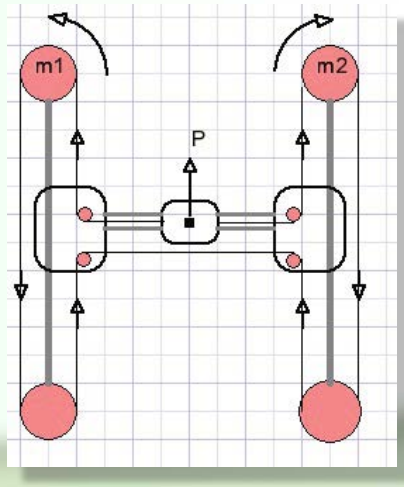
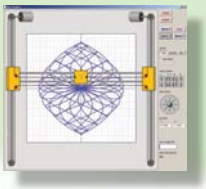


Abbildung 4: Motoren und Führungsstange

Es gibt einen einzigen Gürtel, dessen Enden am Stift befestigt sind.

Motor 1 bewegt sich gegen den Uhrzeigersinn,
Motor 2 bewegt sich im Uhrzeigersinn um die gleiche Strecke.
Der Stift P bewegt sich nach oben.





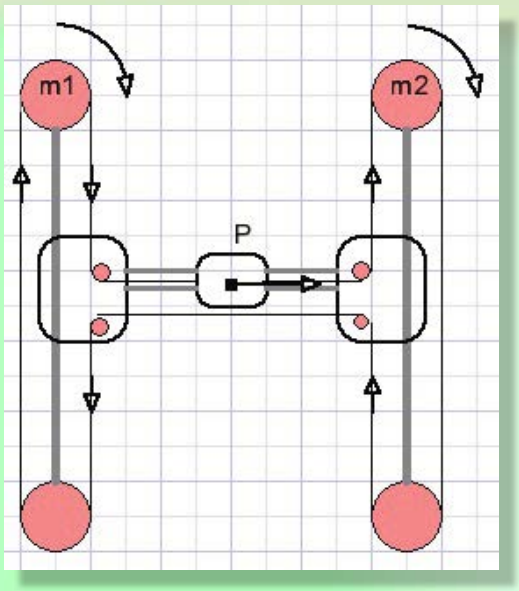


Abbildung 5: Horizontale Stiftbewegungen

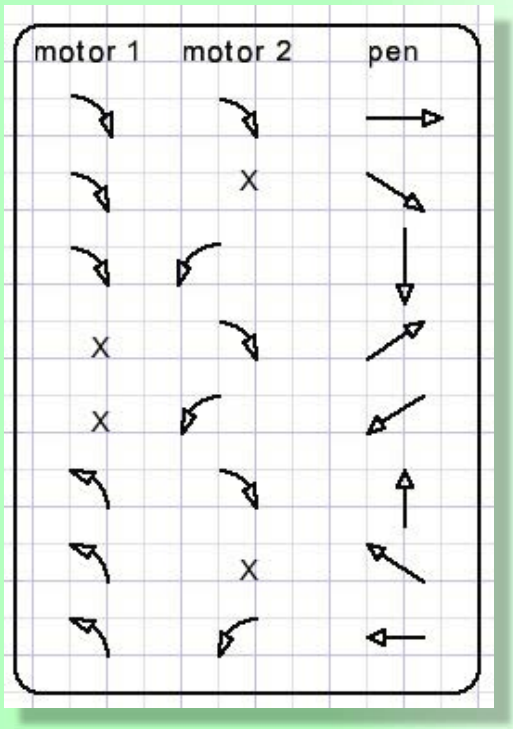


Abbildung 6: Aktionen

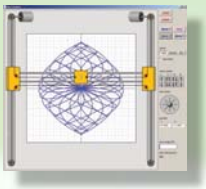
Horizontale Stiftbewegung.

Motor 1
bewegt sich im Uhrzeigersinn,
Motor 2
bewegt sich ebenfalls im Uhrzeigersinn, und zwar um den gleichen Betrag.
StiftP
bewegt sich horizontal nach rechts.
Diagonale Stiftbewegung
Ist das Ergebnis der vertikalen und horizontalen Bewegung.
Für Stift
Oben = Linker Motor CCW d ; Rechter Motor CW d (*Abstand*)
Rechts = Linker Motor CW d; Rechter Motor CW d

Aktionen hinzufügen
Diagonale AUF = Rechter Motor CW 2d
Die diagonale Bewegung nach rechts oben um die Distanz d ist das Ergebnis einer 2d-Distanzdrehung von Motor 2.
Die Bewegungen von Motor 1 heben sich gegenseitig auf.

Zusammengefasst:

Bei diagonaler Bewegung bewegt sich nur ein Motor.
Die Bandbewegung wird verdoppelt:
Sie ist die Summe aus horizontaler und vertikaler Bewegung.



SIMULATOR USAGE, BUTTONS AND INDICATORS

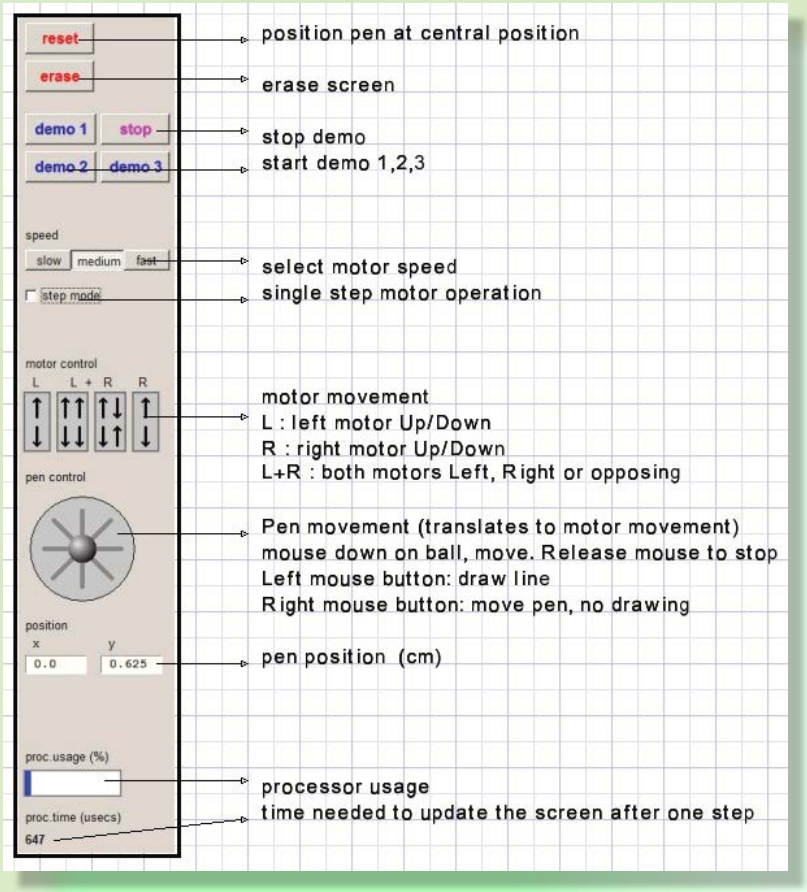


Abbildung 7: Die Programmnutzung

AREAS AND BITMAPS

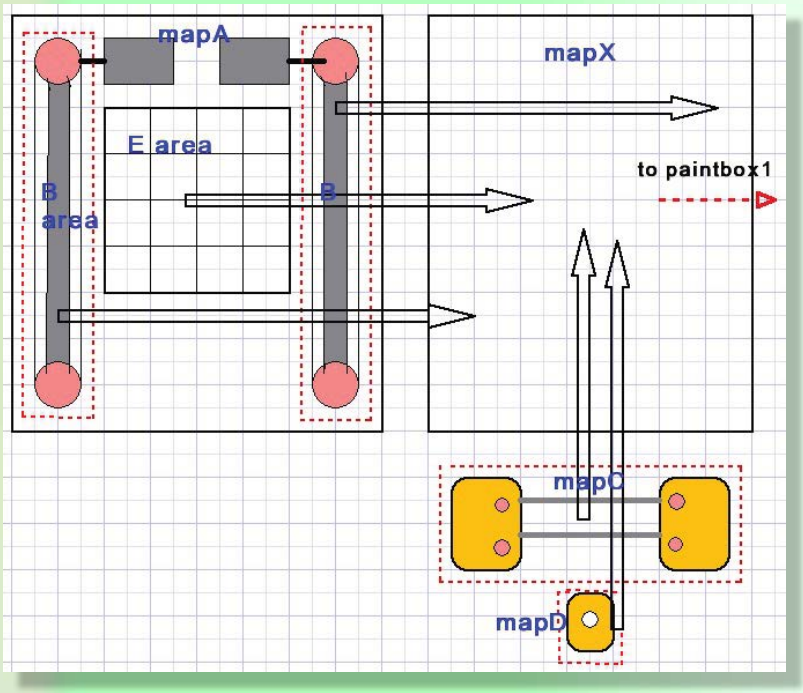


Abbildung 8: Das Mapping (Kartierung)

DAS SIMULATORPROGRAMM

Allgemeine Überlegungen

- Die Bewegungen müssen realistisch sein: fließend ohne Flackern
- Bewegliche Teile müssen als solche sichtbar sein
- Die Tasten müssen das Prinzip der Bedienung zeigen

Der **HBot** ist in einem Malkasten auf dem Hauptformular abgebildet. Dies ist jedoch nicht ausreichend, da das Löschen des Bildes vor dem Zeichnen der aktualisierten Situation ein störendes Flackern verursacht.
Dies wird vermieden, indem der HBot in eine Bitmap (`mapX`) eingebaut wird und veränderte Bereiche von `mapX` mit `copyrects` in die paintbox1 übertragen werden
.
Einige Bereiche des Bildschirms werden einmal gezeichnet und ändern sich nicht.

Dieser Hintergrund wird in `mapA` gezeichnet.
Teile von A werden in `mapX` übertragen.
Auch der Stift zeichnet in `mapA`.
Sich bewegende Räder haben angebrachte Speichen oder eine `Strichmarkierung`, um die Bewegung anzuzeigen.
Lila Punkte werden auf das Band gemalt, um die Bewegung anzuzeigen.

TIMING

Das Aktualisieren des Bildschirms nach einer Bewegung (ein Pixel) dauert einige Zeit.
Diese Zeit wird angezeigt.
Nach dem Aktualisieren des Bildschirms muss der Prozessor eine bestimmte Zeit auf die erforderliche Geschwindigkeit warten.
Ein blauer Balken zeigt den Prozentsatz der Zeit an, die für eine Aktualisierung um ein Pixel benötigt wird.

Die typische Zeit beträgt weniger als 800 Mikrosekunden.

MapA : Bitmap mit Motoren und B Bereichen mit vertikalen Führungen, Rädern und Riemen.
Bereich E ist zum Zeichnen und zeigt ein Koordinatensystem.
Karte A wird nach mapX kopiert.
Map C: Bitmap mit horizontalen Führungsschienen und Rädern. Wird nach `mapX` kopiert.
Karte D: Stifthalter. Wird nach `mapX` kopiert.
`MapX` wird (*teilweise*) in `paintbox1` kopiert, um nach dem Hinzufügen von Speichen zu den Rädern und von Punkten zum Gürtel sichtbar zu werden.

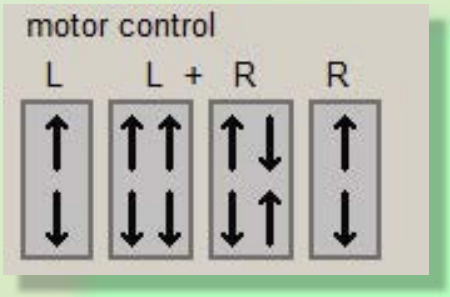


Abbildung 9: Die Motorsteuerung

Delphi-Code

Einzelheiten entnehmen Sie bitte dem Quellcode.

Einheiten

- **`Unit1`** : Simulatorsteuerung, Konstanten, Variablen, Malverfahren.
- **`Unit2`** : `TDav7ELbox` ist eine `Tpaintbox` mit hinzugefügten `enter leave events` und wird für Motorsteuerungstasten verwendet.
- **`Timer_unit`**: Code zur Umwandlung von CPU-Ticks in einen Mikrosekundentakt zur Geschwindigkeitssteuerung.
- **`Demo_unit`**: liefert 3 Demos zur Veranschaulichung des HBot-Betriebs.

Unit1 Prozeduren und Funktionen

`PaintmapA, paintmapC, paintmapD, paintLeftMotor, paintRightMotor:` selbsterklärend.
`PaintB` : malt Bereich B auf KarteA.
`PaintE` : malt das Koordinatensystem auf mapA.

```
procedure paintchain(mp: Tbitmap;x1,y1,x2,y2 : word);
// Gürtel auf Bitmap mp malen (x1,y1) (x2,y2)
```

`XpartialToBox`; // Bereich von mapC, B Bereiche paintbox,
dies sind die aktualisierten Rechtecke von mapX
`procAWheelmovement;` // Speichen auf Räder in B-Bereichen setzen.
`procCwheelmovement;` // Fügt Speichen an kleine Räder im C-Bereich an.
`procBeltMovement ;` // lila Punkte auf dem Gürtel hinzufügen
Bei den letzten drei Prozeduren wird die Position der Speichen oder Punkte anhand der Position des Stifts berechnet.
(0,0) ist der Mittelpunkt des Bereichs E.

Die Koordinaten auf E sind (`-320,-320`) links oben bis (`320,320`) rechts unten.
Der Code von `procBeltMovement` ist langwierig, weil das Band in horizontale und vertikale Strecken und auch die Bögen um die Räder unterteilt ist.

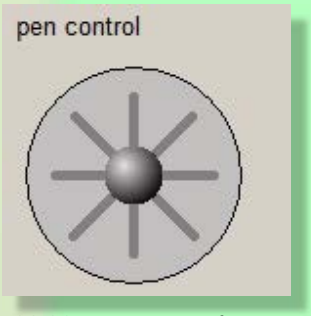


Abbildung 10: Die Stiftsteuerung

BEWEGEN DES STIFTS.
Dieses Simulatorprojekt ist für Ausbildungszwecke gedacht.
Aus diesem Grund gibt es zwei Möglichkeiten, den Stift zu bewegen:
1. Durch Steuerung der Motoren und Beobachtung der Stiftbewegung.
2. Durch die Steuerung des Stifts und die Beobachtung der Motoren.

Abbildung links zeigt vier Farbkästen mit hinzugefügten Ereignissen beim Betreten und Verlassen. Der Code wird von unit2 bereitgestellt.
Diese `Paintboxen` werden zur Laufzeit erstellt.

Ein Druck mit der linken **Maustaste** auf die obere Hälfte der **L-Taste** bewegt den linken oberen Motor gegen den Uhrzeigersinn. Die untere Hälfte bewirkt eine Bewegung im Uhrzeigersinn. Mit der rechten **Maustaste** werden diese Aktionen umgekehrt.
Der **R-Malkasten** funktioniert auf ähnliche Weise, wobei eine gedrückte Maustaste auf dem oberen Teil nun eine Bewegung im Uhrzeigersinn bewirkt.
Die Tasten **L+R** aktivieren beide Motoren, die sich entweder in die gleiche oder in entgegengesetzte Richtung bewegen.

UM DEN STIFT ZU BEWEGEN, OHNE ZU ZEICHNEN:

Rechte Maustaste auf der Kugel drücken, in die gewünschte Richtung bewegen. Lassen Sie die Maustaste los, wenn der Stift seine Position erreicht hat. Verwenden Sie die **linke Maustaste**, um den Stift zum Zeichnen zu bewegen.
Wenn die Maus auf der Kugel gedrückt wird, wird das `movebusy-Flag` gesetzt.
Das `movebusy-Flag` aktiviert **Mausbewegungsereignisse**.
Der Malbereich der Stiftsteuerung ist in Zellen unterteilt, die Zellennummer wird in den Richtungscode (`xdircode`) übersetzt.

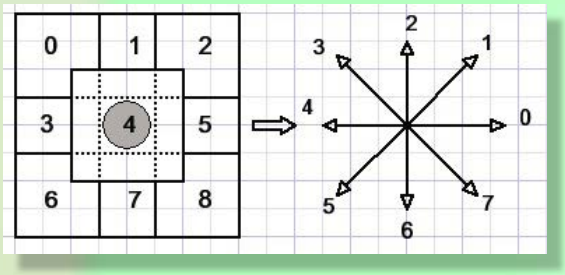


Abbildung 11: Die Zellen des Farbkastens

Rechts sind die Zellen des Malkastens abgebildet und rechts wird der Richtungscode angezeigt.
Die Prozedur `procXpainting` wird aufgerufen, die das `Moveflag` setzt und kontinuierlich die Prozedur procmove aufruft, solange das `Moveflag` gesetzt ist.
Procmove kümmert sich um die Geschwindigkeit. Bei Operationen mit einem Motor verdoppelt sich die Zeitspanne, da der Motor die doppelte Strecke zurücklegen muss, um den Stift diagonal zu bewegen.
Procmove ruft die Prozedur `moveControl` auf, um die neue Stiftposition (`penPosX`, `penPosY`) zu berechnen, `mapX` zu aktualisieren und Teile von `mapX` in paintbox1 auf form1 zu kopieren.
Wenn die äußeren Grenzen des E-Bereichs erreicht sind, löscht `procMove moveFlag` und die Stiftbewegung stoppt.

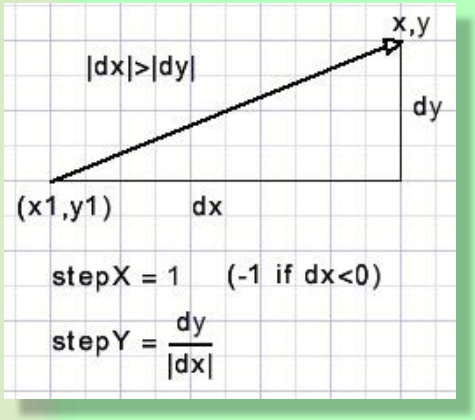


Abbildung 12: horizontale Ausrichtung

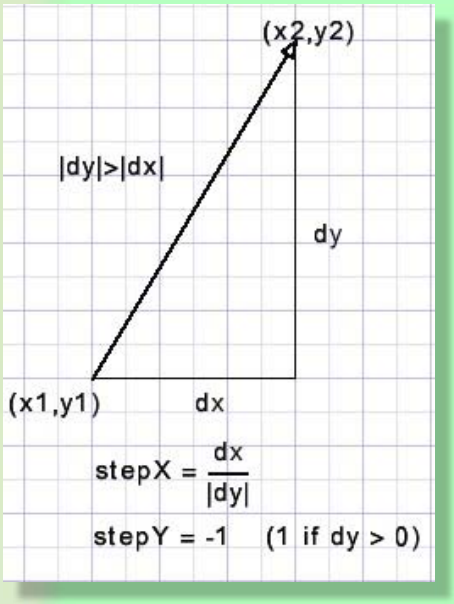


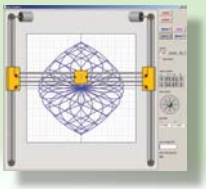
Abbildung 13: Vertikale Ausrichtung

DIE DEMO-UNIT

Es werden drei Demos bereitgestellt. Die Demo-Schaltflächen haben die **Tags 1..3** und teilen sich das Ereignis `OnClick`. Ein Klick auf eine Demo-Schaltfläche ruft die Prozedur `startdemo(demoNr)` auf.

DEMO 1:
Zeichnet die Parameterfunktion
`X = 150(sin(8t)+sin(t))`
`Y = 150(cos(8t)+cos(t))`
Dabei geht t von 0...2*pi in 500 Schritten.
Für jeden Wert von t werden x und y berechnet und die Prozedur `movetoXY` wird aufgerufen, um eine Linie zur neuen (`x, y`) Position zu ziehen.

DEMO 2:
Ähnlich wie demo1, aber die Funktion lautet
`X = 500(sin(9t)*cos(9t)*sin(7t))`
`Y = 275(sin(9t)*cos(7t))`
Prozedur `movetoXY(x,y)`
Diese Prozedur ruft anschließend die Prozedur `procmove(direction)` auf, um die Stiftposition `(x,y)` zu erreichen; Dies geschieht, indem zuerst die Werte `Xstep` und `Ystep` berechnet werden und dann x (um `Xstep`) und y (um `Ystep`) inkrementiert werden;
Es muss zwischen horizontaler und vertikaler Linienausrichtung unterschieden werden.



DEMO 3:
In dieser Demo werden einige Textzeilen geplottet.
Der Text wird im `Array` **demotext** zusammen mit den Koordinaten des ersten Zeichens, der Schrifthöhe und natürlich der Zeichenkette vorgegeben.
Die Prozedur **`startdemo(3)`** ruft die Prozedur `painttextline(line nr)` für jede Textzeile auf.
`Painttextline` ruft die Prozedur `drawdemochar(x,y)` für jedes Zeichen der Zeile auf.
`Drawdemochar` ruft schließlich `movotoXY(x,y)` und `procmove(direction)` auf.
Bleibt noch zu erklären, wie die Parameter für die Prozeduren `movetoXY ( )` und `procmove( )` berechnet werden.
Eine Bitmap namens **scanmap** (*width=60, height=60 Pixel*) wird für jedes Zeichen mit einem schwarzen Hintergrund gelöscht. Dann wird das Zeichen in scanmap mit roter Stiftfarbe und weißem Hintergrund gezeichnet. Die schwarze Farbe zeigt die Grenzen des Zeichens an.

Abbildung 14: Zeichnen einer Figur

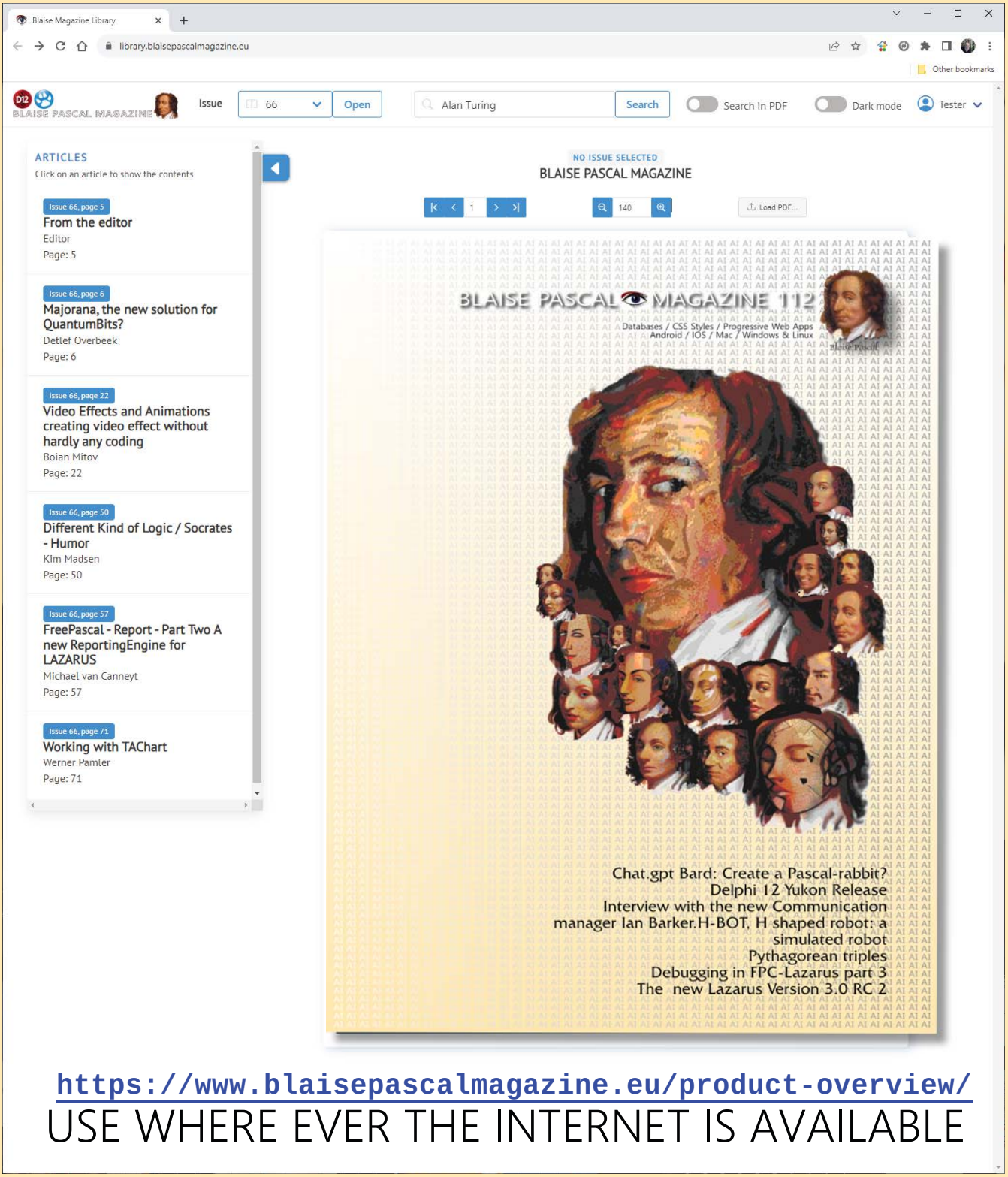
SCANMAP (Abtastungskarte)

Drawdemochar hat Variablen `scanX`, `scanY` und `x, y`, die auf Pixelpositionen der **Scanmap** zeigen.
Zuerst werden scanX und scanY auf 0 gesetzt;
Dann wird die Funktion `scanChar(var scanX,scanY) : Boolean;` wird aufgerufen. Diese Funktion scannt die **Scanmap** (von links nach rechts, von oben nach unten), um ein rotes Pixel zu finden, und gibt true zurück, nachdem dieses rote Pixel auf weiß gesetzt wurde.
`ScanX, scanY` zeigen auf das erste gefundene rote Pixel.

Als nächstes senkt drawdemochar den Stift, um einen Punkt zu setzen. x wird auf scanX gesetzt, y wird auf scanY gesetzt und Funktion `subscan(var x,y; var dir) : Boolean` wird aufgerufen.
Diese Funktion sucht nach roten Nachbarpixeln von x,y.
Falls gefunden, wird true zurückgegeben, zusammen mit dem Richtungscode, der für den Aufruf der Prozedur `procmove( )` benötigt wird, um den Stift zu bewegen.Außerdem werden die x,y-Koordinaten aktualisiert, um die Position des letzten roten Pixels wiederzugeben.
Wenn `subscan false` zurückgibt, wird `scanChar` erneut aufgerufen, um die Suche nach den verbleibenden roten Punkten fortzusetzen, beginnend bei `scanX, scanY.`
Die Schriftart **Roman** wird zum Zeichnen der Zeichen in scanmap verwendet.
Das `GoDemoFlag` muss true sein, damit die Textzeichnung fortgesetzt werden kann.
Ein Klick auf den Stop-Button löscht das Flag und beendet die Demo.

THE NEW SUBSCRIPTION MODEL OF BLAISE PASCAL MAGAZINE

1. SUBSCRIPTION: PER YEAR - NOTHING CHANGES ISSUES STARTING AT THE LATEST ISSUE AVAILABLE +1 YEAR / CODE INCLUDED € 70,00 FOR ALL COUNTRIES INCLUDED INTERNET (LIBRARY) USE FOR ALL MAGAZINES FROM 1- THE LATEST ISSUE FOR ALL COUNTRIES
2. LIB-STICK USB-CARD: ALL ISSUES / CODE INCLUDED. SAME INTERFACE AS THE INTERNET LIBRARY.€ 120,00 FOR ALL COUNTRIES

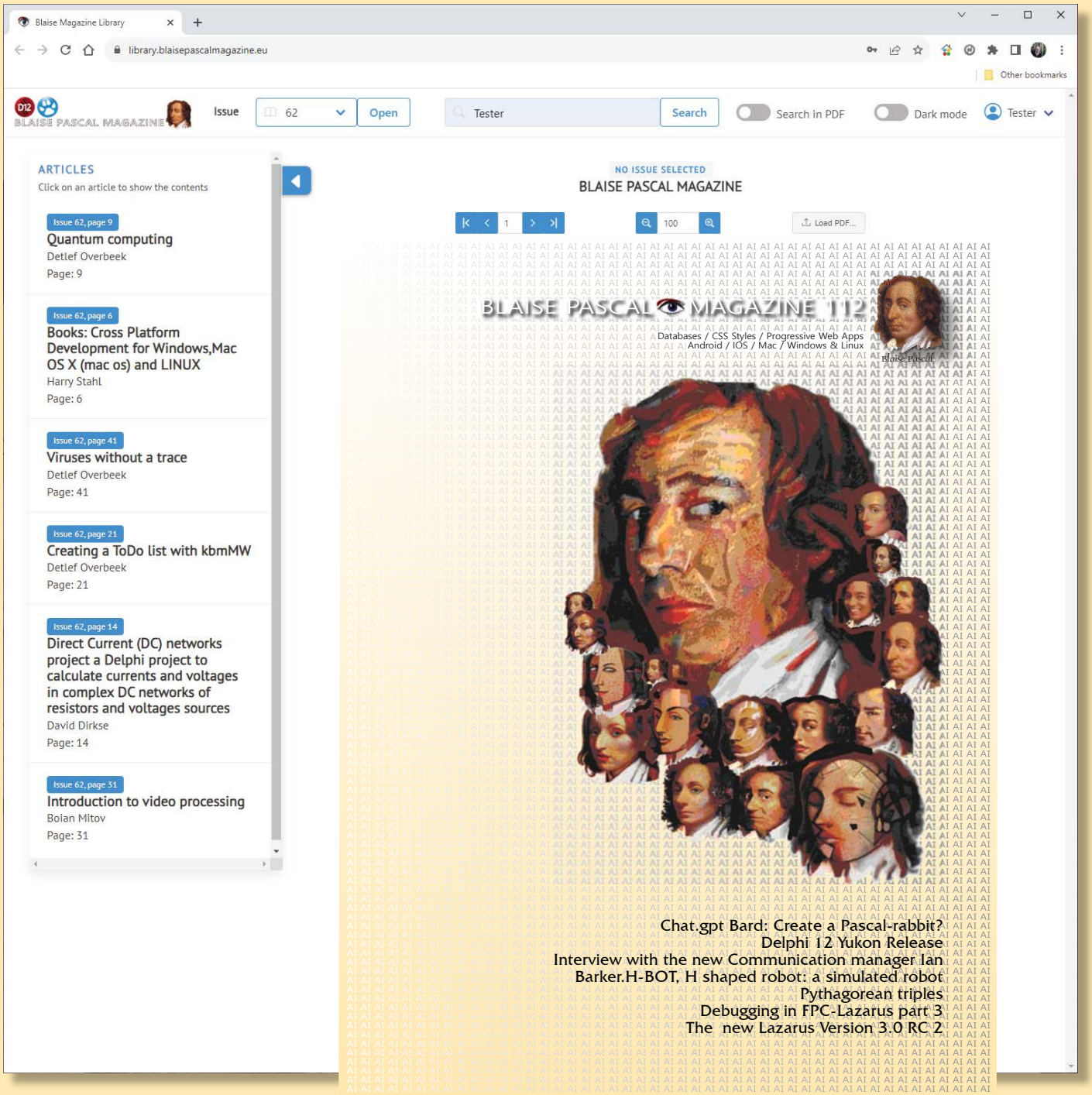


https://www.blaisepascalmagazine.eu/product-overview/

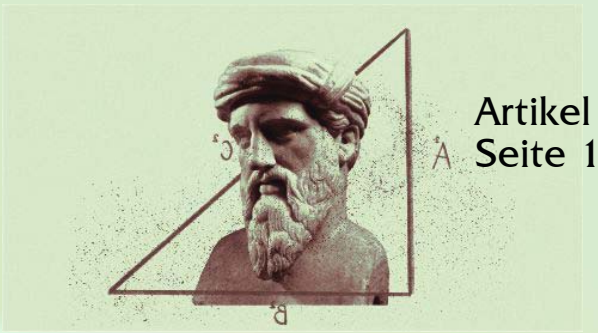
USE WHERE EVER THE INTERNET IS AVAILABLE

LIB-STICK ON USB CREDIT CARD BLAISE PASCAL MAGAZINE

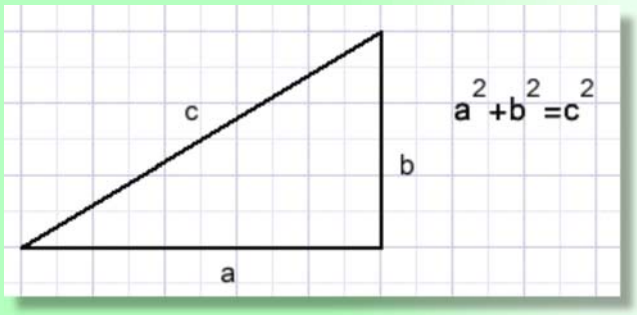
LIB-STICK USB-CARD: ALL ISSUES / CODE INCLUDED. SAME INTERFACE AS THE INTERNET LIBRARY € 120,00



PYTHAGOREISCHE DREIERGRUPPEN



Starter Expert D11



EINLEITUNG

In der Sekundarschule werden die Schüler mit dem Satz des **Pythagoras** konfrontiert:

In einem rechtwinkligen Dreieck ist das Quadrat der **Hypotenuse** c gleich der Summe der Quadrate der rechtwinkligen Seiten a und b.
Berechnungen mit der obigen Formel ergeben in der Regel Wurzeln, Zahlen, die nur angenähert werden können.
Einige Werte von a und b führen jedoch zu einem ganzzahligen Wert von c, wie zum Beispiel

- `3,4  c=5`
- `5,12 c=13`

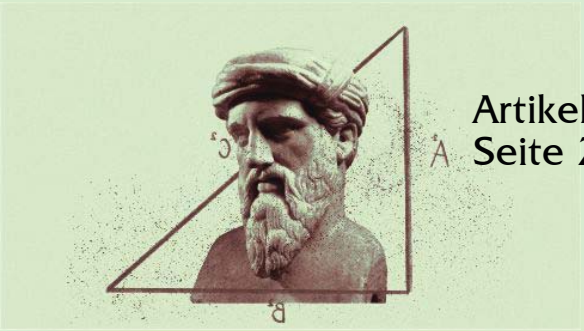
`(3,4,5)` und `(5,12,13)` werden als pythagoreische Tripel bezeichnet.
Es stellt sich die Frage: Gibt es noch mehr `Tripel`?

Dieses Projekt wurde geschrieben, um alle Dreiergruppen unter 1000 zu finden.

ANALOGE DREIECKE

Da `(3,4,5)` ein Tripel ist, bedeutet dies, dass auch `(6,8,10), (9,12,15)` ... Tripel aus ähnlichen Dreiecken sind.
Ein einfaches Kriterium `GCD(a,b) = 1` eliminiert diese analogen Dreiecke

```
function GCD(a,b : word) : word; //greatest common divisor
var h : word;
begin
  repeat
  if a < b then
    begin
      h := a;
      a := b;
      b := h;
    end;
  a := a mod b;
  until a = 0;
  result := b;
end;
```



Diese Funktion verwendet das
Euklidische Lemma:
`GCD(a,b) = GCD(a mod b,b).`
Beispiel: `GCD(77,21) = GCD(14,21)=GCD(21,14)=GCD(7,14)=GCD(14,7)=GCD(0,7)=7`

Zeitmessung.
Dem Projekt wird eine **Mikrosekunden-Timerkomponente** hinzugefügt.
Um die tatsächliche Verarbeitungszeit zu ermitteln, ohne die Last der Berichterstattung (`memo1.lines.add( string)`) werden die erkannten Tripel zunächst im Array `ptriples[ ]` gespeichert.

```
type TP3 = record
   a,b,c : word;
   end;
...
var p3nr : byte; //sequence number of triple
   ptriples : array[1..200] of TP3;
```

Das hier vorgestellte Projekt verfügt über drei wählbare Verfahren, um pythagoreische Dreiergruppen zu finden.

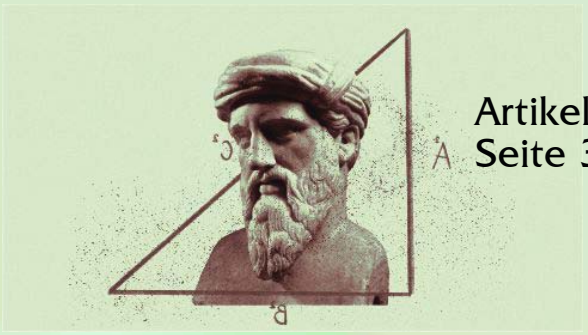
METHOD NR 1.
Uses no floating point operations and a preset table of squares.

```
Var    squares : array[1..1500] of dword;
...
procedure presetsquares;
   var i : word;
begin
   for i := 1 to 1500 do squares[i] := i*i;
end;
```

SUMMARY:

```
Var c2 : dword;
...
{a and b are incrementing variables in nested repeat..until loops}
C2 := squares[a] + squares[b];
C  := b;
{a third nested repeat..until loop increments c}
If  c2 =  squares[c] then  ....   //report a,b,c  as new triple
```

Dies ist die langsamste Methode. Die Verarbeitungszeit beträgt über 100 Millisekunden.



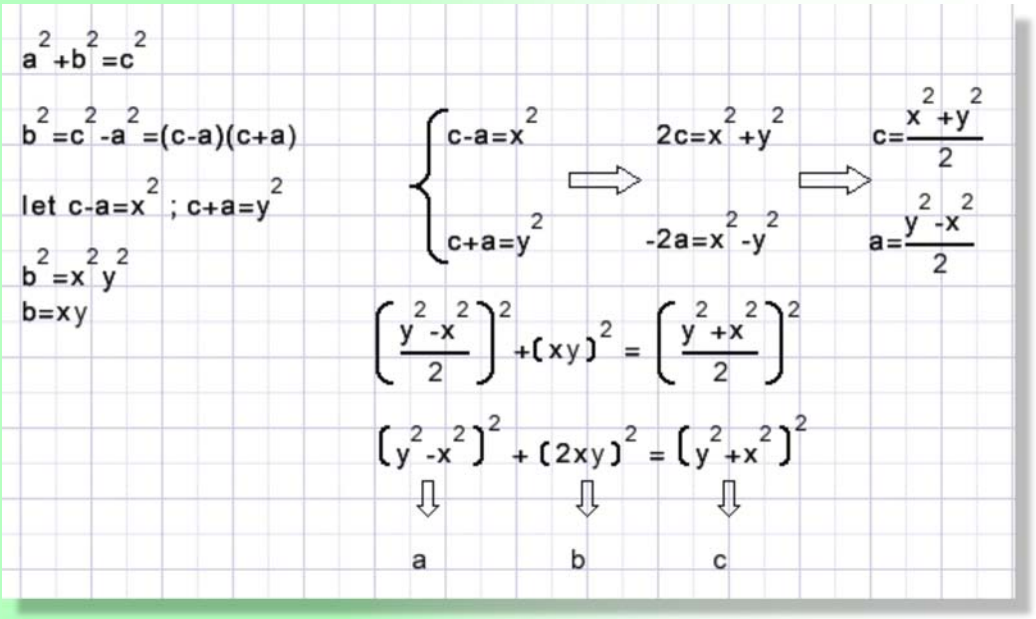
METHODE 2

```
procedure p3method2;
var a,b   : word;
    a2, b2 : dword;
    c     : single;
begin
 form1.proctimer.start;
 for a := 1 to 998 do
  begin
    a2 := a*a;
    for b := a+1 to 999 do
    begin
      b2 := b*b;
      c := sqrt(a2 + b2);
      if frac(c) = 0 then
        if GCD(a,b) = 1 then addtriple(a,b,round(c));
    end;
   end;
 form1.proctimer.stop;
end;
```

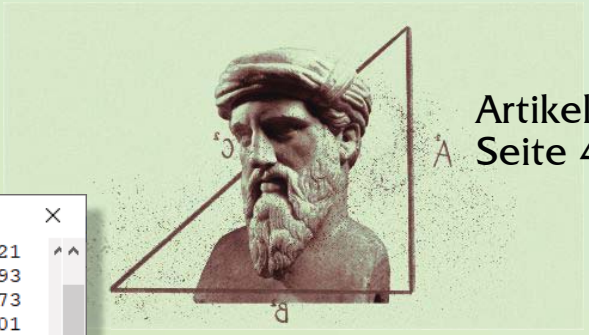
Diese Methode hat zwei verschachtelte for-Schleifen, um a und b zu inkrementieren.
Sie verwendet die Fließkommaoperationen sqrt( ) und `frac( )`, um die Quadratwurzel zu berechnen und den Bruch zu extrahieren.
Wenn `frac(V) = 0` ist, ist der Wert v eine ganze Zahl.
Die Verarbeitungszeit beträgt etwa 40 Millisekunden.

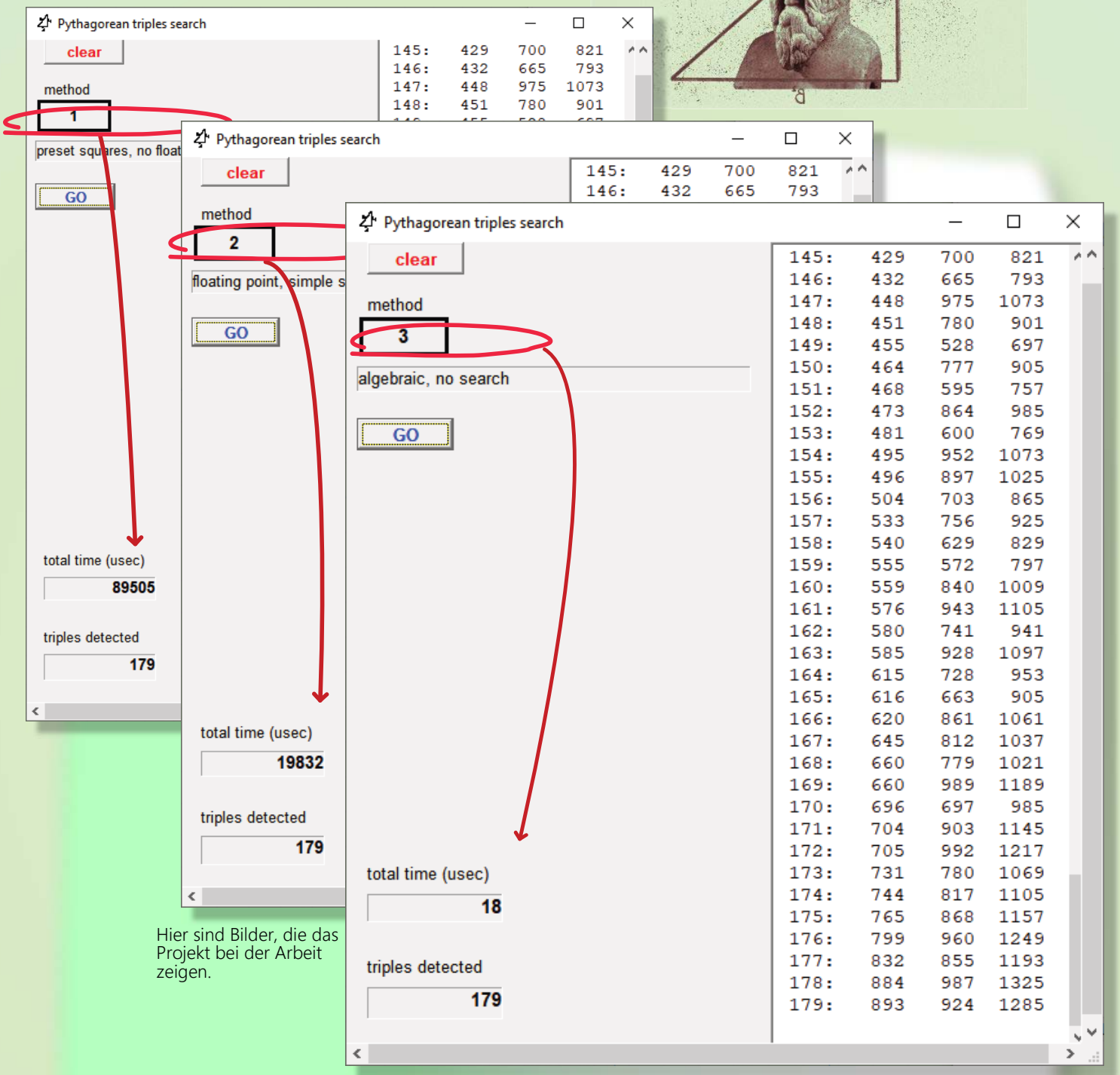
METHODE 3

Bei dieser Methode wird nicht nach Dreiergruppen gesucht, sondern es werden Formeln verwendet, die Dreiergruppen ergeben.
Im Folgenden wird die Theorie erläutert:



`a,b,c` werden durch `x,y` ersetzt.





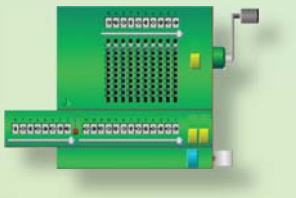
Hier sind Bilder, die das Projekt bei der Arbeit zeigen.

Jede Kombination, bei der x<> y ist, erzeugt ein Tripel.
Die Verarbeitungszeit beträgt 85 Mikrosekunden. Im Gegensatz zu den Methoden 1. und 2., bei denen a und b systematisch inkrementiert wurden, müssen die Tripel sortiert werden, um die gleiche Ergebnisfolge zu erhalten. Es wird ein einfaches Austausch-Sortierverfahren verwendet. Die Sortierzeit wird nicht gemessen.
Einzelheiten sind dem Quellcode zu entnehmen.

Auswahl der Methode
Ein einfaches Etikett wird als Schaltfläche verwendet.
Auf dem Canvas des übergeordneten Formulars (Form1) werden Ränder gezeichnet, die bei einem Enter/Leave-Ereignis die Farbe wechseln.

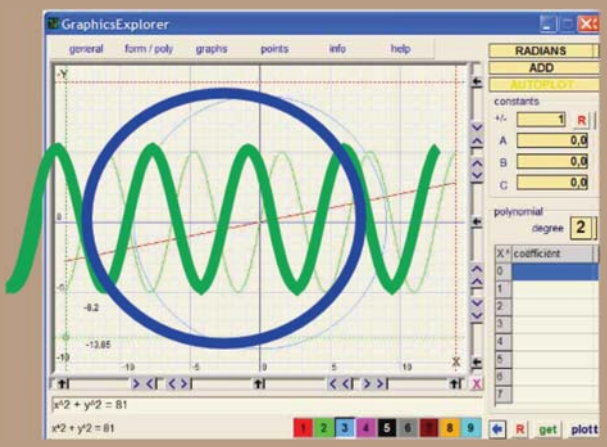
Ein Linksklick auf das Etikett wählt die nächste Methode aus, ein Rechtsklick die untere Methode.

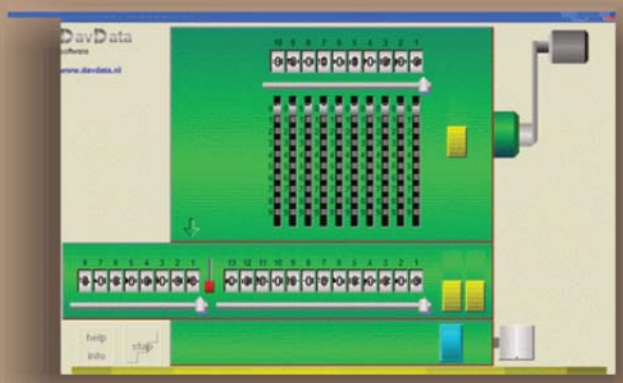
ADVERTISEMENT



DAVID DIRKSE

including 50 example projects





COMPUTER (GRAPHICS) MATH & GAMES IN PASCAL

https://www.blaisepascalmagazine.eu/product-category/books/

ease see our products here:
ttps://www.nexusdb.com

20 years of

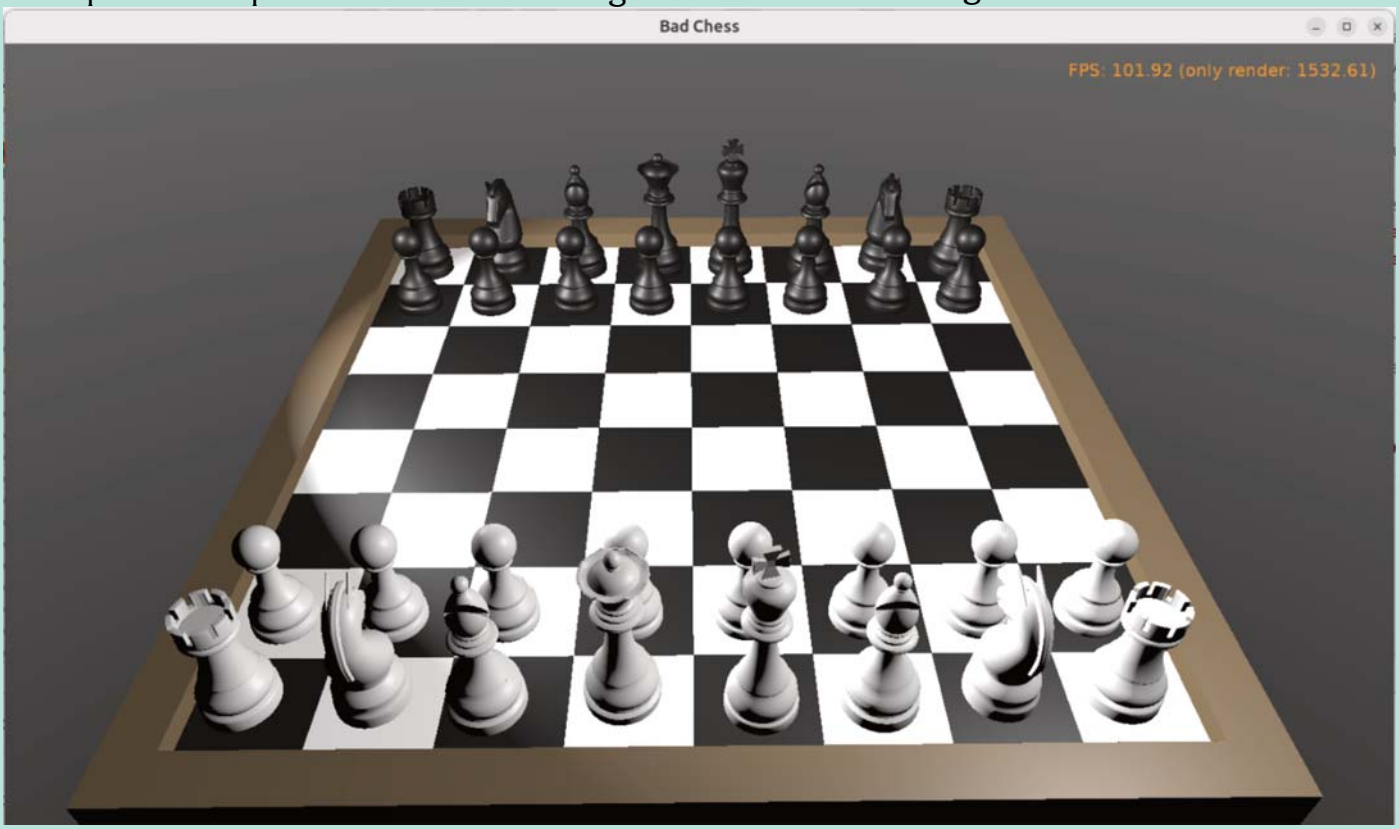
nexusdb

Our YouTube channel: https://www.youtube.com/@nexusdb9051

DIE SCHLECHTE ART, SCHACH ZU SPIELEN:
3D-PHYSIK-SPASS MIT DER CASTLE GAME ENGINE (TEIL 1)

VON MICHALIS KAMBURELIS

Überblick über diesen Artikel



❶ EINLEITUNG

Ich erinnere mich an mein erstes Buch über Schach, als ich ein Kind war. Es war ein Buch, das jungen Menschen beibrachte, wie man Schach spielt. Das erste Kapitel begann mit einer Geschichte über Kinder, die falsch Schach spielten:

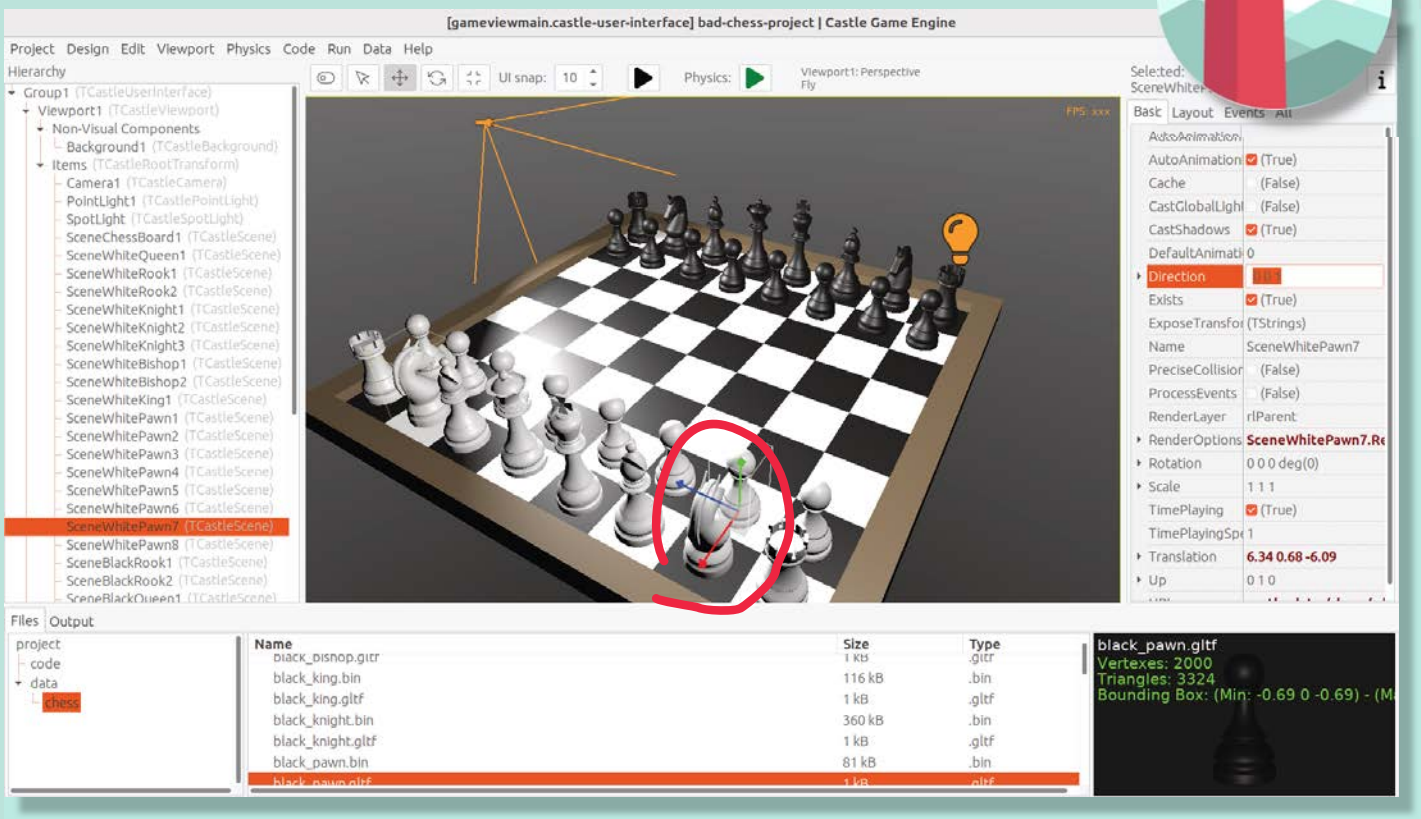
Sie kannten die Regeln nicht, also legten sie die Schachfiguren wahllos auf das Schachbrett und schnippten sie mit den Fingern auf die andere Seite. Die hölzernen Schachfiguren flogen durch die Luft und prallten gegeneinander. Schließlich fielen die meisten Schachfiguren vom Schachbrett auf den Boden. Derjenige, der die letzte Schachfigur auf dem Schachbrett übrig hatte, war der Gewinner.

Das war natürlich eine schlechte Art, Schach zu spielen. Im zweiten Kapitel des Buches kam ein Erwachsener, sagte den Kindern, dass sie Schach falsch spielen, und lehrte sie die richtige Art und Weise - wie sich jede Figur bewegt, wie der König etwas Besonderes ist, was es bedeutet, zu schachern und dann den Gegner matt zu setzen.

Das Buch war insgesamt großartig, und es ist wahrscheinlich dafür verantwortlich, dass ich Schach (die richtige Version des Spiels, mit Regeln anstelle von schnipsenden Gegenständen) bis heute liebe.
Davon abgesehen... Willst du nicht auch einmal diese "falsche" Version des Schachspiels spielen, die Kinderversion, bei der es auf nichts anderes ankommt, als jede Schachfigur auf die andere Seite zu schicken?

In dieser Artikelserie werden wir in der Zeit zurückgehen, unser hart erarbeitetes Wissen darüber, wie man wirklich Schach spielt, auslöschen und eine einfache 3D-Physik-Spaßanwendung implementieren, in der Sie Schachfiguren mit Hilfe der Physik schnippen können. Sie können es als ein Spiel für 2 Personen behandeln - spielen Sie es einfach auf einem Computer, und lassen Sie jeden Spieler abwechselnd die Maus und die Tastatur benutzen.

THE BAD WAY TO PLAY CHESS:
3D PHYSICS FUN USING CASTLE GAME ENGINE (PART 1)



❷ DIE EIGENTLICHE EINFÜHRUNG

Der eigentliche Zweck dieses Artikels ist es, eine unterhaltsame, aber auch nützliche Einführung in die Verwendung der **Castle Game Engine**.
Castle Game Engine ist eine plattformübergreifende (`Desktop, Mobile, Konsolen`) 3D- und 2D-Spiele Engine. Wir werden lernen, wie man ein Spiel für Desktops (*Linux, Windows, macOS, FreeBSD*) erstellt.
Im ersten Teil des Artikels werden wir zeigen, wie man ein 3D-Schachbrett und Schachfiguren mit dem **Castle Game Engine Editor** entwerfen und wie man die Physik verwendet. Im nächsten Teil werden wir etwas in Pascal programmieren, um die Spiellogik zu implementieren. In zukünftigen Artikeln möchten wir auch die Entwicklung für andere Plattformen (*wie Android und iOS*) und zukünftige Pläne (*wie die Web-Plattform*) zeigen.

Sie können FPC oder **Delphi** verwenden, um die hier vorgestellte Anwendung zu entwickeln. In unserer Engine bemühen wir uns um eine perfekte Unterstützung für beide Pascal-Compiler. Beachten Sie jedoch, dass Sie mit Delphi derzeit nur Windows ansprechen können (*mit FPC sind alle Plattformen verfügbar*).

Castle Game Engine bietet einen leistungsstarken visuellen Editor, um Ihre Spiele in 2D oder 3D zu entwerfen. Genau wie die visuellen Bibliotheken von **Delphi** und **Lazarus** basiert alles auf einem einfachen RAD-Konzept: Sie können eine funktionale Anwendung einfach visuell entwerfen, aber gleichzeitig wird alles, was Sie tun, mit Pascal-Klassen und -Eigenschaften ausgeführt.
So ist all Ihr Wissen, das Sie bei der Benutzung des Editors gewonnen haben, auch nützlich, wenn Sie **Pascal-Code** schreiben müssen. Sie werden die gleichen Klassen und Eigenschaften in **Pascal** verwenden, die Sie im visuellen Editor gesehen haben.

Die Engine ist frei und Open-Source. Verwenden Sie sie, um Open-Source- oder proprietäre Anwendungen zu entwickeln. Sie können sie auf beliebige Weise an Freunde weitergeben, Sie können sie auf `Steam, Itch.io, Google Play (Android), AppStore (iOS),` Ihrer eigenen Website veröffentlichen - überall.

❸ HERUNTERLADEN UND INSTALLIEREN DES MOTORS

Laden Sie zunächst den Motor von unserer Website herunter:
`https://castle-engine.io/download.`

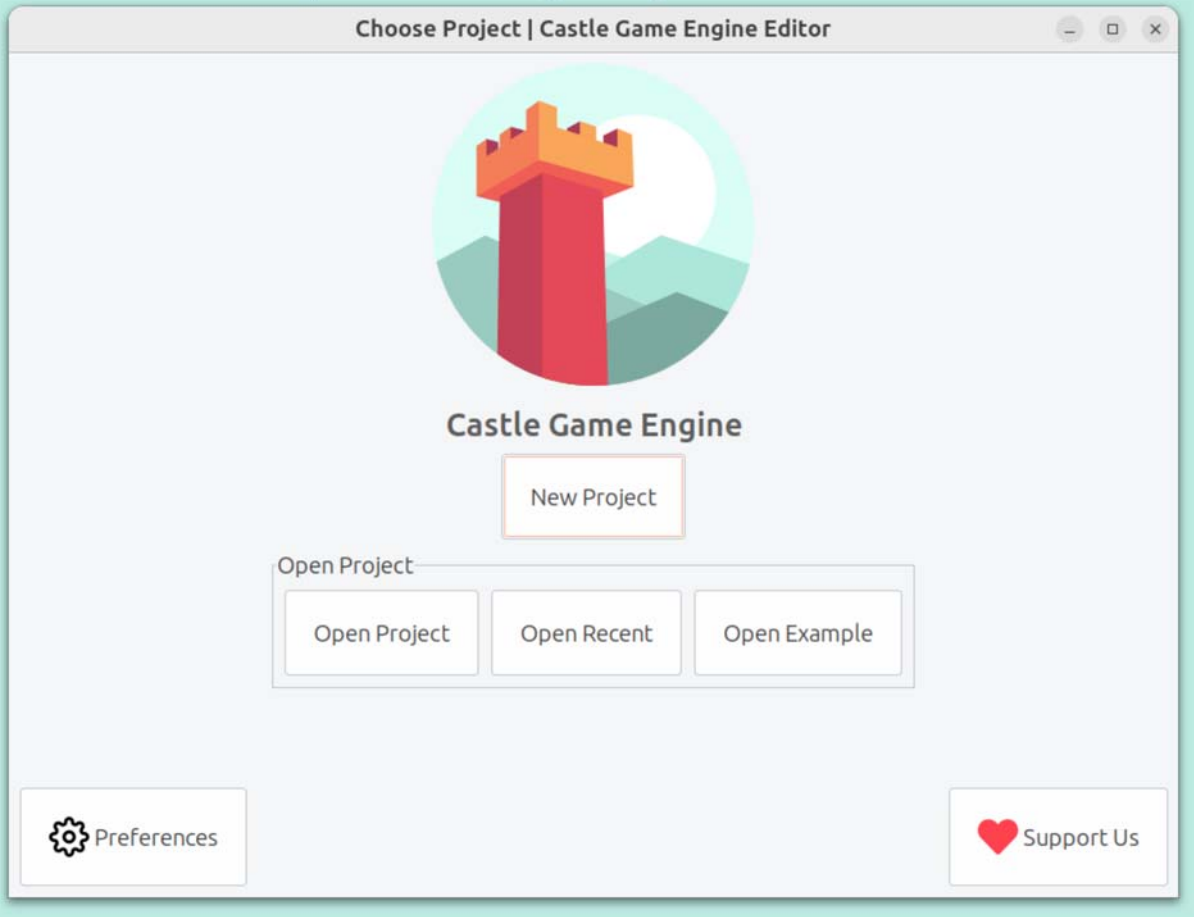
Wählen Sie die für Ihr Betriebssystem geeignete Version.

- Unter Windows ist der empfohlene Download ein einfaches Installationsprogramm. Führen Sie ihn einfach aus.
- Unter Linux entpacken Sie einfach die heruntergeladene Zip-Datei in ein beliebiges Verzeichnis.
- Auf unserer Website finden Sie weitere detaillierte Anweisungen und Informationen zu anderen Plattformen.

Führen Sie nach der Installation den Editor der Castle Game Engine aus.

- Wenn du das Installationsprogramm unter Windows benutzt hast, wurde die Verknüpfung zum Ausführen von Castle Game Engine bereits für Sie erstellt worden.

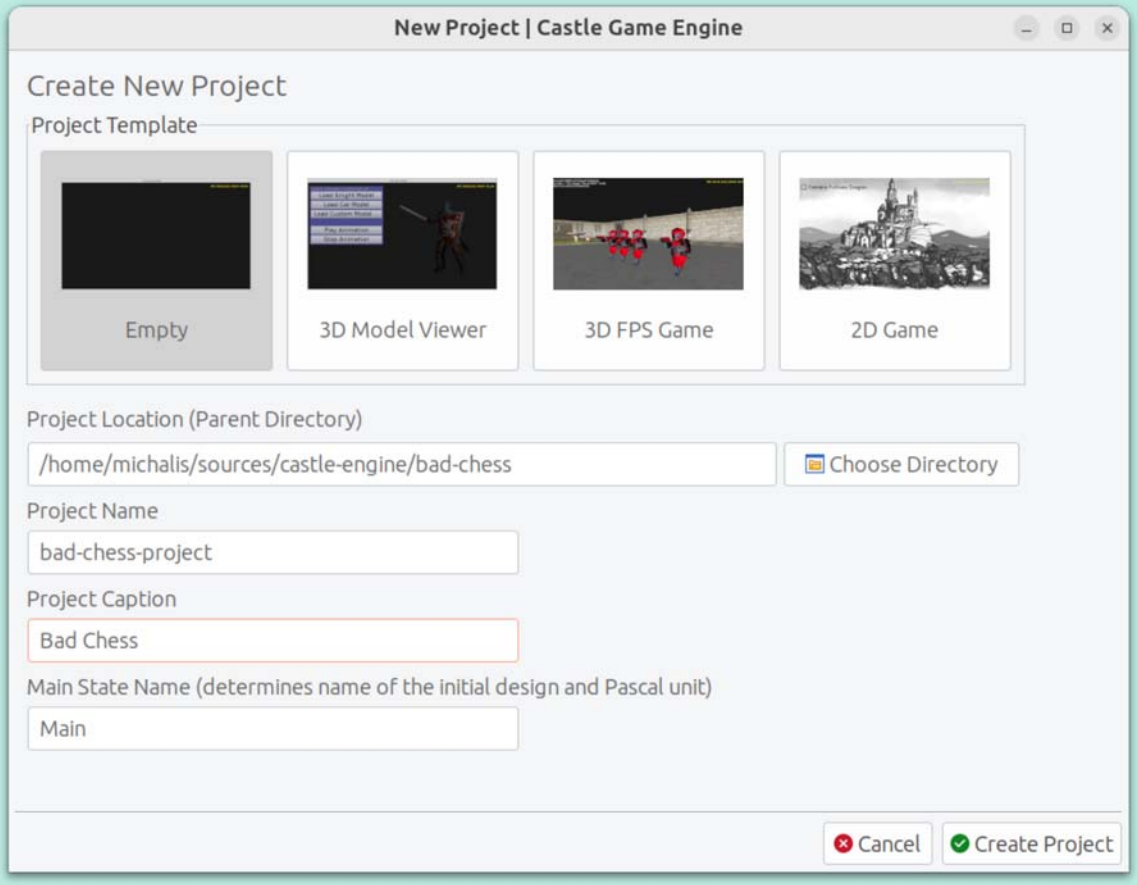
- Wenn Sie die Engine als Zip-Datei entpackt haben, führen Sie den binären Castle-Editor aus dem Unterverzeichnis bin, in das Sie die Engine entpackt haben.



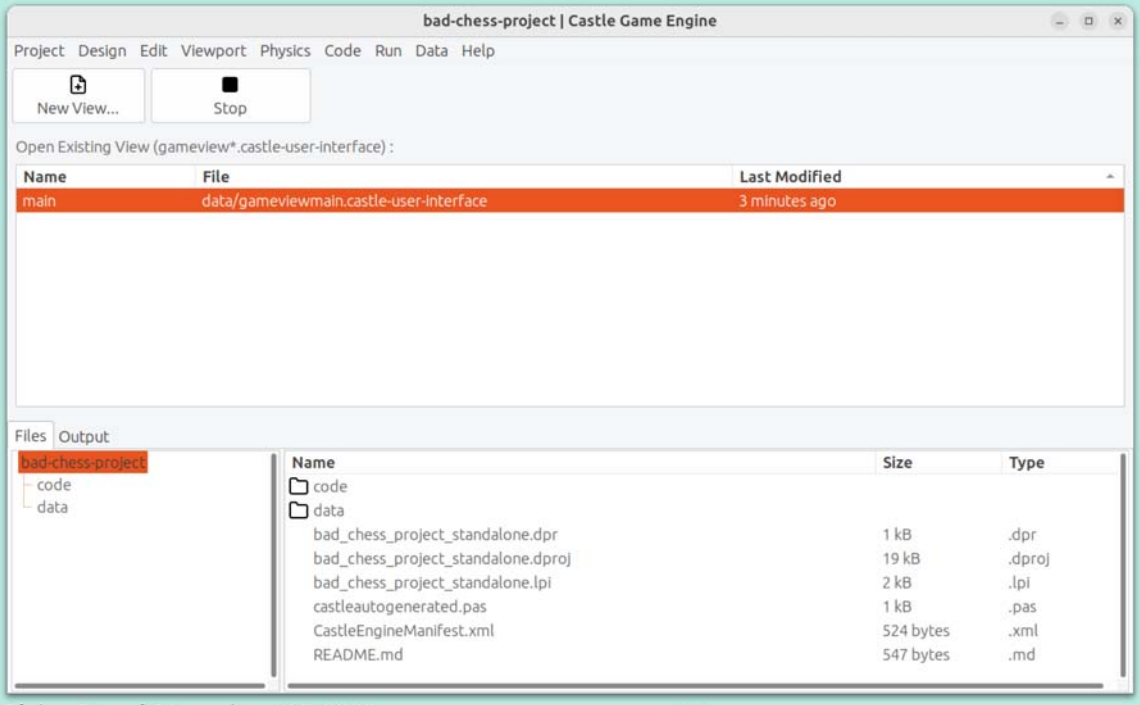
If you encounter any issue, consult our manual on https://castle-engine.io/install .

❹ IHR ERSTES PROJEKT ERSTELLEN

Lassen Sie uns ein neues Projekt erstellen. Klicken Sie auf die Schaltfläche "Neues Projekt", wählen Sie die Projektvorlage "Leer", konfigurieren Sie den Projektnamen und das Verzeichnis wie gewünscht und klicken Sie auf "Projekt erstellen".



Als Antwort erstellen wir ein neues Verzeichnis mit einigen Projektdateien, die Ihre Projektdaten und den anfänglichen Pascal-Code definieren.



❹ IHR ERSTES PROJEKT ERSTELLEN / FORTSETZUNG

Sie können die Dateien in Ihrem Projekt im unteren Bereich des Editors durchsuchen.
Sie können sie auch einfach mit Ihrem normalen Dateimanager durchsuchen - es gibt nichts Besonderes an diesem Verzeichnis, es sind normale Dateien und Verzeichnisse.

Die wichtigsten Dateien und Verzeichnisse sind:

- code ist ein Unterverzeichnis, in dem wir empfehlen, den gesamten Pascal-Quellcode (Units) Ihrer Anwendung. Anfänglich enthält es nur 2 Units, `GameInitialize` und `GameViewMain`.
- data ist ein Unterverzeichnis, in dem Sie alle Daten ablegen sollten, die zur Laufzeit von deiner Anwendung geladen werden müssen. Alle 3D- und 2D-Modelle, Texturen, Designs müssen
  müssen hier abgelegt werden, wenn Sie sie in Ihrem Spiel verwenden wollen.
  Zu Beginn enthält es das Design namens gameviewmain.castle-user-interface (*und, weniger wichtig, CastleSettings.xml und README.txt Dateien)*.

Die allgemeine Idee ist, dass die anfängliche Anwendung (die aus der Vorlage "Empty" erstellt wird) nur eine einzige Ansicht namens Main enthält. Eine Ansicht ist ein Castle Game Engine-Konzept, das etwas darstellt, das in einer Castle Game Engine-Anwendung angezeigt werden kann.
Sie verwenden es normalerweise ähnlich wie ein Formular in Delphi oder Lazarus. Es ist ein grundlegender Weg zur Organisation Ihre Anwendung

- Jede Ansicht kann visuell gestaltet werden. Doppelklicken Sie einfach auf die Ansicht im Fenster "Vorhandene Ansicht öffnen" oder im Fenster "Dateien"
  (*wenn Sie das Unterverzeichnis "Daten" durchsuchen*).
  Dies ermöglicht die visuelle Gestaltung des Inhalts der Datei **gameviewmain** `.castle-user-interface` Datei visuell zu gestalten. Die Datei hat eine Erweiterung `.castle-user-interface,` weil eine Ansicht ein Spezialfall einer Benutzeroberfläche in der Castle Game Engine ist.
  In größeren Anwendungen können Sie mehrere Ansichten haben. Außerdem kann man in größeren Anwendungen, können Sie visuell einige Benutzeroberflächenelemente entwerfen, die keine Ansichten sind, sondern nur wiederverwendbare Teile einer Benutzeroberfläche.
  Alle diese Dateien haben die Erweiterung `.castle-user-interface` und können mit dem Editor visuell gestaltet werden.
  Die Ansichten haben nach Konvention einen Namen wie gameview*.castle-user-interface.
- Jede Ansicht hat auch eine zugehörige Pascal-Unit. Die Unit hat den gleichen Namen wie die View, aber ohne die Erweiterung `.castle-user-interface`. In unserem Fall heißt die Unit also `gameviewmain.pas`. Die Unit enthält den Pascal-Code, der ausgeführt werden soll ausgeführt werden soll, wenn die Ansicht angezeigt wird. Sie definiert eine Klasse, die virtuelle Methoden hat, um auf verschiedene nützliche Ereignisse zu reagieren (*wie das Starten der Ansicht oder das Drücken einer Taste oder eine Maustaste*).
  Oft werden Sie weitere Methoden hinzufügen, um Ihre Anwendungslogik zu implementieren.

Siehe
`https://castle-engine.io/view_events` und
`https://castle-engine.io/views`,
um mehr über die Ansichten in unserer Engine zu erfahren.

Um die in unserer Engine verwendete Terminologie zu verdeutlichen:

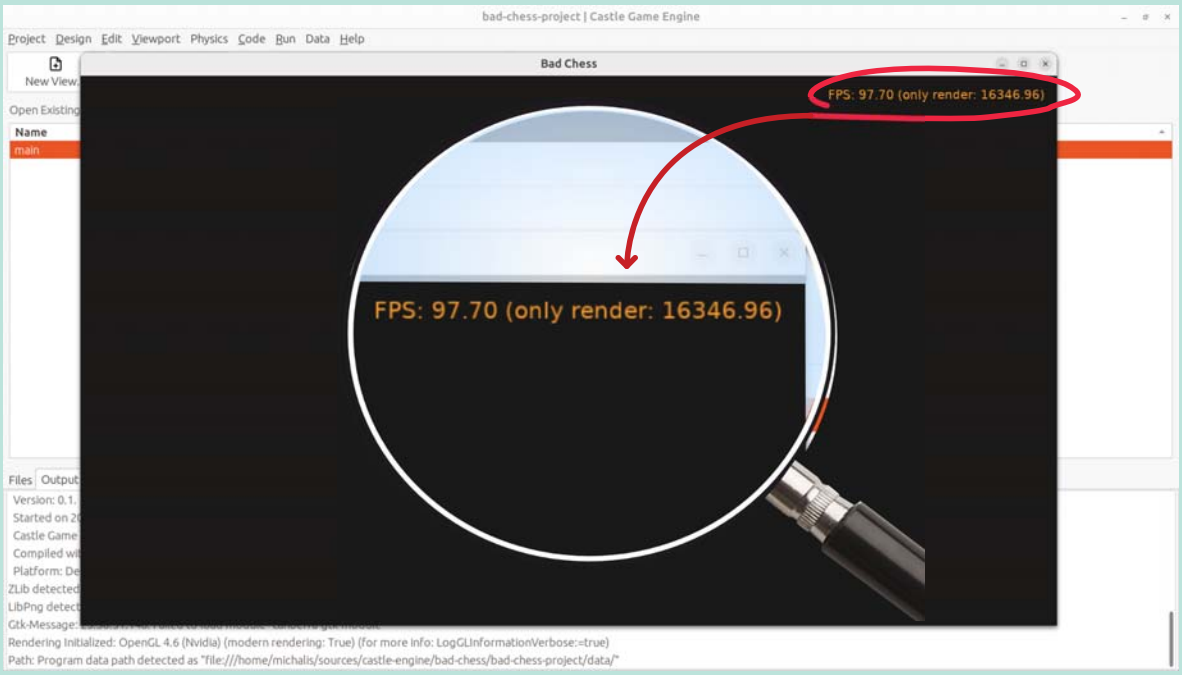
- Ein Entwurf ist ein Name für eine Datei, die Sie mit unserem Editor visuell gestalten können. Ein Entwurf kann eine Datei mit der Erweiterung:
  - **`.castle-user-interface`** (Benutzeroberfläche, kann in eine Klasse geladen werden, die von von `TCastleUserInterface`)
  - **`.castle-transform`** (*3D oder 2D Transformation, kann in eine Klasse geladen werden die von* `TCastleTransform` *abstammt*)
  - **`.castle-component`** (*jede andere Komponente; kann in eine Klasse geladen werden absteigend von* `TComponent`)

❹ IHR ERSTES PROJEKT ERSTELLEN / FORTSETZUNG

- Es handelt sich um eine Datei mit der Erweiterung .castle-user-interface.
- Eine Ansicht ist ein spezieller Fall eines Benutzeroberflächenentwurfs. Konventionell wird sie wie gameview*.castle-user-interface genannt.

Nach dieser ausführlichen Einführung brennt es Ihnen wahrscheinlich in den Fingern, endlich etwas zu tun. Fangen wir also an.

Als Erstes sollten Sie sicherstellen, dass alles funktioniert. Verwenden Sie die große Schaltfläche "Kompilieren und Ausführen" (**Tastenkürzel F9**) und beobachten Sie, wie das Projekt kompiliert und ausgeführt wird.
Das Ergebnis wird langweilig sein - ein dunkles Fenster mit einem FPS-Zähler (**Bilder pro Sekunde**) in der oberen rechten Ecke.
FPS sind eine Standardmethode, um die Leistung Ihrer Anwendung zu messen.



❺ OPTIONALES ANPASSEN DER EDITOREINSTELLUNGEN

Wenn alles funktioniert, können Sie die Einstellungen im Editor unter "**Preferences**" (*Einstellungen*) anpassen. Im Besonderen:

- Der Editor verwendet standardmäßig eine gebündelte Version des neuesten stabilen **FPC** (**Free Pascal Compiler**). Wenn Sie lieber Ihre eigene FPC-Installation oder **Delphi** verwenden möchten, konfigurieren Sie dies in den Einstellungen.
- Um die Pascal-Dateien zu bearbeiten, versucht der Editor standardmäßig, automatisch verschiedene Pascal-fähige IDEs und Editoren, wie Lazarus, Delphi, Visual Studio Code. Wenn Sie es vorziehen, einen bestimmten Editor zu konfigurieren, wählen Sie ihn in den Einstellungen.
  Mehr Details über die Editor-Konfiguration finden Sie in unserem Handbuch auf `https://castle-engine.io/install` .

Der Editor kann jeden Pascal-Compiler und jeden Texteditor verwenden. Wir stellen absichtlich keine besonderen Anforderungen an das, was Sie verwenden können. Allerdings stellen wir sicher, dass wir die populäre Auswahl perfekt unterstützen. Insbesondere haben wir eine spezielle Unterstützung für die Verwendung von **Visual Studio Code** mit **Pascal** *(und der Castle Game Engine im Besonderen)*, siehe `https://castle-engine.io/vscode` .

❻ LERNEN, WIE MAN 3D-OBJEKTE IN EINEM ANSICHTSFENSTER ENTWIRFT

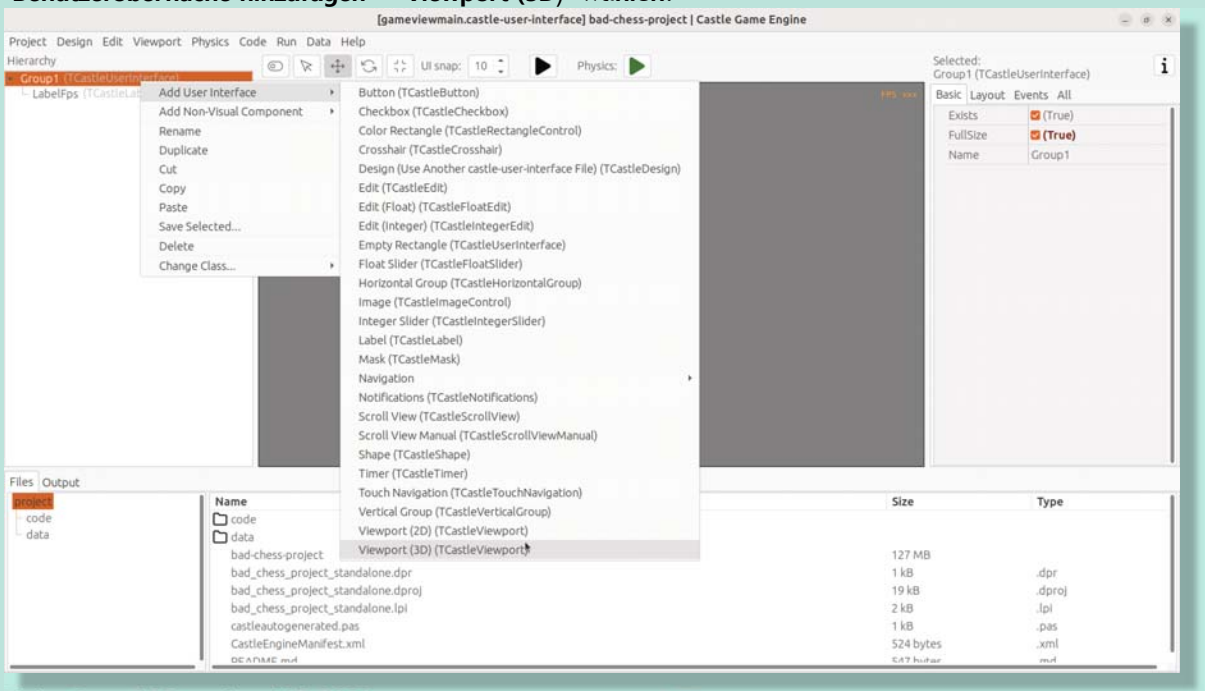
Wenn Sie es noch nicht getan haben, öffnen Sie die Hauptansicht im Editor.

Sie können im Fenster "Vorhandene Ansicht öffnen" oder im Fenster "Dateien" (*wenn Sie das Unterverzeichnis "Daten" durchsuchen*) auf die Ansicht doppelklicken.

Die ursprüngliche Ansicht ist größtenteils leer.
Sie hat eine Stammkomponente Group1, die eine Instanz von `TCastleUserInterface` ist. Diese Komponente wird alles andere enthalten, was wir entwerfen.

Und sie hat ein Label `LabelFps` (eine Instanz der Klasse `TCastleLabel`). Zur Laufzeit wird dieses Label den **FPS-Zähler** anzeigen.

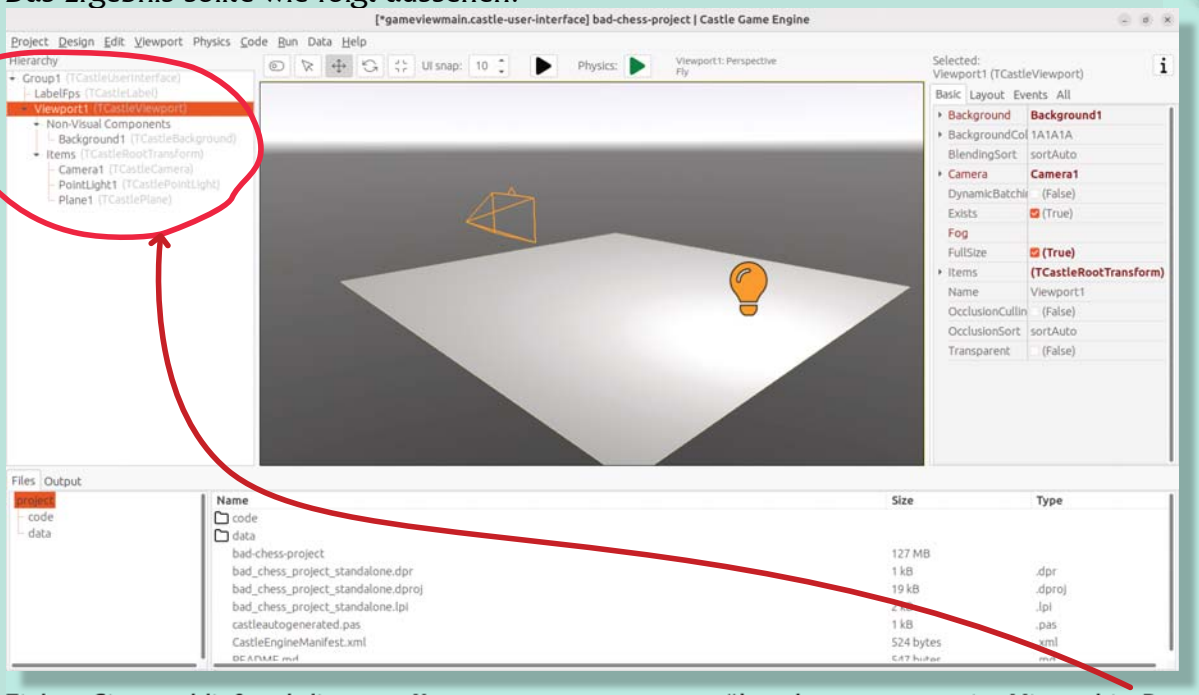
Fügen wir ihm mehr Inhalt hinzu. Um etwas in 3D darzustellen, benötigen Sie zunächst ein Ansichtsfenster. Ein Ansichtsfenster ist eine Möglichkeit, 3D- oder 2D-Inhalte anzuzeigen.
Es ist eine Instanz der Klasse `TCastleViewport`. Fügen Sie es dem Entwurf hinzu, indem Sie mit der rechten Maustaste auf die Komponente Group1 klicken und im angezeigten Menü "**Benutzeroberfläche hinzufügen → Viewport (3D)**" wählen.



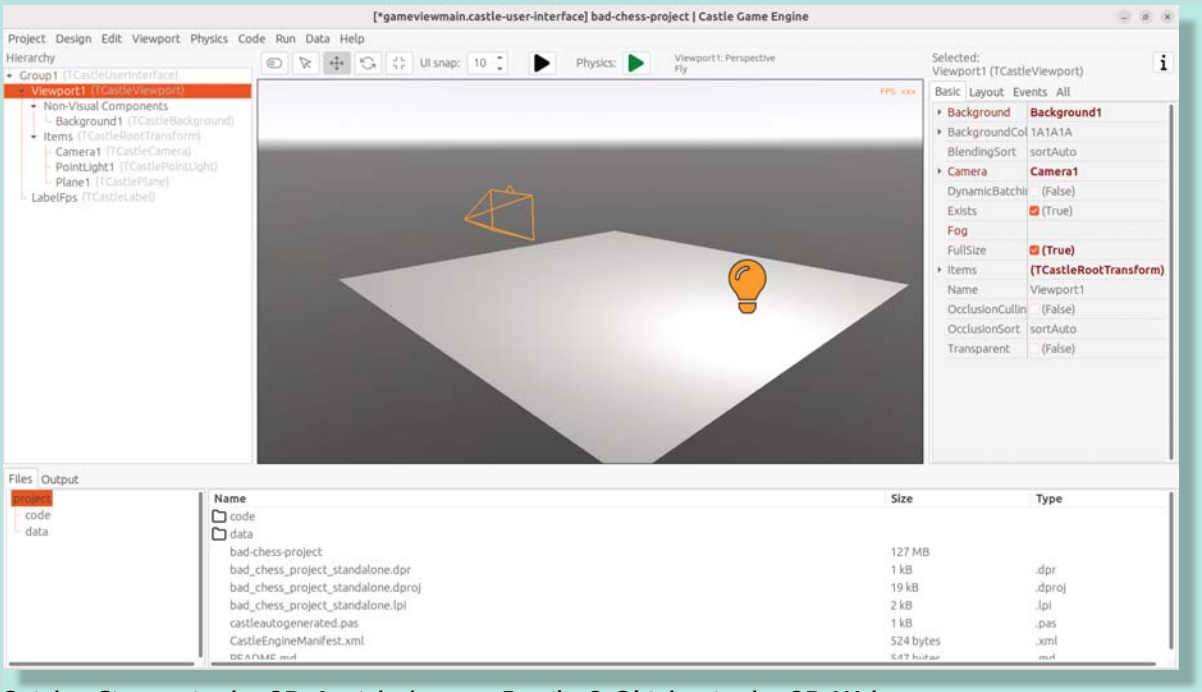
DIE SCHLECHTE ART, SCHACH ZU SPIELEN:
3D-PHYSIK-SPASS MIT DER CASTLE GAME ENGINE (TEIL 1)

❻ LERNEN, 3D-OBJEKTE IN EINEM ANSICHTSFENSTER ZU ENTWERFEN / FORTSETZUNG

Das Ergebnis sollte wie folgt aussehen:

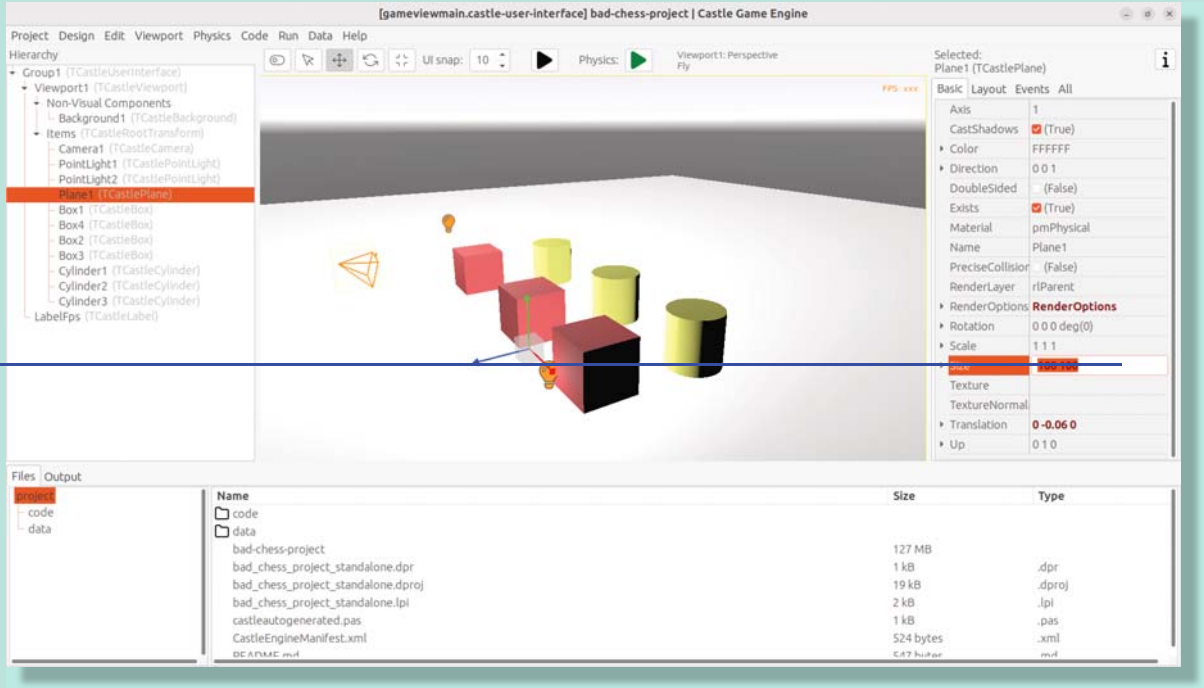


Ziehen Sie anschließend die neue Komponente `Viewport1` über das `LabelFps` im Hierarchie-Panel (*auf der linken Seite*). Auf diese Weise wird der FPS-Zähler vor dem Ansichtsfenster angezeigt.



Spielen Sie nun in der 3D-Ansicht herum. Es gibt 3 Objekte in der 3D-Welt:
Die Kamera, die nur `Camera1` genannt wird, bestimmt, was der Benutzer tatsächlich sieht, wenn das Spiel gestartet wird.
Die Lichtquelle sorgt dafür, dass die Dinge beleuchtet (*hell*) sind. Die anfängliche Lichtquelle heißt `PointLight1` und ist eine Instanz von `TCastlePointLight`, einem einfachen Licht, das von einer bestimmten 3D-Position aus in alle Richtungen leuchtet.
Ein Rechteck, das einen Boden namens `Plane1` darstellt.
Mathematisch gesehen handelt es sich nicht um eine Ebene, sondern um ein Rechteck - die Bezeichnung "Ebene" ist jedoch eine Konvention, die von vielen 3D-Programmen verwendet wird.

Klicken Sie mit der rechten Maustaste auf das Ansichtsfenster und halten Sie sie gedrückt, um sich umzusehen. Verwenden Sie die **AWSD-Tasten**, um sich zu bewegen.
Verwenden Sie den Mauszeiger (*während Sie die rechte Maustaste gedrückt halten*), um die Bewegungsgeschwindigkeit zu erhöhen oder zu verringern.
Spielen Sie mit der Bewegung der Objekte. Ziehen Sie die 3D-Achse, um ein beliebiges Objekt zu bewegen.Spielen Sie mit dem Hinzufügen neuer 3D-Elemente. Klicken Sie mit der rechten Maustaste auf die Komponente Items im **Viewport1** und fügen Sie aus dem Kontextmenü Primitive wie **"Box", "Sphere", "Cylinder"** hinzu. Verschieben Sie sie, löschen Sie sie (*mit der* **Entf-Taste**), duplizieren Sie sie (*mit der* **Strg+D-Taste**). Ändern Sie einige Eigenschaften. Auf der rechten Seite sehen Sie einen **Objektinspektor**, den jeder Lazarus- und Delphi-Benutzer kennt. Passen sie die Eigenschaften an, z.B. ändern sie die Größe von **Plane1** um viel größer zu sein. Klicken sie auf "..." (*3 Punkte, genannt Ellipsis*) bei der "**Farbe**" Eigenschaft eines beliebigen Primitivs (*wie eine Ebene, eine Box, eine Kugel...*) um die Farbe zu ändern.



Wenn Sie nicht weiterkommen, konsultieren Sie unser Handbuch, insbesondere
https://castle-engine.io/viewport_and_scenes und
https://castle-engine.io/viewport_3d sind hilfreich, um grundlegende 3D-Manipulationen zu erlernen.

7. EIN 3D-SCHACHBRETT MIT SCHACHFIGUREN ENTWERFEN

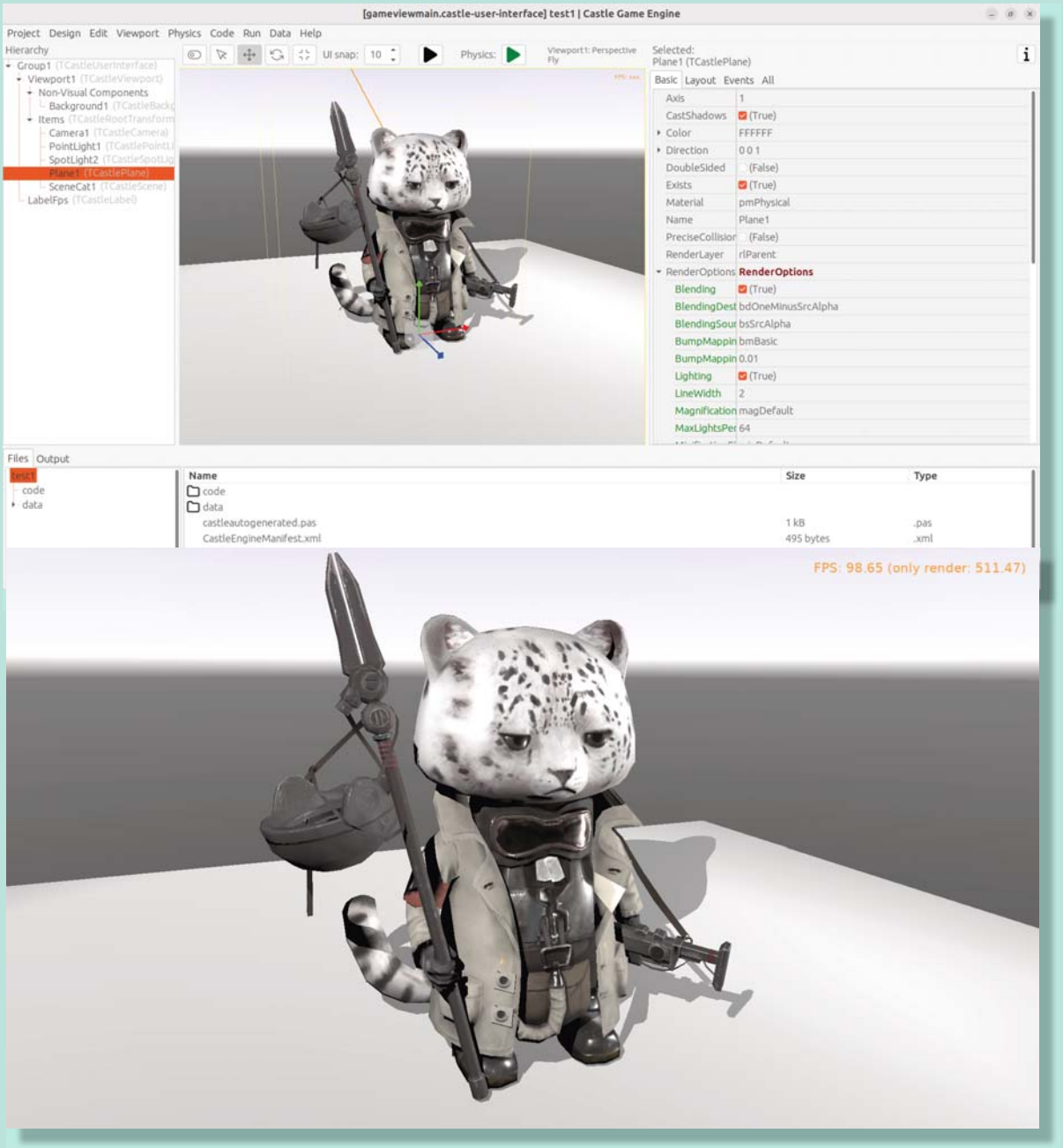
Oben haben wir gelernt, eine 3D-Welt zu entwerfen, die aus einfachen Primitiven wie Kisten und Kugeln besteht. Dies ist jedoch kein Weg, um realistische 3D-Grafiken zu erstellen.
In den meisten 3D-Grafikanwendungen wird der Inhalt mit einem speziellen **3D-Authoring-Tool** wie **Blender** erstellt. Der 3D-Künstler erstellt ein Netz (*eine Reihe von Eckpunkten, die miteinander verbunden sind, um Kanten und Polygone zu bilden*), weist Materialien und Texturen zu und exportiert das resultierende Objekt in eine Datei, die von einer Spiel-Engine gelesen werden kann - wie eine *glTF* (*1)-Datei.
Castle Game Engine bietet eine hervorragende Unterstützung für glTF.
Siehe **https://castle-engine.io/gltf**
Auf der Seite der **Castle Game Engine** ist unsere wichtigste Komponente zur Darstellung eines 3D-Modells **TCastleScene**. Es ist eine große Komponente, die eine zentrale Rolle in unserer Engine spielt (*auf die eine oder andere Weise ist sie sogar für das gesamte 3D- und 2D-Rendering in unserem Viewport verantwortlich*). Die Verwendung ist einfach: Sie erstellen eine Instanz von `TCastleScene` und setzen die **URL-Eigenschaft** so, dass sie auf das Modell zeigt, das Sie anzeigen möchten (*wie eine glTF-Datei*). Die `TCastleScene`-Klasse stammt von der `TCastleTransform`-Klasse ab, und als solche können Sie die `TCastleScene`-Instanzen bewegen, drehen und skalieren. Alternativ können Sie auch die `glTF-Datei` aus dem "Dateien"-Panel in das Ansichtsfenster ziehen. Der Editor erstellt dann automatisch eine `TCastleScene-Instanz`, die das angegebene Modell lädt.

DIE SCHLECHTE ART, SCHACH ZU SPIELEN:
3D-PHYSIK-SPASS MIT DER CASTLE GAME ENGINE (TEIL 1)

Wir unterstützen eine Reihe von 3D- und 2D-Modellformaten, nicht nur `glTF`. Sie sind auf **`https://castle-engine.io/creating_data_model_formats.php`** aufgelistet.
Wenn Sie in der Lage sind, Ihre eigenen 3D-Modelle zu erstellen, zum Beispiel in Blender, können Sie jetzt einen Umweg machen: Entwerfen Sie ein 3D-Modell in Blender und exportieren Sie es nach glTF, indem Sie unsere Anleitung auf **`https://castle-engine.io/blender`** verwenden.
Oder du kannst vorgefertigte Sachen verwenden:

- Es gibt eine Reihe von hochwertigen 3D-Inhalten im Internet, die auch kostenlos erhältlich sind und unter Open-Source-kompatiblen Lizenzen. Wir sammeln nützliche Links auf **`https://castle-engine.io/assets.php`** .
- Unsere Engine bietet auch eine Integration mit Sketchfab, um Ihnen die Suche und den aus einem riesigen Fundus kostenloser 3D-Modelle herunterladen können, ohne unseren Editor zu verlassen.
  Siehe die **`https://castle-engine.io/sketchfab`** Dokumentation.

Hier ist ein Beispiel - ein kampferprobtes Katzenmodell von Sketchfab, direkt in unserem Editor:



Credits: Das "Cat" 3D Modell wurde von Muru (https://sketchfab.com/muru) erstellt und ist auf Sketchfab (https://sketchfab.com/3d-models/cat-16c3444c8d1440fc97fdf10f60ec58b0) unter CC-BY-4.0 Lizenz verfügbar.

- Schließlich haben wir einen fertigen Satz von 3D-Modellen für das Schachbrett und alle Schachfiguren, die Sie für diese Demo verwenden können.

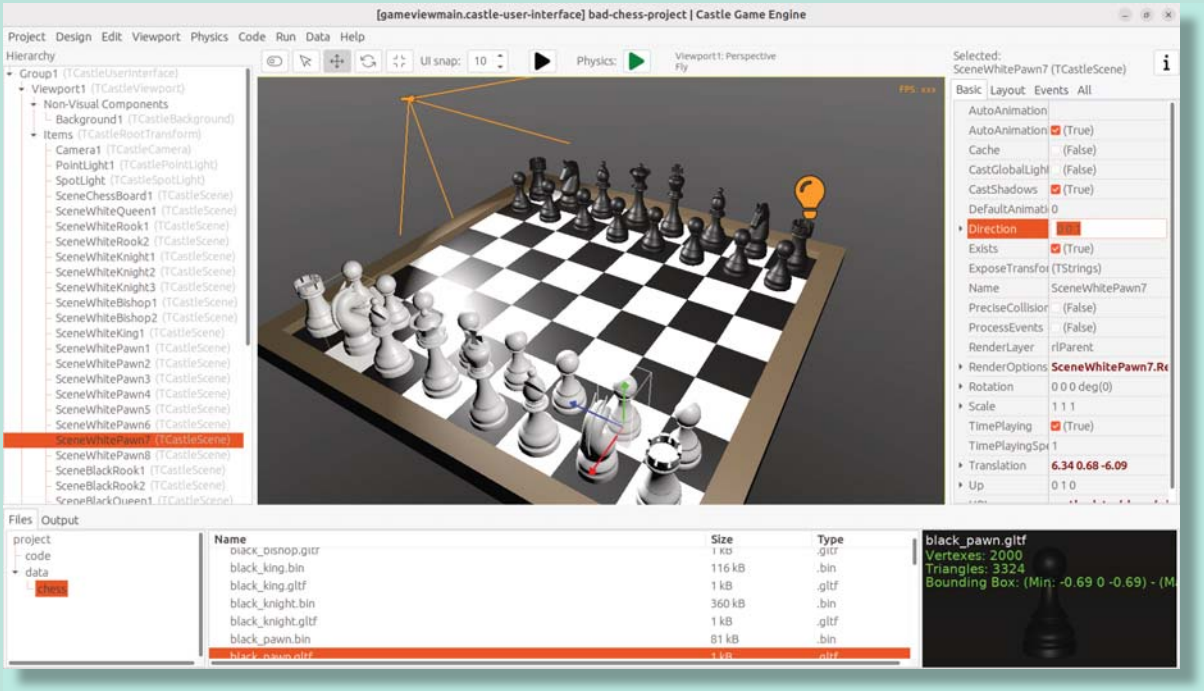
Um die letzte Option zu nutzen, laden Sie die 3D-Modelle von **https://github.com/castle-engine/bad-chess/releases/download/chess-models/chess-models.zip** herunter.
Sie wurden auf der Grundlage des Open-Source Blender-Modells erstellt, das auf **https://blendswap.com/blend/29244** von Phuong2647.
Entpacken Sie das resultierende Archiv irgendwo in das Unterverzeichnis data Ihres Projekts. Ziehen Sie dann einfach die **`*.gltf`**-Dateien auf das Ansichtsfenster. Verschieben und duplizieren Sie sie nach Bedarf, um sie in einer Schachstartposition anzuordnen.

HINWEIS
Für unser albernes Physikspiel ist es eigentlich völlig egal, wie du sie anordnen wirst. Sie müssen auch nicht perfekt positioniert und gedreht werden. Viel Spaß 😁

WIKIPEDIA

glTF ist ein Standarddateiformat für dreidimensionale Szenen und Modelle. Eine glTF-Datei verwendet eine von zwei möglichen Dateierweiterungen: .gltf (JSON/ASCII) oder .glb (binär). Sowohl .gltf- als auch .glb-Dateien können auf externe Binär- und Texturressourcen verweisen. Alternativ können beide Formate auch in sich geschlossen sein, indem sie Binärdatenpuffer direkt einbetten (als base64-kodierte Strings in .gltf-Dateien oder als rohe Byte-Arrays in .glb-Dateien).
Es handelt sich um einen offenen Standard, der von der Khronos Group entwickelt und gepflegt wird und 3D-Modellgeometrie, Erscheinungsbild, Szenengraphenhierarchie und Animation unterstützt. Es soll ein rationalisiertes, interoperables Format für die Bereitstellung von 3D-Assets sein und gleichzeitig die Dateigröße und Laufzeitverarbeitung durch Anwendungen minimieren. Seine Schöpfer haben es als das "JPEG von 3D" bezeichnet.

This is an example result:



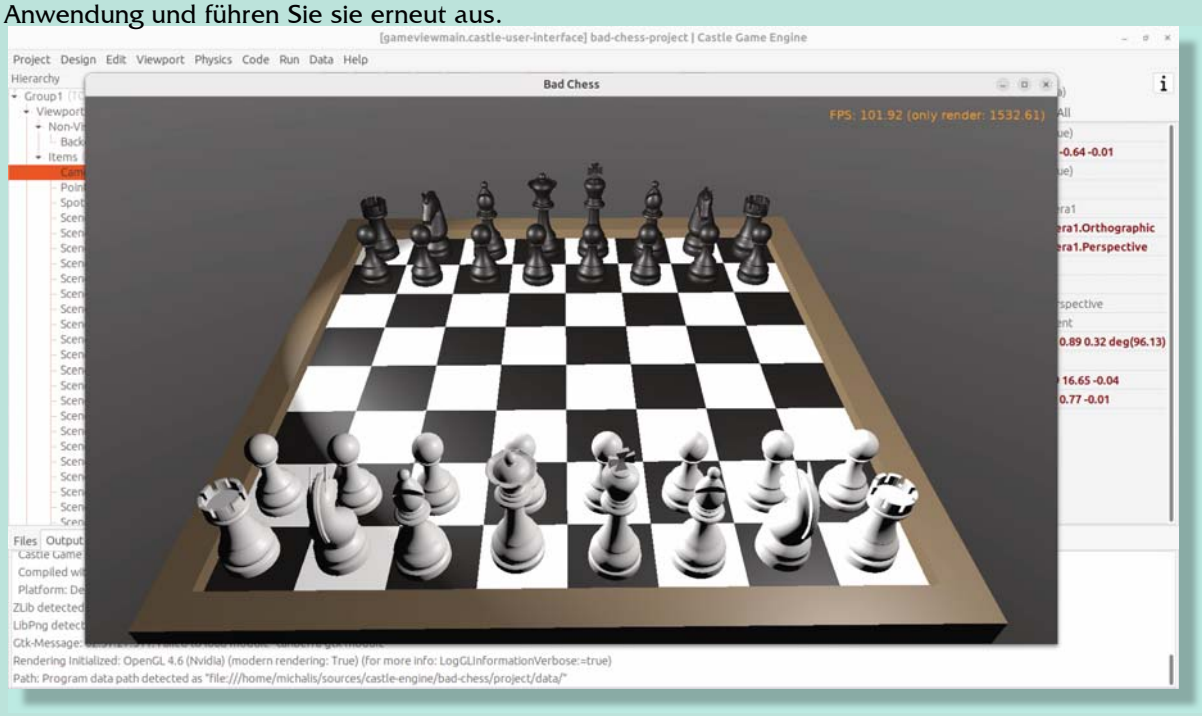
DIE SCHLECHTE ART, SCHACH ZU SPIELEN:
3D-PHYSIK-SPASS MIT DER CASTLE GAME ENGINE (TEIL 1)

Wenn Sie das Schachbrett entworfen und die Schachfiguren darauf platziert haben, stellen Sie sicher, dass Sie auch die Beleuchtung so einstellen, dass alles schön hell ist (*aber nicht zu hell*). Stellen Sie schließlich die Kamera so ein, dass der Benutzer beim Starten der Anwendung einen schönen Blick auf das Brett hat. Wenn Sie eine Kamerakomponente auswählen (*z. B. Kamera1, wenn Sie die Standardkamera nicht umbenannt haben*), zeigt der Editor ein kleines Fenster mit einer Kamera-vorschau an. Sie können in diesem Fenster auf "Pin" klicken, um die Welt von dieser Kamera aus zu beobachten.
Es gibt grundsätzlich 2 Möglichkeiten, die Kamera zu manipulieren:

❶ Bewege und drehe die Kamera wie jedes andere 3D-Objekt. Schauen Sie sich die Kamera-vorschau, um zu beurteilen, ob die Kameraansicht gut aussieht.

❷ Oder navigieren Sie im Editor und verwenden Sie dann den Menüpunkt "Ansichtsfenster → Kamera an Ansicht ausrichten" (**Tastenkürzel Strg + Numpad 0**), damit die Kamera Ansicht an die aktuelle Ansicht im Editor anzupassen.

Sobald Sie eine schöne Ansicht haben, stellen Sie sicher, dass alles funktioniert: Kompilieren Sie die Anwendung und führen Sie sie erneut aus.



DIE SCHLECHTE ART, SCHACH ZU SPIELEN:
3D-PHYSIK-SPASS MIT DER CASTLE GAME ENGINE (TEIL 1)

❽ VERWENDUNG DER PHYSIK IM EDITOR

Jetzt, wo das richtige Schachbrett mit Schachfiguren entworfen ist, können wir die Physik benutzen, um die Dinge verrückter zu machen.

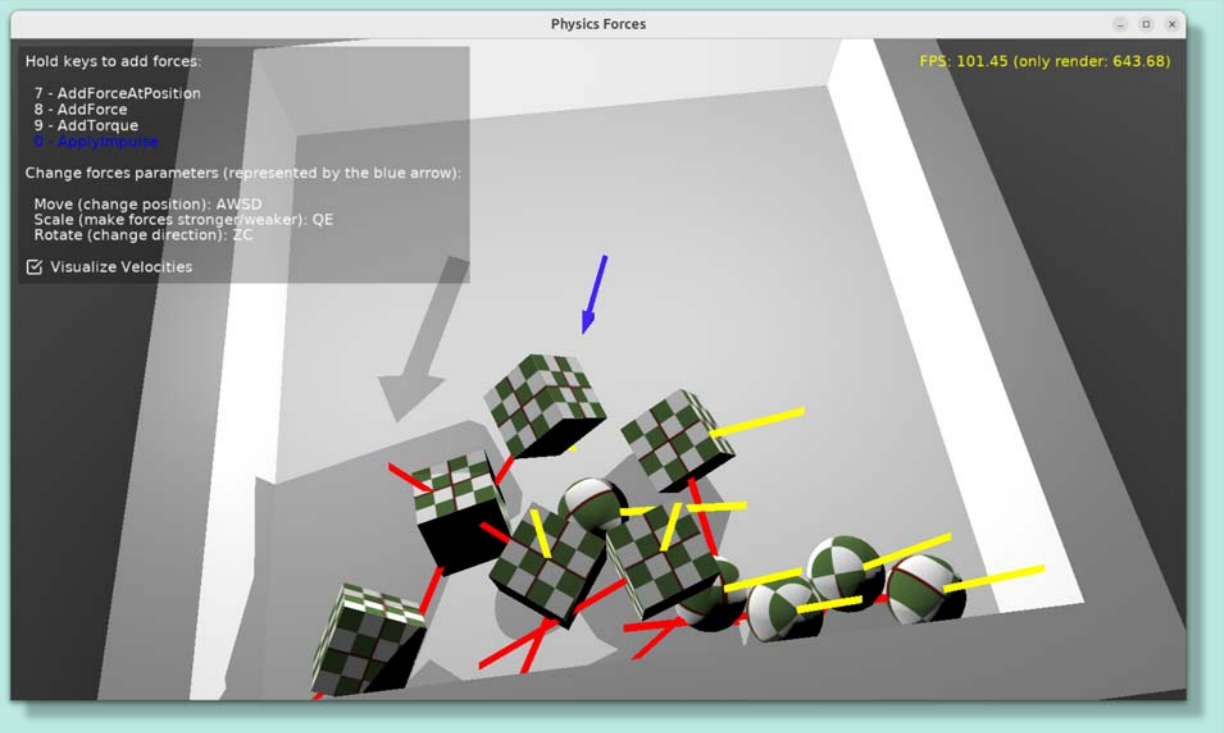
Die Castle Game Engine unterstützt Starrkörperphysik. Das bedeutet, dass:

- Objekte können von Kräften beeinflusst werden.
  Die Kraft, die automatisch wirkt, ist die Schwerkraft, die Objekte nach unten zieht *(in Richtung der negativen Y-Achse, standardmäßig)*.
  Sie können auch zusätzliche Kräfte aus dem Code definieren, um z.B. Dinge in eine beliebige Richtung zu schieben. Ihre eigenen Kräfte können eine Reihe von Effekten aus dem wirklichen Leben realisieren, wie Wind, Explosionen, wirbelnde Tornados, etc.

- Kollisionen zwischen Objekten werden automatisch erkannt und aufgelöst.
  Das heißt, standardmäßig prallen die Objekte aneinander ab.
  Es ist auch möglich, Kollisionen im Code zu erkennen und darauf zu reagieren *(z. B. kann ein Feind explodieren, wenn er mit einer Rakete zusammenstößt)*.

- Sie können bestimmte Objekte auch über Gelenke miteinander verbinden.

Wir werden in diesem Artikel nicht auf alle diese Funktionen eingehen, aber wir werden Ihnen zeigen, wie Sie die Grundlagen nutzen können.
Um mehr über die Möglichkeiten zu erfahren, lesen Sie unser Handbuch **`https://castle-engine.io/physics`** und spielen Sie mit den Demos im Unterverzeichnis `examples/physics/` der Engine. Hier ist ein Screenshot aus einer der Demos, der die explizite Anwendung von Physik-kräften zeigt:



DIE SCHLECHTE ART, SCHACH ZU SPIELEN:
3D-PHYSIK-SPASS MIT DER CASTLE GAME ENGINE (TEIL 1)

Die **Castle Game Engine** *Physik* verwendet intern Kraft, eine von *Benjamin 'BeRo' Rosseaux* in Pascal entwickelte **Physik-Engine**.

Jede Komponente, die von `TCastleTransform` abstammt, einschließlich Primitiven (*wie* `TCastleBox`) oder Szenen, die von Modellen geladen werden (`TCastleScene`) oder eine Gruppe anderer Objekte (`TCastleTransform` *mit Kindern*) kann ein starrer Körper für die `Physik-Engine` sein, der an der Kollisionserkennung und der daraus resultierenden Bewegung teilnimmt. Das Objekt muss zwei Verhaltensweisen haben:

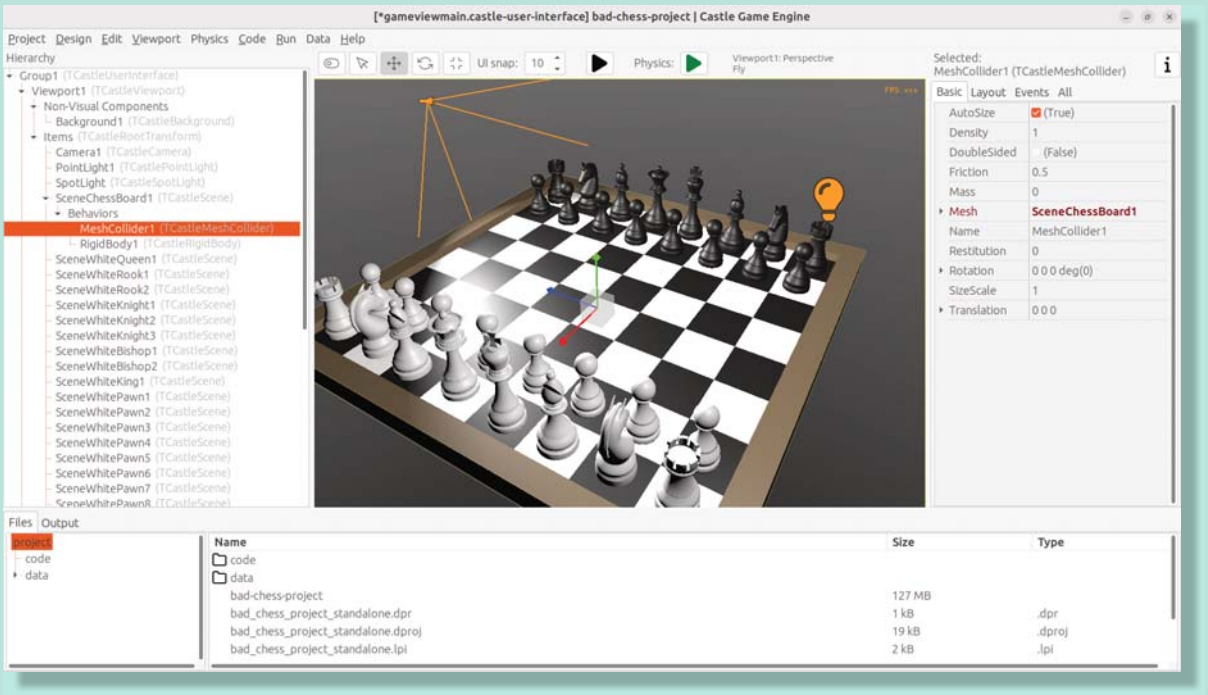
- Das Verhalten `TCastleRigidBody` macht die Komponente zu einem Starrkörper. Es definiert allgemeine Physikeigenschaften, z. B. ob das Objekt von der Schwerkraft beeinflusst wird und die anfängliche Bewegungsgeschwindigkeit.
- Ein Collider, der für jede Komponente steht, die von der abstrakten Klasse `TCastleCollider`. Viele Collider-Formen sind möglich, wie `TCastleSphereCollider`, `TCastleBoxCollider` und `TCastleMeshCollider`.

  Die Verwendung des `TCastleMeshCollider` führt zu den präzisesten Kollisionen, aber das kollidierende Objekt muss statisch sein, was bedeutet, dass andere Objekte an diesem Objekt abprallen werden, aber das Objekt mit `TCastleMeshCollider` wird sich selbst nicht bewegen.

Der Begriff Behavior, den wir oben verwendet haben, ist ein spezieller Mechanismus in der **Castle Game Engine**, um zusätzliche Funktionalität an eine `TCastleTransform` anzuhängen. Behaviors sind eine großartige Möglichkeit, verschiedene Funktionen zu definieren, die ein bestimmtes Spielobjekt verbessern. Es gibt verschiedene eingebaute Verhaltensweisen und Sie können auch Ihre eigenen definieren.
Siehe **`https://castle-engine.io/behaviors`** für weitere Informationen.

Nach diesem Überblick sind Sie bereit, die Physik in unserem Schachspiel zu verwenden.

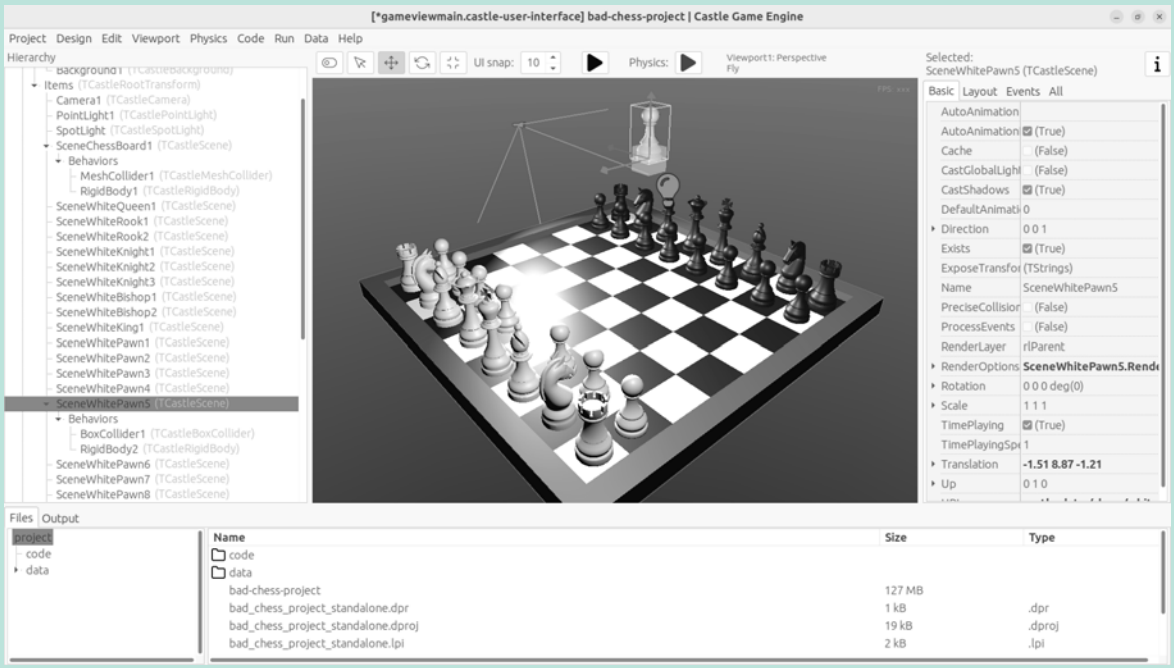
Klicken Sie mit der rechten Maustaste auf die Komponente, die das Schachbrett darstellt. Wählen Sie aus dem Kontextmenü "Verhalten hinzufügen (**Erweitert Elterntransformation) → Physik → Kollider → Netz**".
Als Antwort werden Sie feststellen, dass 2 Komponenten im Komponentenbaum erschienen sind: **MeshCollider1** und **RigidBody1**.
Das ist eine praktische Funktion des Editors: Das Hinzufügen eines Kolliders fügt auch eine Starrkörperkomponente hinzu.

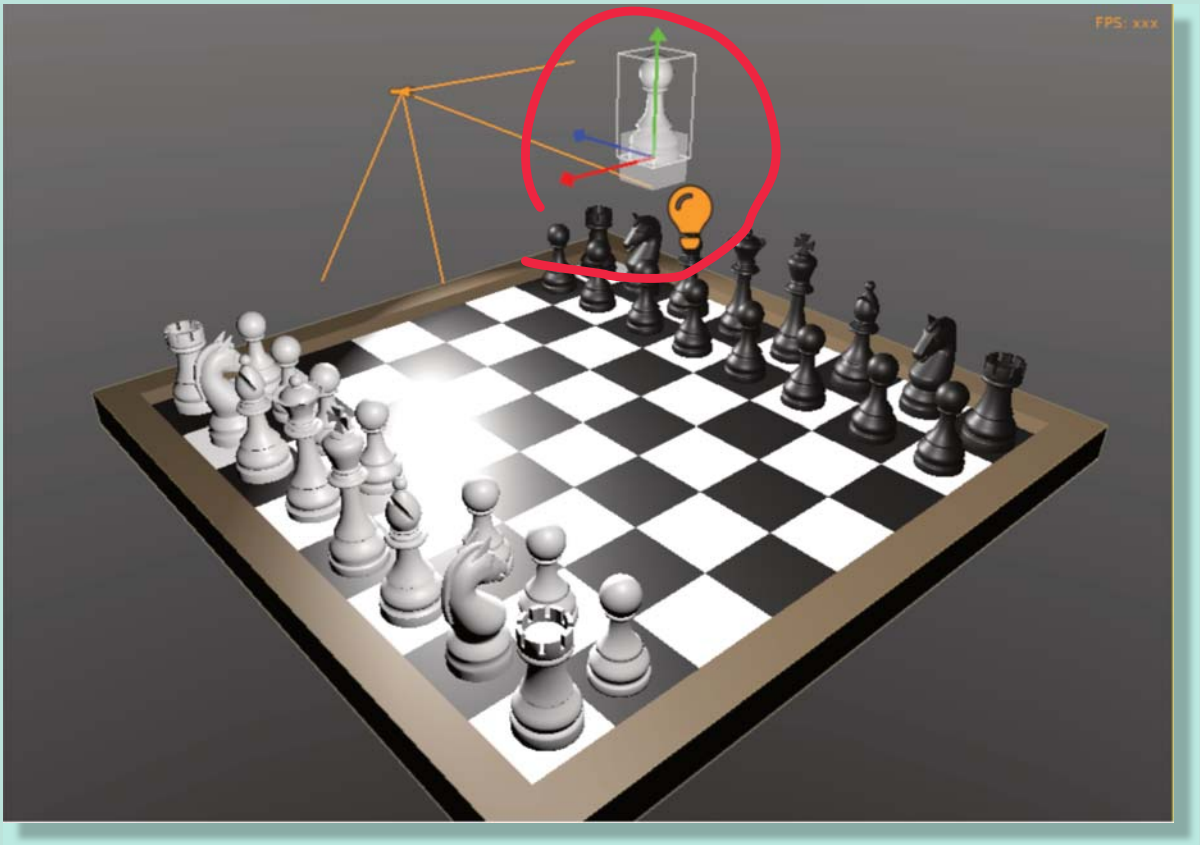


DIE SCHLECHTE ART, SCHACH ZU SPIELEN:
3D-PHYSIK-SPASS MIT DER CASTLE GAME ENGINE (TEIL 1)

Wählen Sie dann eine beliebige Schachfigur. Klicken Sie mit der rechten Maustaste darauf und wählen Sie aus dem Kontextmenü "Verhalten hinzufügen (*Erweitert übergeordnete Transformation*) → **Physik → Kollider → Box**". Beachten Sie, dass wir einen einfacheren Collider für die Schachfigur verwenden, der auch dynamisch ist. Dadurch kann die Schachfigur tatsächlich auf dem Brett herunterfallen.

Bewegen Sie die Schachfigur schließlich in eine dramatischere Position über dem Brett, so dass sie herunterfällt, wenn die Physik beginnt.





DIE SCHLECHTE ART, SCHACH ZU SPIELEN: 3D-PHYSIK-SPASS MIT DER CASTLE GAME ENGINE (TEIL 1)

Wir sind bereit für die Ausführung der Physik. Eine Möglichkeit wäre, die Anwendung einfach auszuführen, mit der Funktion "Kompilieren und Ausführen", wie Sie es bereits getan haben.

Aber es gibt einen schnelleren Weg, um mit der Physik zu experimentieren: Starten Sie die Physiksimulation mit dem grünen Play-Symbol in der Kopfzeile des Editors (*oder Menüpunkt* "**Physik → Simulation abspielen**", **Tastenkürzel Strg+P**).

Beobachten Sie, wie die Spielfigur auf das Brett fällt.

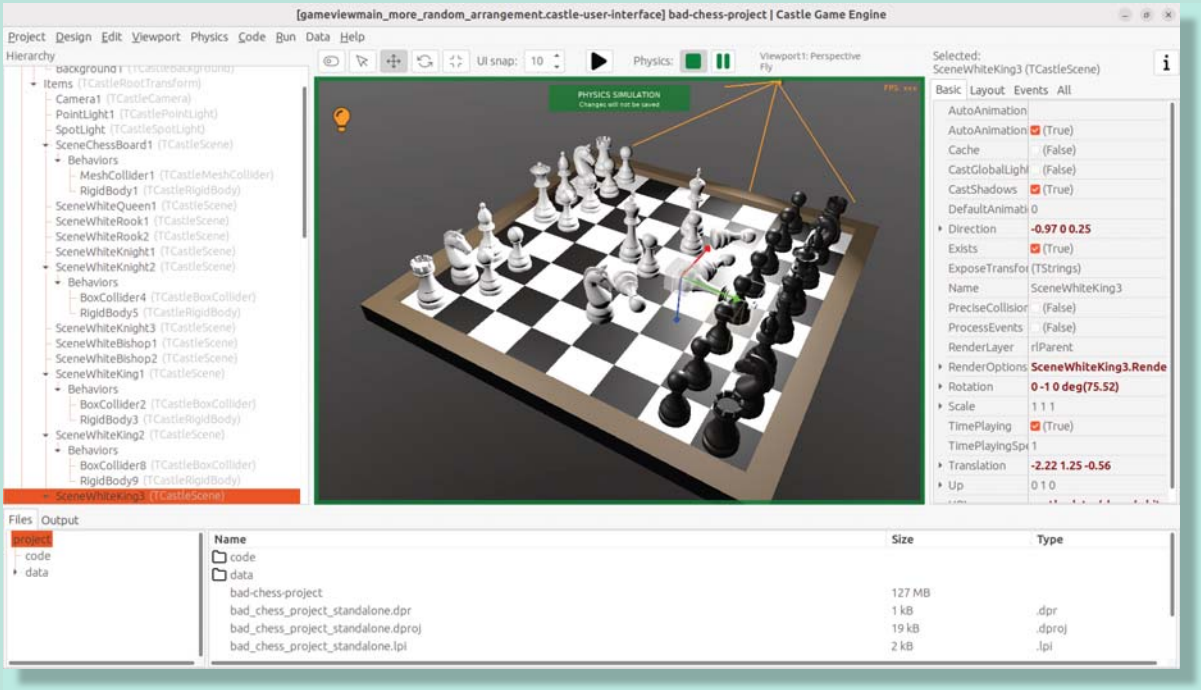
Vergessen Sie nicht, die Physiksimulation zu beenden, wenn Sie fertig sind (*drücken Sie die grüne Stopptaste, oder wieder den Menüpunkt* **"Physik → Simulation abspielen"**, **Tastenkürzel Strg+P**).

Die Bearbeitung des Entwurfs während der Physiksimulation ist erlaubt (*und eine gute Möglichkeit, mit verschiedenen Physikeinstellungen zu experimentieren*), aber die Änderungen werden nicht gespeichert, wenn die Physiksimulation läuft.
Das liegt daran, dass die Objekte in der Regel durch die Physik bewegt werden, und Sie wollen diese Position, die sich aus den physikalischen Interaktionen ergibt, nicht speichern.
Stellen Sie also sicher, dass Sie die Physiksimulation beenden, bevor Sie dauerhafte Änderungen am Entwurf vornehmen.
Für noch spektakulärere Ergebnisse:

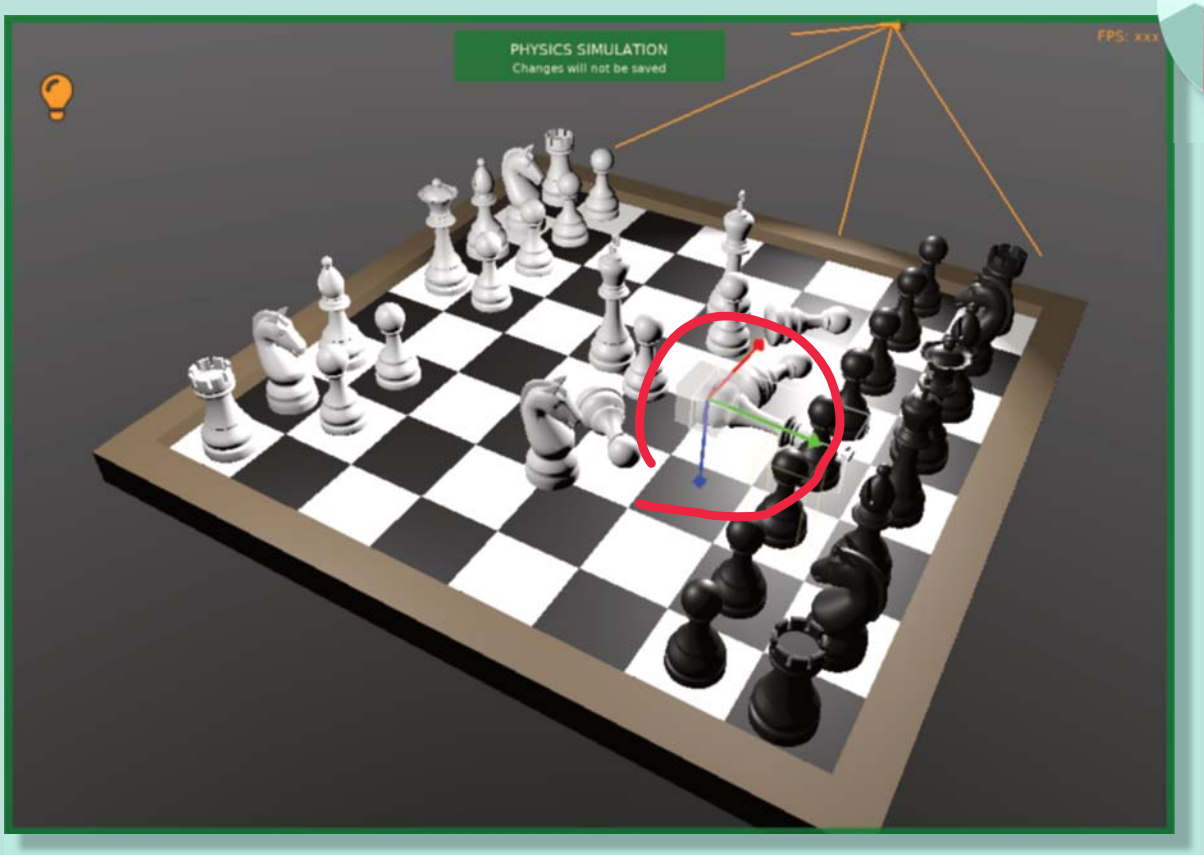
- Bewegen Sie die Schachfiguren in interessantere Positionen, so dass mehrere Figuren von oben auf mehrere andere Schachfiguren herunterfallen.
- Sie können die Schachfiguren auch duplizieren (**Tastenkürzel Strg+D**)
  (dabei wird das gesamte ausgewählte Objekt dupliziert, einschließlich des Physikverhaltens, falls vorhanden).
- Das ist ein einfacher Weg, um eine Menge physikalischer Objekte zu haben, die aneinander abprallen.

Nach jeder Änderung spielen Sie die Physiksimulation einfach ab und stoppen sie wieder.

Achten Sie darauf, dass die Ausgangsposition aller Starrkörper nicht dazu führt, dass ein Paar gleich zu Beginn miteinander kollidiert. Wenn die beiden Objekte beim Start kollidieren, kann es sein, dass die Physik-Engine sie (*manchmal ziemlich explosiv*) voneinander wegbewegt.
Dies ist ein Beispielergebnis:



DIE SCHLECHTE ART, SCHACH ZU SPIELEN:
3D-PHYSIK-SPASS MIT DER CASTLE GAME ENGINE (TEIL 1)



Eine letzte Sache bleibt in diesem (*ersten*) Teil des Artikels zu lernen:
Wie schnippt man die Schachfigur?

❶ In Pascal-Code können Sie verschiedene Methoden verwenden,
um eine Kraft auf einen starren Körper auszuüben.
Mehr dazu im nächsten Teil des Artikels.
Sie können auch mit der Beispielanwendung experimentieren
`examples/physics/physics_forces/` experimentieren, wenn Sie ungeduldig sind.

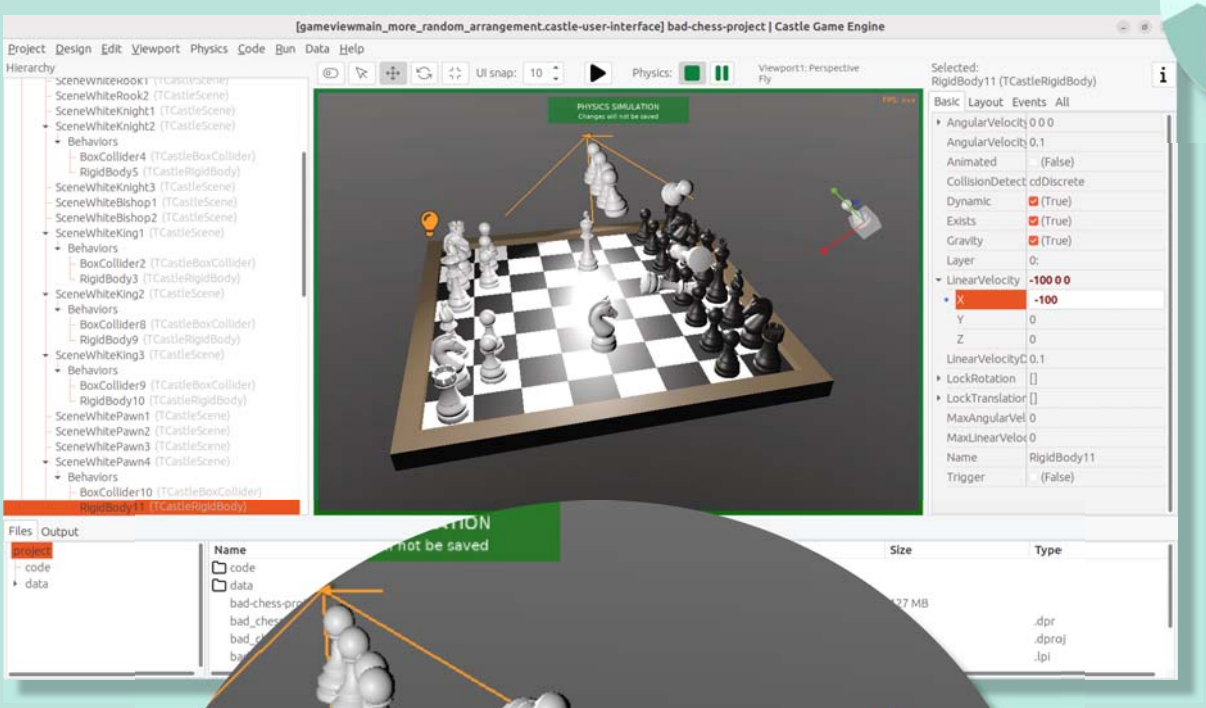
❷ Sie können auch eine bestimmte **LinearVelocity** für eine **Starrkörperkomponente** festlegen.

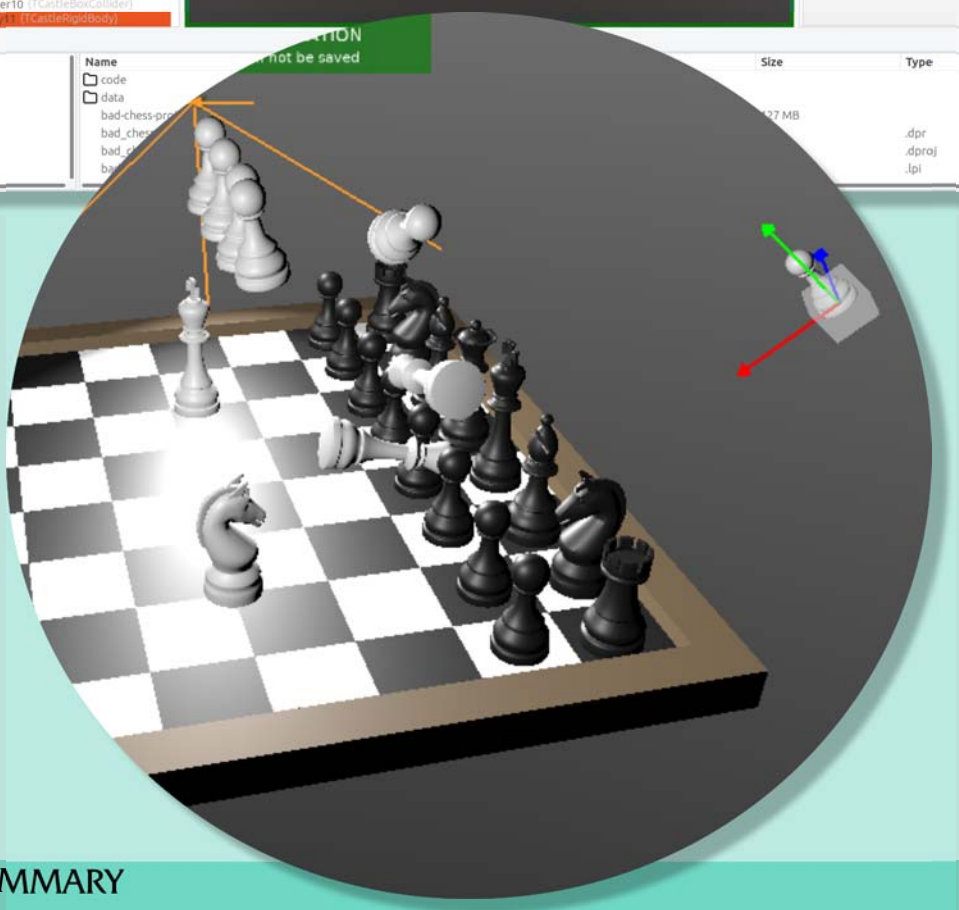
Wir werden den letzteren Ansatz verwenden, da er trivialerweise im Editor durchgeführt und getestet werden kann.

- Wählen Sie die Schachfigur aus. Jede Schachfigur, die Sie "schnippen" wollen (*über das Brett werfen*).
- Vergewissern Sie sich, dass sie einen Kollider und **Starrkörperkomponenten** hat (*falls nicht, fügen Sie sie wie oben beschrieben hinzu*).
- Wählen Sie die `TCastleRigidBody-Komponente` der Figur aus, und suchen Sie darin die Eigenschaft `LinearVelocity`.
- Setzen Sie `LinearVelocity` auf einen großen Vektor, der nicht Null ist, z. B. -100 0 0. Dies bedeutet, dass wir eine Geschwindigkeit von 100 Einheiten pro Sekunde in der negativen X-Richtung.

Starten Sie die Physiksimulation und beobachten Sie das Chaos.

DIE SCHLECHTE ART, SCHACH ZU SPIELEN:
3D-PHYSIK-SPASS MIT DER CASTLE GAME ENGINE (TEIL 1)





❾ SUMMARY

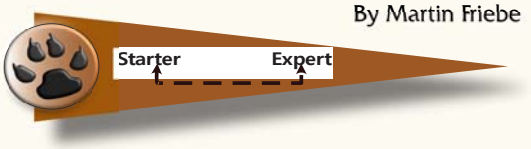
Wir haben eine 3D-Anwendung mit der **Castle Game Engine** und ein wenig Physik entwickelt.
Wir haben noch keinen Pascal-Code für die Interaktionen geschrieben - das werden wir im nächsten Teil des Artikels nachholen.

Wenn Sie eine fertige Anwendung herunterladen möchten, gehen Sie zu
`https://github.com/castle-engine/bad-chess`.
Das Unterverzeichnis project dieses Repositories enthält die fertige Arbeitsdemo.
Sie wird im nächsten Teil des Artikels noch erweitert werden.

Ich hoffe, Sie hatten Spaß bei dieser Demo und beim Erkunden der Möglichkeiten der **Castle Game Engine**.
Wenn ihr Fragen oder Feedback zur Engine habt, seid nicht schüchtern!
Sprechen Sie uns an, fragen Sie und teilen Sie Ihre Kommentare in unserem Forum
`https://forum.castle-engine.io` oder auf Discord
`https://castle-engine.io/talk.php`.

DER LAZARUS DEBUGGER

By Martin Friebe



TEIL 3: AUFRUF - DER STAPEL

WOHER WIR KOMMEN

Im letzten Teil der Reihe haben wir gelernt, wie wir in Funktionen ein- und aussteigen können. Wir haben uns angesehen, was in der äußeren Funktion passiert, und dann nach weiteren Details gesucht, indem wir in eine andere Funktion eingestiegen sind.
Dieses Mal werden wir unseren Blickwinkel erweitern.
Wir werden eine Funktion debuggen, aber anstatt herauszugehen, um zu sehen, was in der aufrufenden Funktion passiert, werden wir den Debugger dazu bringen, uns die Daten der aufrufenden Funktion zu zeigen, während wir noch in der aktuellen Funktion sind. Wie üblich erforschen wir all dies, indem wir ein kleines Beispielprojekt debuggen.

```
1. program FindRepeat;
2. uses Math;
3.
4. const
5.   TESTDATA = 'Test a random text. Repeat: a random text';
6.
7.
8. function EqualSubText(AText: Ansistring; AStart1, AStart2, AMaxLen: Integer): AnsiString;
9. var
10.   EqualLen: Integer;
11. begin
12.   EqualLen := 0;
13.   while (AMaxLen > EqualLen) and (AText[AStart1 + EqualLen] = AText[AStart2 + EqualLen]) do
14.     inc(EqualLen);
15.
16.   Result := copy(AText, AStart1, EqualLen);
17. end;
18.
19. (* DoFindLongestRepeat
20.    AStart1: Iterates over all potential start positions for the first match of the text
21.    AMaxSearchLen1: Count of chars up to the start of for the second match
22.    AStart2: Iterates over all potential start positions for the second match of the text
23.           Must always be greater the AStart1
24.                  (or equal, which will be handled by "AMaxSearchLen1 = 0")
25.     AMaxSearchLen2: Count of chars up to the end of the string
26.  *)
27.  function DoFindLongestRepeat(AText, AFound: Ansistring;
28.    AStart1, AMaxSearchLen1,
29.    AStart2, AMaxSearchLen2: Integer
30.   ): Ansistring;
31. var
32.   EqualTxt: String;
33. begin
34.   Result := AFound;
35.
36.   EqualTxt := EqualSubText(AText, AStart1, AStart2, Min(AMaxSearchLen1, AMaxSearchLen2));
37.   if Length(EqualTxt) > Length(Result) then
38.     Result := EqualTxt;
39.
40.   if AMaxSearchLen2 > 1 then
41.     Result := DoFindLongestRepeat(AText, Result,
42.       // AStart2 increases, so there is one more char available after AStart1
43.       AStart1,    AMaxSearchLen1 + 1,
44.       // And there is one char less after AStart2
45.       AStart2 + 1, AMaxSearchLen2 - 1
46.     )
47.   else
48.   if AStart1 < Length(AText) - 1 then begin
49.     Result := DoFindLongestRepeat(AText, Result,
50.       // AStart2 is set equal to AStart1, so AMaxSearchLen1 will be 0
```

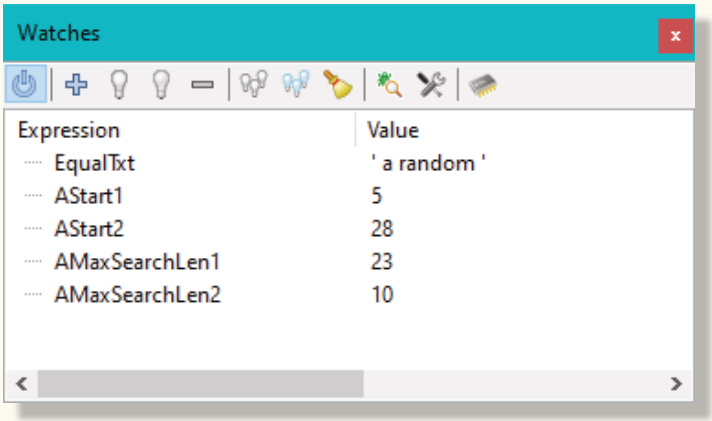
WEITERFÜHRUNG

```
51.     AStart1 + 1, 0,
52.     // AStart2 will be 1 more than AStart1 was,
53.     // so there will be 1 char less to search after AStart2
54.     AStart1 + 1, AMaxSearchLen1 - 1
55.   );   56.  end;
57. end;
58.
59. function FindLongestRepeat(AText: Ansistring): Ansistring;
60. begin
61.  Result := DoFindLongestRepeat(AText, '',
62.    1, 0,         // AStart1 equals AStart2: There are 0 chars between
63.    1, Length(AText)  // AStart2 has the entire string
64.  );
65. end;
66.
67. begin
68.  writeLn('"' + FindLongestRepeat(TESTDATA) + '"');
69.  readln;
70. end.
```

Der Code ist ein rekursives Beispiel dafür, wie man die längste, sich nicht überschneidende, wiederkehrende Teilzeichenkette findet. Es durchläuft alle Kombinationen von zwei Startpunkten *("AStart1" und "AStart2"*) und sucht für jeden dieser Punkte nach einer passenden Teilzeichenkette.
Für die gegebenen Testdaten erwarten wir das Ergebnis: **"ein zufälliger Text"** (*mit führendem Leerzeichen*).
Wenn wir es ausführen, wird es Folgendes ausgeben
" **einen zufälligen** "

Das letzte Wort "**Text**" wird irgendwie übersehen.

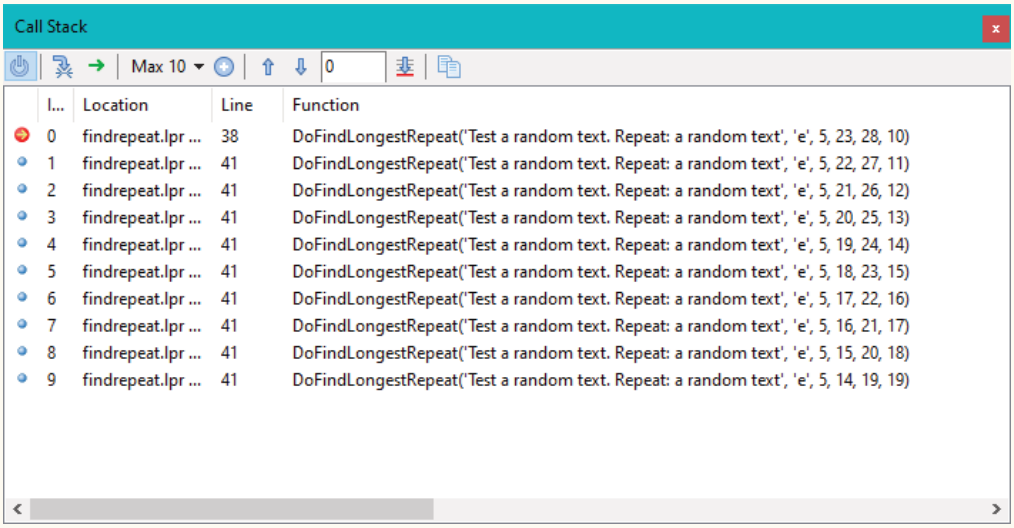
Wie zuvor beginnen wir unsere Debug-Sitzung, indem wir zu einem Haltepunkt laufen. Zeile 38 könnte ein guter Kandidat für den Anfang sein. Jedes Mal, wenn ein mögliches Ergebnis gefunden wird, wird eine Pause eingelegt.
Wenn also das Teilergebnis "**a random**" zugewiesen wird, können wir uns die verfügbaren Daten ansehen und prüfen, ob sie irgendwelche Anhaltspunkte liefern.
Nachdem wir das Projekt mit **F9** gestartet und den Haltepunkt gedrückt haben, sehen wir uns den Wert von "**EqualTxt**" an. Wir können dies im Fenster "**Locals**" oder "**Watches**" tun und sehen, dass er "e" lautet.
Da dies nicht die Übereinstimmung ist, an der wir interessiert sind, führen wir (**F9**) erneut aus.
Da wir den Haltepunkt zum 2. Mal treffen, wird "**EqualTxt**" als "**a random**" angezeigt. Dies ist der Wert, der fälschlicherweise als die längste wiederholte Übereinstimmung ausgegeben wurde. Wir werden uns die Werte ansehen, die am Aufruf von "**EqualSubText**" beteiligt sind.



Die Werte für "**AStart1**" und "**AStart2**" sehen korrekt aus.
Eine Überprüfung der 2 Maximallängen "**AMaxSearchLen2**" könnte sich lohnen. 10 ist nur die Länge bis zum Ende des zurückgegebenen Wertes in "**EqualTxt**". Wir wissen aber, dass die "TESTDATA" danach mehr Text enthält. Der Wert sollte also größer sein.
Wir können überprüfen, dass "**TESTDATA**" 41 Zeichen hat. Wenn also "**AStart2**" bei 28 liegt, sollten noch 14 Zeichen übrig sein. 41 - 28 + 1 ("+ 1", *da das letzte Zeichen einbezogen werden muss*).
"**AMaxSearchLen2**" wurde von der aufrufenden Funktion als Parameter übergeben.
Um mehr herauszufinden, müssen wir wissen, was im Aufrufer passiert ist.

DAS AUFRUFSTAPEL-FENSTER

Wie in der Einleitung angekündigt, kann uns der Debugger dabei helfen.
Mit **Strg-Alt-S** oder dem Menü "**Ansicht** → **Debug-Fenster** → **Aufrufstapel**" wird das Aufrufstapel-Fenster geöffnet.



Bevor wir die Verfolgung des falschen Wertes fortsetzen, sehen wir uns den Inhalt des neuen Fensters an und welche Informationen es liefert.
Die oberste Zeile ist die Zeile 38 (*Spalte "Zeile"*), in der unsere Anwendung derzeit pausiert.
Die Zeile darunter zeigt, von wo aus der aktuelle Aufruf von "**DoFindLongestRepeat**" erfolgte. Da wir uns in einer Rekursion befinden, hat sich die Funktion selbst aufgerufen. Der Aufruf erfolgte jedoch ab Zeile 41.
Betrachten Sie alle Spalten im Raster.

- Die erste Spalte zeigt an, ob die Zeile einen Haltepunkt hat. In unserem Fall gilt dies für Zeile 38. Hätten wir aber einen Haltepunkt in Zeile 41, würde er auch in den anderen Zeilen angezeigt werden.
- Die 2. Spalte "I..." (Index) ist eine laufende Nummer, die uns zeigt, wie viele Aufrufe wir von der Spitze entfernt sind. Dies kann nützlich sein, wenn man durch eine sehr lange Liste blättert.
- "Ort" und "Zeile" sind die Einheit (Dateiname) und die Zeilennummer. Wenn sie nicht bekannt sind, kann eine Adresse angezeigt werden.
- "Funktion" ist der Name der Routine. Im Falle einer Methode wird er in der Datei "Klassenname.Methode" notiert. Diese Spalte enthält auch die Werte der der übergebenen Parameter. (*Bitte beachten Sie den Hinweis zu params und locals im Abschnitt* **"Der vollständige Stack"**)

Wir werden später noch auf weitere Einzelheiten eingehen.

WERTE VON DEN ÄUSSEREN AUFRUFERN

Bevor wir alle Funktionen des Aufrufstapels untersuchen, werden wir die aktuelle Ansicht verwenden, um den Wert von "**AMaxSearchLen2**" zu verfolgen.
Die Argumente für den Aufruf in jeder Zeile werden in der gleichen Reihenfolge angegeben, wie die Parameter im Quelltext deklariert sind. So ist "**AMaxSearchLen2**" der letzte Wert in der Liste.

```
, 'e', 5, 23, 28, 10)
, 'e', 5, 22, 27, 11)
```

Für die oberste Zeile - die die Funktion darstellt, in der das Projekt gerade pausiert - haben wir "**5, 23, 28, 10**" für "**AStart1, AMaxSearchLen1, AStart2, AMaxSearchLen2**". Wie wir also gesehen haben, ist "**AMaxSearchLen2**"=**10**.

Und für den direkten Aufrufer haben wir "**5, 22, 27, 11**". Der Aufrufer hatte an der Position ein Zeichen zuvor (**"AStart2"=27**) nach wiederholtem Text gesucht und hatte bis zu 11 Zeichen ("**AMaxSearchLen2**") zu prüfen. Der Vergleich der Gesamtsumme des Aufrufers von "**27 + 11**" mit dem aktuellen Wert "**28 + 10**", sind beide Funktionen gleich weit von der tatsächlichen Länge des Textes entfernt.

Wenn man sich den Stapel in jeder Zeile ansieht, geht "**AStart2**" eine Zeile nach unten, und "AMaxSearchLen2" um eins nach oben. Wir sehen jedoch nur die ersten 10 Aufrufer, und die relevanten Informationen liegen möglicherweise weiter entfernt. Wir können mehr Zeilen erhalten, wenn wir auf den Button drücken oder die Dropdown-Liste "**Max 10**" verwenden.
Verwenden wir die "**Max 10**"-Dropdown und wählen Sie 50 Einträge.
(Wenn wir mehr brauchen, müssen wir auf den Button klicken)

Wir blättern nach unten, bis wir eine Unterbrechung im Muster von +/-1 finden.
Dies geschieht bei dem Aufrufer aus Zeile 49.

Call Stack

Max 50 | 0

I...	Location	Line	Function
22	findrepeat.lpr ...	41	DoFindLongestRepeat('Test a random text. Repeat: a random text', 'e', 5, 1, 6, 32)
23	findrepeat.lpr ...	41	DoFindLongestRepeat('Test a random text. Repeat: a random text', 'e', 5, 0, 5, 33)
24	findrepeat.lpr ...	49	DoFindLongestRepeat('Test a random text. Repeat: a random text', 'e', 4, 34, 38, 1)
25	findrepeat.lpr ...	41	DoFindLongestRepeat('Test a random text. Repeat: a random text', 'e', 4, 33, 37, 2)
26	findrepeat.lpr ...	41	DoFindLongestRepeat('Test a random text. Repeat: a random text', 'e', 4, 32, 36, 3)

Ein Blick auf den Index 23, der von 24 aufgerufen wird, zeigt, dass "AStart1" auf 5 erhöht wurde (wo das erste Vorkommen von "a random text" beginnt),
und "**AStart2**" wurde ebenfalls auf 5 gesetzt. Bei der Überprüfung von Index 24 stellen wir fest, dass "**AStart2**"+"**AMaxSearchLen2**" = "**38 + 1**" = 39.
Das ist nicht die volle Länge, aber 1 mehr als "**27 + 11**" = 38.
Bei diesem Aufruf haben wir also 1 Zeichen von "**AMaxSearchLen2**" verloren.

Ein Blick auf den Code

```
Result := DoFindLongestRepeat(AText, Result,
  AStart1 + 1, 0,
  // AStart2 will be 1 more than AStart1 was,
  // so there will be 1 char less to search after AStart2
  AStart1 + 1, AMaxSearchLen1 - 1
);
```

Der Code geht davon aus, dass "**AStart1** + **AMaxSearchLen1**" (*ohne das passende +/-1*) die gesamte Zeichenkette von "**AStart1**" bis zum Ende der Zeichenkette abdeckt, so dass die Werte für "**AStart2**" und "**AMaxSearchLen2**" verwendet werden können.
Tatsächlich haben sie aber nur die Zeichenkette bis vor "**AStart2**" abgedeckt, also bis zum letzten Zeichen der Zeichenkette. Daher ist "**AMaxSearchLen1**" bereits um eins kürzer als die verbleibende Länge der Zeichenkette. Es besteht keine Notwendigkeit, 1 abzuziehen. Der korrekte Code sollte lauten:

```
Result := DoFindLongestRepeat(AText, Result,
   // AStart2 is set equal to AStart1, so AMaxSearchLen1 will be 0
   AStart1 + 1, 0,
   // AStart2 will be 1 more than AStart1 was,
   // so there will be 1 char less to search after AStart2
   AStart1 + 1, AMaxSearchLen1 // no "-1"
  );
```

Eine erneute Ausführung des Projekts mit dieser Änderung führt zu dem erwarteten
" a random text""

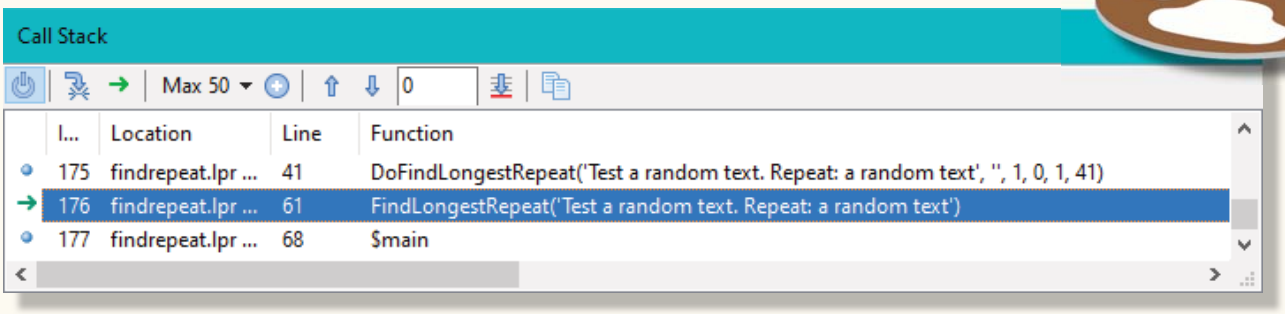
DER VOLLSTÄNDIGE STACK
Nachdem wir nun das Problem im Beispielprojekt gelöst haben, sollten wir uns etwas mehr Zeit nehmen, um das Stack-Fenster zu erkunden. Wir sollten uns auch ein paar Namen ansehen, die in diesem Zusammenhang häufig verwendet werden.
Der Stack selbst hat seinen Namen von der gleichnamigen Datenstruktur, die auch als "LiFo Stack" bekannt ist. Dies ist eine Funktion, die viele CPUs haben. Wenn eine Funktion aufgerufen wird, speichert (schiebt) die CPU die aktuelle Ausführungsadresse auf den Stack, so dass sie sie später abrufen kann, um die Ausführung im Code des Aufrufers fortzusetzen.
Viele Aufrufe können verschachtelt werden, und wenn sie zurückkehren, werden die Adressen in umgekehrter Reihenfolge abgerufen (Last-in, First-out).
Der Stack enthält oft mehr als nur die Adresse für die Rückkehr. Er enthält auch die Werte von lokalen Variablen. Dieser Speicher auf dem Stack wird als "**Stackframe**" bezeichnet...

Der Name "Frame" oder "Stack-Frame" wird häufig verwendet, um einen Eintrag im Stack-Fenster zu bezeichnen.

Während der obigen Debug-Übung haben wir gesehen, dass die Werte der Funktionsparameter für jeden Frame im Stack aufgelistet sind. Der Rahmen enthält auch lokale Variablen.
Im Stack-Fenster können wir jeden Frame als "aktuell" auswählen.
Dazu wählen wir den Rahmen mit der Maus oder der Tastatur aus und klicken dann auf die grüne Pfeiltaste: ➔
Sobald ein Frame "aktuell" ist, zeigen andere Debug-Fenster (*wie Watches und Locals*) ihren Inhalt entsprechend diesem Frame an.
Wenn wir den Rahmen bei Index 1 aktuell machen würden, dann würde Locals die Werte für diesen Rahmen anzeigen, also anstatt "**AStart2"=28** für den obersten Rahmen, würde es **"AStart2"=27** anzeigen.

HINWEIS: Die Parameter und lokalen Variablen eines Frames werden mit ihren aktuellen Werten angezeigt. Oft ist das der Wert, den sie zum Zeitpunkt des Aufrufs hatten. Wenn jedoch die aufgerufene Funktion den lokalen Wert geändert hat, wird dieser geänderte Wert angezeigt.

Wenn wir genügend Frames anzeigen, können wir andere Aufrufer als "**DoFindLongesRepeat**" sehen



Wenn Sie "FindLongestRepeat" zum aktuellen Frame machen, können Sie sehen, dass das lokale Fenster nicht mehr die Variablen `AStart.../AMaxSearchLen` anzeigt. Stattdessen wird "**AText**" angezeigt, der Parameter, der an "**FindLongestRepeat**" übergeben wird. Wir sehen auch, dass wir den ganzen Weg zurückverfolgen können bis zum Programm "`begin...end`."-Block zurückverfolgen können, der als "**$main**" angezeigt wird. Und der Index sagt uns, dass unsere Rekursion zum Zeitpunkt des Auftreffens des Haltepunkts 175 Aufrufe tief ist.

Manchmal sind die Stack Traces viel tiefer als das, und in diesem Fall kann es mühsam sein, den Button zu benutzen, um das Ende des Stacks zu erreichen. In diesem Fall können die blauen **Auf-/Ab-Buttons** helfen, schnell nach oben und unten zu navigieren. Und das Editierfeld mit dem Button kann verwendet werden, um einen beliebigen Index einzugeben und die Bilder ab diesem Index anzuzeigen.
Apropos Navigation: Das Stack-Fenster ermöglicht auch das Navigieren im Quellcode. Ein Doppelklick auf eine beliebige Zeile im Stack (*oder über den Button*) bringt den Code in den Quelltext-Editor.
Natürlich nur, wenn der Stackframe eine Quelldatei und eine Zeilennummer hat.

Der Button kopiert alle Einträge in die Zwischenablage. Und der Power-Button friert die aktuell angezeigten Einträge ein. Wenn die Stromversorgung ausgeschaltet ist, wird das Stack-Fenster nicht aktualisiert, wenn Sie die Anwendung starten.
Für den Fall, dass Sie die aktuelle Frame-Liste als Erinnerung oder ähnliches behalten wollen.

ZUSAMMENFASSUNG
Der Aufrufstapel kann verwendet werden, um die Locals eines beliebigen Aufrufers zu untersuchen. Er kann uns auch zeigen, wer die aktuelle Funktion aufgerufen hat.

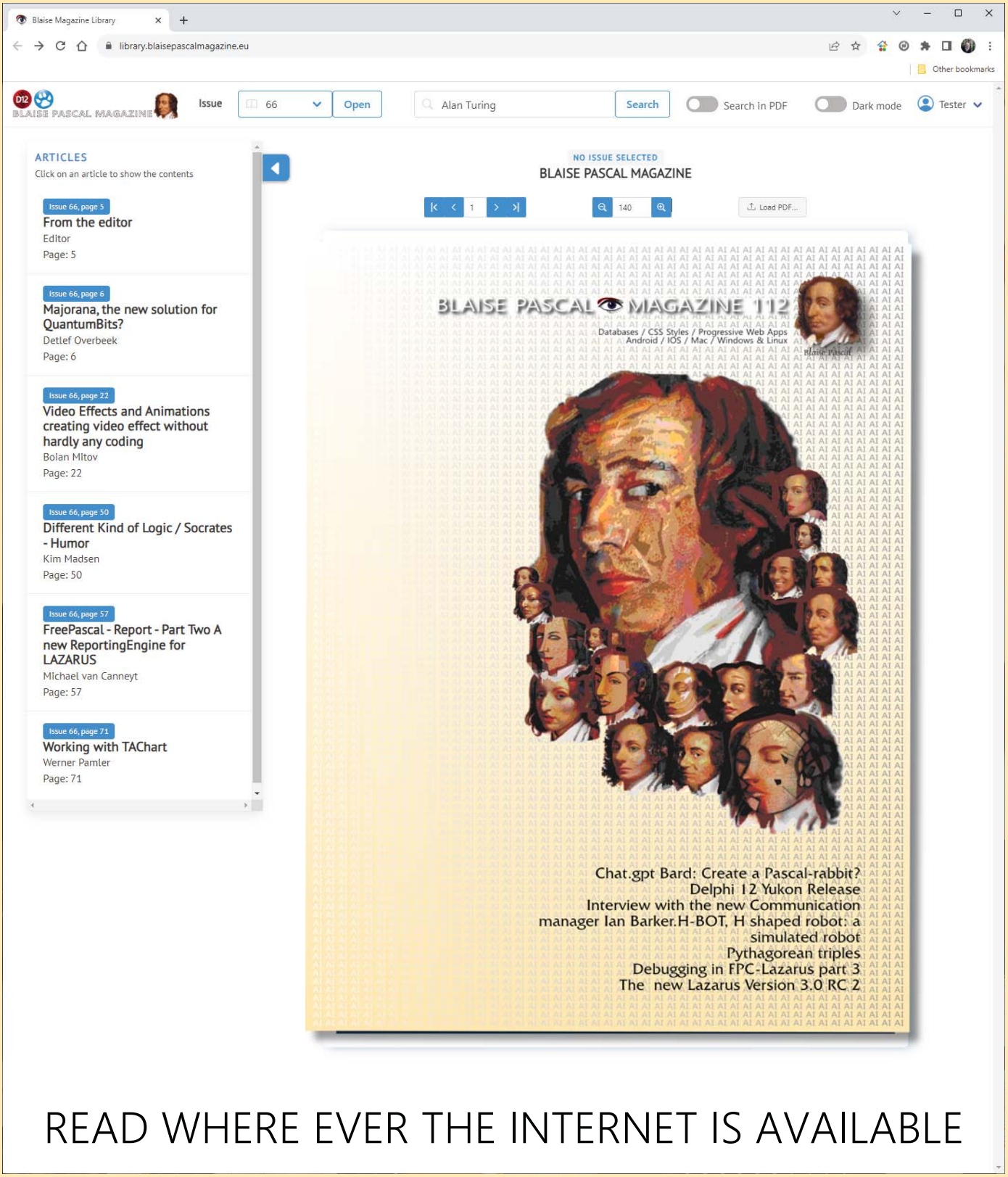
- Öffnen Sie das Stack-Fenster:
  - Strg-Alt-S
  - Menü: Ansicht → Debug-Fenster → Stack aufrufen
- Wählen Sie einen Frame als "aktuell" aus
  -
- Anzahl der angezeigten Frames erhöhen
  -
  - "Max 10" Drop-Down

Im nächsten Artikel: Teil 4: SCHAUEN - WATCHES

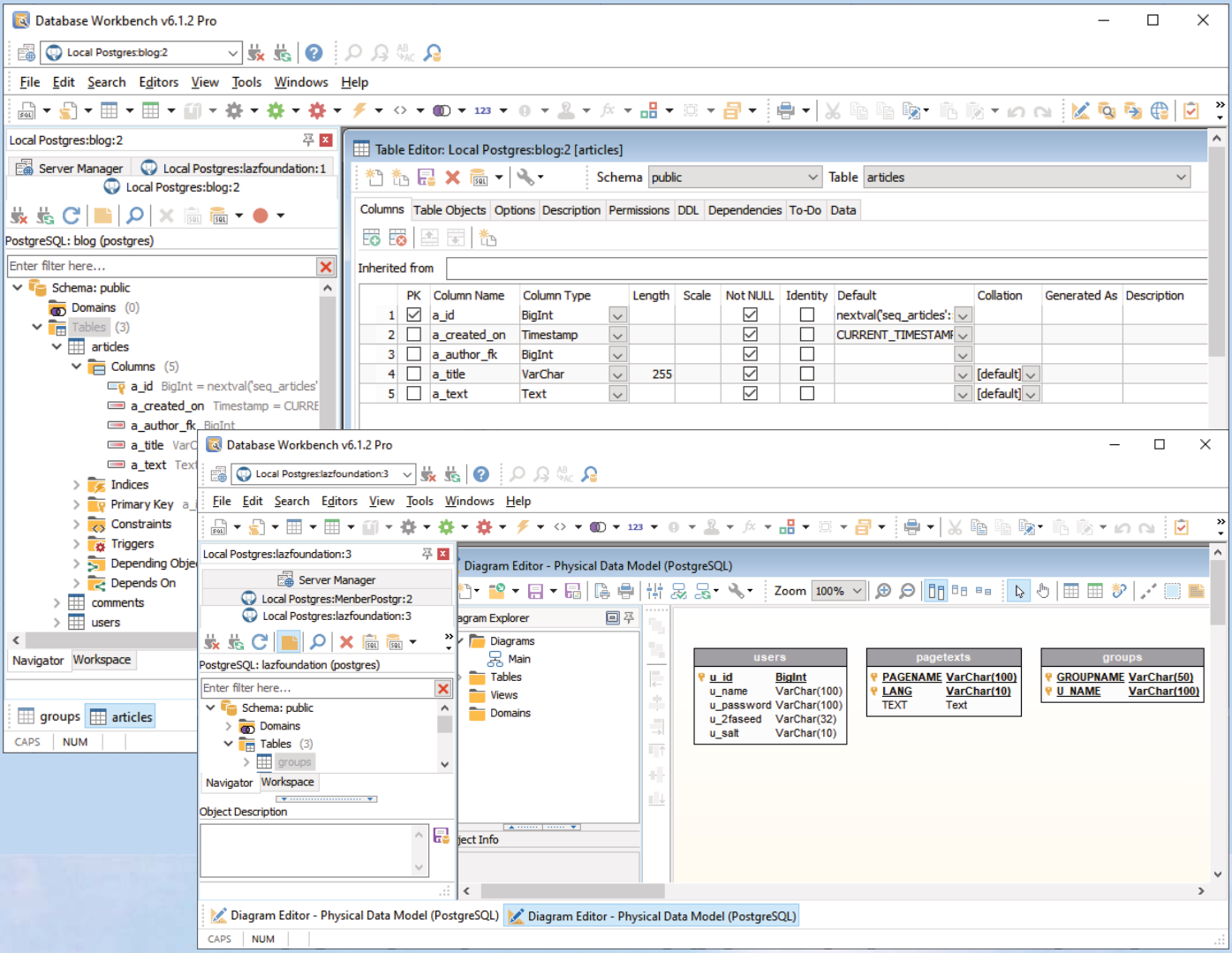
THE NEW INTERNET BLAISE PASCAL LIBRARY 2023

https://library.blaisepascalmagazine.eu/

JUST OPEN ANY BROWSER
(CHROME, SAFARI, EDGE, FIREFOX, OPERA, DUCKDUCKGO)
AND LOGIN: YOU WILL HAVE ALL ISSUES AVAILABLE - 6500 PAGES. FOR ALL ISSUES STARTING AT NR1 UP TO THE LATEST ITEM.



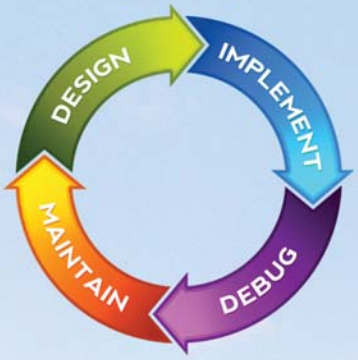
READ WHERE EVER THE INTERNET IS AVAILABLE



Introducing

Database Workbench 6

database development environment



Consistent user interface, modern code editors, Unicode enabled, HighDPI aware, ER designer, reverse engineering, meta data browsing, visual object editors, meta data migration, meta data compare, stored routine debugging, SQL plan visualizer, test data generator, meta data printing, data import and export, data pump, Grant Manager, DBA tasks, code snippets, SQL Insight, built in VCS, report editor, database meta data search, numerous productivity tools and much more...

for SQL Server, Oracle, MySQL, MariaDB, Firebird, InterBase, NexusDB and PostgreSQL

Upscene *Database tools for developers*

www.upscene.com

AUSFÜHREN VON PROGRAMMEN AUF DEM SERVER IN PAS2JS

Von Michael Van Canneyt

ABSTRACT
In diesem Artikel zeigen wir, wie man dem Benutzer eines browserbasierten Programms **Rückmeldung von lang laufenden Prozessen auf dem Server** geben kann. Dazu verwenden wir 2 Komponenten: eine in **PAS2JS**, eine in **Free Pascal/Lazarus**.

❶ EINFÜHRUNG

Bei der Verwendung eines webbasierten Programms kann nicht alles im Browser erledigt werden.
Oft werden Aufgaben über einen **RPC-Mechanismus (Remote Procedure Call)** auf dem Webserver ausgeführt. Dabei kann es sich um eine einfache Aufgabe wie die Ausführung einer SQL-Anweisung in einer Datenbank und die Rückgabe eines Ergebnisses handeln.
Es kann sich aber auch um eine kompliziertere und zeitaufwändigere Aufgabe handeln, wie z. B. die Erstellung einer Sicherungskopie einer Datenbank, die Indizierung von PDF-Dateien, die Kompilierung eines Softwareprojekts und die Durchführung einer Testsuite oder sogar die Installation von Software auf dem Server.
Idealerweise sollte die Ausgabe dieser Remote-Programme auch dem Benutzer angezeigt werden.

Um die Skalierbarkeit der Programme zu gewährleisten, sollten diese Aufgaben nur von kurzer Dauer sein. Eine Rückkehrzeit von 1 Sekunde für eine HTTP-Anfrage ist bereits eine lange Zeit, so dass es keine gute Idee ist, eine zeitaufwändige Aufgabe auszuführen und auf die Rückkehr mit einer einzigen HTTP-Anfrage zu warten:
Der HTTP-Server ist mit der Anfrage beschäftigt, der Browser oder etwaige Proxy-Server zwischen dem HTTP-Server und dem Browser können beschließen, Ihre Anfrage zu unterbrechen.
Viel besser ist es, den Prozess mit einer HTTP-Anfrage zu starten und einen Mechanismus zur Abfrage des Status des ausgeführten Prozesses zu verwenden.
In diesem Artikel stellen wir einen solchen Mechanismus vor.

❷ ARCHITEKTUR

Die Lösung, die wir hier vorstellen, besteht aus 2 Komponenten. Die eine Komponente wird auf dem Server eingesetzt und kann dazu verwendet werden, einen Prozess zu starten, seine Ausgabe zu erfassen und den Status des Prozesses abzufragen.
Die andere Komponente kümmert sich um den Polling-Prozess auf dem Client.

Diese Komponenten wissen nichts über den Kommunikationsmechanismus zwischen Browser und Server, d.h. sie implementieren nicht die eigentlichen RPC-Aufrufe, die zum Starten des Prozesses verwendet werden: Es gibt viele mögliche Mechanismen, und einige können für Ihren Zweck besser geeignet sein als andere.
Die Komponenten heißen `TProcessCapture` für den Serverteil und `TProcessCapturePoller` für den Clientteil (**PAS2JS**).

Der **Serverteil** kümmert sich um die Ausführung eines Programms und die Umleitung der Ausgabe in eine Datei, der **Clientteil** implementiert den **Polling-Mechanismus** und einige **Callbacks**, um die eigentlichen Serveraufrufe und das Ergebnis zu verarbeiten.
Wir werden beide Komponenten mit einer einfachen Reihe von Programmen demonstrieren:

- Ein Testprogramm, das ausgeführt werden soll.
  Es wird nur zu Demonstrationszwecken verwendet.
- Ein HTTP-Serverprogramm, das es erlaubt
  HTML-Dateien und bietet einen
- RPC-Mechanismus zum Starten des Testprogramms und
  Statusanfragen zu behandeln. Ein einfaches PAS2JS-Programm
  das im Browser ausgeführt wird und das Testprogramm ferngesteuert
  das Testprogramm ausführt. Es zeigt die Ausgabe des
  des Testprogramms im Browser.

Wir beginnen mit dem Testprogramm.

❸ DAS TESTPROGRAMM

Um die Funktionsweise zu demonstrieren, muss das Testprogramm 4 Dinge tun:

1. Es muss **eine gewisse Zeit** lang laufen, mindestens einige Sekunden.
   Dies wird mit einer einfachen Schleife und einem Aufruf von „**sleep**“ erreicht.
2. Es muss zeigen, dass es **Befehlszeilenargumente empfängt**:
   Wir werden einfach die Programmparameter ausgeben.
3. Es muss zeigen, dass es in einem bestimmten **Verzeichnis ausgeführt** wird.
   Wir werden einfach das Arbeitsverzeichnis ausgeben.
4. Es muss eine **OUTPUT** erzeugen.

All dies lässt sich leicht mit einem trivialen Programm bewerkstelligen:

```
uses sysutils;

var
  i : integer;
  D : TDateTime;
begin
  Writeln('Current dir: ', GetCurrentDir);
  Write('Args:');
  For I:=1 to ParamCount do
    Write(' ',ParamStr(i));
  Writeln();
  D:=Now;
  For I:=1 to 150 do
    begin
    Sleep(100);
    Writeln('Tick ',i);
    Flush(output);
    end;
  Writeln(SecondsBetween(Now,D), ' seconds elapsed');
  flush(output);
end;
```

Das einzig Bemerkenswerte an diesem Programm ist, daß es die Standardausgaben (output) nach dem Schreiben einer Zeile leert: Standardmäßig puffert **Free Pascal** die Ausgabe von `writeln`-Anweisungen, wenn es feststellt, daß es nicht auf eine Konsole schreibt.
Da unser Programm mit umgeleiteter Ausgabe ausgeführt wird, wird die Pufferung aktiviert, und um die Ausgabe schneller an den Browser zu senden, leeren wir die Standardausgabe manuell.

❹ DIE SERVER-KOMPONENTE

Bevor die Server-Komponente erklärt wird, ist es eine gute Idee zu erklären, warum eine neue Komponente benötigt wird. Schließlich wird **Free Pascal** schon seit Ewigkeiten mit der `TProcess`-Komponente ausgeliefert, die dazu benutzt werden kann, einen Prozess zu starten und seine Ausgabe über einen Stream zu lesen.
Warum also nicht einfach diese Komponente verwenden? Diese Komponente ist für unsere Aufgabe nicht wirklich geeignet, und zwar aus mehreren Gründen:

- Ein Webserver-Prozess (*z.B. cgi, fastcgi*) kann beendet werden, bevor der Prozess abgeschlossen ist. Alle Informationen über den ausgeführten Prozess würden verloren gehen.
- Die Komponente kann nicht verwendet werden, um die Ausgabe in eine Datei umzuleiten. Dies würde alle Daten in einem **separaten Thread** aus der Datei lesen, sie irgendwo speichern usw. Dies verkompliziert die Sache erheblich, und wenn das HTTP-Programm beendet wird, würden alle weiteren **Ein-/Ausgaben** stoppen. In ähnlicher Weise kann keine Eingabedatei angegeben werden, sie würde würde eine ähnliche Behandlung wie die Ausgabedatei erfordern.
- Da die `TProcess`-Komponente auf einen einzigen Prozess beschränkt ist auf einen einzigen Prozess beschränkt ist, gibt es keine Möglichkeit, die Ihrer Webanwendung skaliert.(Größe ändert)

Im Wesentlichen ist die `TProcess`-Komponente zustandsabhängig, und wir brauchen eine zustandslose Komponente, um in einer Web-Umgebung zu arbeiten.
Es wird also eine neue Komponente benötigt.
Die Serverkomponente `TProcessCapture` hat die folgende Deklaration:

```
TProcessCapture = Class(TComponent)
Public
  Function Execute(Exe : String; Args: Array of string) : string;
  Function Execute(Exe : String; Args: TStrings) : string;
  Function CleanupProcess(Const AProcess : String) : Boolean;
  Function GetOutputFile(Const AProcess : String) : String;
  Function GetPidFile(Const AProcess : String) : String;
  Function GetStatusFile(Const AProcess : String) : String;
  Function GetProcessID(Const AProcess : String) : Integer;
  Function IsProcessRunning(Const AProcess : String) : Boolean;
  Function GetProcessExitStatus(Const AProcess : String) : Integer;
  Function GetProcessOutput(Const AProcess : String; Var AOffSet : Integer) : RawByteString;
Published
  Property LogDir : String Read FLogDir Write FLogDir;
  Property InputFile : String Read FInputFile Write FInputFile;
  Property Working-Dir : String Read FWorkingDir Write FWorkingDir;
  Property OutputCodePage : TSystemCodePage Read FOutputCodePage Write FOutputCodePage;
end;
```

Die Hauptmethoden sind fast selbsterklärend:

- **`Execute`** - führt das Programm Exe aus und übergibt ihm die Argumente **Args**, die als Array von `Strings` oder als `Stringliste` angegeben werden können. Der Rückgabewert dieser Funktion ist ein Prozessbezeichner.
- **`CleanupProcess`** - bereinigt die Ausgabe- und Statusdateien für den durch `AProcess` identifizierten Prozess. Sie sollten diese Funktion erst aufrufen, nachdem der Prozess beendet wurde.
- **`IsProcessRunning`** - gibt True zurück, wenn der durch `AProcess` identifizierte Prozess noch läuft.
- **`GetProcessExitStatus`** - gibt den Exit-Status des durch `AProcess` identifizierten Prozesses zurück.
  Wenn der Prozess noch läuft, wird -1 zurückgegeben.
- **`GetProcessOutput`** - gibt die Ausgabe des von `AProcess` identifizierten Prozesses zurück, beginnend beim `Byte-Offset AOffSet` (*nullbasiert*) bis zum Ende der verfügbaren Ausgabe. Es gibt einige Hilfsmethoden, die Sie unter normalen Umständen nicht benötigen:
- **`GetOutputFile`** - gibt den Namen der Ausgabedatei zurück, die mit dem Prozess `aProcess` verbunden ist.
- **`GetPidFile`** - gibt den Namen der **Prozess-ID-Datei** zurück, die mit dem Prozess `aProcess` verbunden ist. Diese Datei wird erstellt, sobald der Prozess startet.
- **`GetStatusFile`** - gibt den Namen der **Statusdatei** zurück, die mit dem Prozess `aProcess` verbunden ist. Diese Datei existiert erst, nachdem das Programm beendet wurde.
- **`GetProcessID`** - gibt die Prozess-ID des Prozesses AProcess zurück.
  Schließlich gibt es noch einige Eigenschaften:
- **`LogDir`** - das Verzeichnis, in dem alle Protokoll- und Statusdateien erstellt werden. Das Verzeichnis wird erstellt, wenn es nicht vorhanden ist.
- **`InputFile`** - eine Datei mit vorbereiteten Eingaben für den Prozess.

HINWEIS: Sie können damit nicht mit dem Prozess interagieren.
Diese Eigenschaft wird nur beim Starten des Programms verwendet.

- **`WorkingDir`** - Das Arbeitsverzeichnis für das gestartete Programm. Diese Eigenschaft wird nur beim Starten des Programms verwendet.
- **`OutputCodePage`** - Die Codepage, in der das Programm seine seine Ausgabe schreibt.

Um mit dieser Komponente zu arbeiten, führen Sie normalerweise die folgenden Schritte aus:

❶ Setzen Sie geeignete Werte für `LogDir, InputFile, WorkingDir` und `OutputCodePage`. Sie enthalten sinnvolle Standardwerte, aber es ist besser, sie explizit anzugeben.
❷ Starten Sie das Programm mit der Execute-Methode, und speichern Sie die resultierende ProcessID-Zeichenkette.
❸ Initialisieren Sie eine Offset-Variable auf Null.
❹ Prüfen Sie mit IsProcessRunning, ob der Prozess noch läuft, und übergeben Sie ihm ProcessID.
❺ Rufen Sie die Ausgabe des Prozesses mit GetProcessOutput ab und übergeben Sie dabei ProcessID und den aktuellen Versatz. Aktualisieren Sie den Versatz.
❻ Wiederholen Sie die letzten 2 Schritte, bis das Programm beendet wird.

Es ist zu beachten, dass Sie `TProcessCapture` nach jedem Schritt freigeben und vor der Ausführung eines Aufrufs neu erstellen können: es ist zustandslos. Dies ist notwendig, wenn die Komponente in einer Web-Umgebung arbeiten soll, in der die verschiedenen Schritte als Teil verschiedener HTTP-Anfragen ausgeführt werden: die Schritte können von verschiedenen Instanzen des Anwendungsservers ausgeführt werden.

Um korrekt zu funktionieren, müssen die Eigenschaften `LogDir` und `OutputCodePage` zwischen den Aufrufen auf dieselben Werte gesetzt werden.

Das bedeutet auch, dass ein und dieselbe Komponente zur Steuerung verschiedener Prozesse verwendet werden kann. Dies ist jedoch nicht empfehlenswert, wenn Sie **Threads** verwenden: Die Komponente ist nicht **re-entrant**.

Um ihre Arbeit zu erledigen, führt die Komponente `TProcessCapture` ein kleines Hilfsprogramm namens `taskhelper` aus: Dieses Programm startet das eigentliche Programm, das mit umgeleiteter Ein- und Ausgabe ausgeführt werden muss.

Es kümmert sich auch um die Registrierung des **Exit-Status** des Programms. Auf **Unix**-Plattformen ist es möglich, ohne dieses Programm auszukommen, aber unter **Windows** erfordert der Mechanismus zum Starten eines neuen Prozesses `CreateProcess` die Verwendung eines zusätzlichen Programms. Um das Verhalten auf allen Plattformen einheitlich zu gestalten, wird überall das Programm taskhelper verwendet, dessen Quellen mit der Stammversion von **FPC** verteilt werden, das aber in die Quellen dieses Artikels aufgenommen wurde.

❺ THE SERVER PROGRAM

Um die Funktionsweise der Komponente zu demonstrieren, erstellen wir einen kleinen HTTP-Server, der das Testprogramm ausführt, wenn er einen **StartProcess**-Befehl vom Browser über **JSON-RPC** erhält, und der über einen **GetStatus**-Befehl verfügt, um den Status des Prozesses abzurufen.

Der Prozess wird auch die Dateien für die **Client-Anwendung** bereitstellen. Dazu wählen wir im Dialogfeld "**Neues Projekt**" die Option "**HTTP-Server-Anwendung**", und Im angezeigten Assistenten wählen wir "**Serverdateien vom Standardspeicherort**" und unter "**Zu erstellendes Webmodul**" wählen wir "**Web JSON-RPC Modul**", wie in Abbildung 1 auf Seite 5 des Artikels dargestellt.

Im nächsten Dialog, in dem das Modul als **JSON-RPC**-Modul erstellt wird, müssen wir nur das Webmodul registrieren (*wir haben nur 1 Modul in der Serveranwendung*),(*siehe Abbildung 2 auf Seite 6*) und wir verwenden den URL-Pfad `/RPC`, um **JSON-RPC** Anfragen zu bedienen.

Danach benötigen wir 2 **JSONRPCHandler-Komponenten** aus der Registerkarte **FPWeb** in der Komponentenpalette, eine für jede Anforderung:

`StartProcess` - dieser Aufruf benötigt 2 Argumente: 2 Strings,
die wir in der Eigenschaft
in der Eigenschaft `Params` der Komponente definieren müssen.
Wir geben ihnen die Namen A und B. Der Aufruf gibt den **Prozessbezeichner** an die Client-Anwendung Anwendung.

`GetStatus` dieser Aufruf nimmt ebenfalls 2 Argumente entgegen:
eine Zeichenkette (die `ProcessID`) und eine Int64-Zahl (*ein Offset*),
die wir ebenfalls definieren sie in der Eigenschaft `Params`.
Der Aufruf wird den Prozess zurückgeben **exit code** (*-1 wenn der Prozess noch läuft*), die verfügbare Ausgabe beginnend mit dem angegebenen Offset-Identifikator an die **Client-Anwendung** zurück. Er gibt auch
wird der neue Offset zurückgegeben.

Diese beiden **RPC-Aufrufe** sind die **API**, die wir dem Browser zur Steuerung unseres Prozesses zur Verfügung stellen.
Die eigentliche Arbeit wird von der Komponente `TProcessCapture` erledigt.
Die `TProcessCapture` Komponente ist (*noch*) nicht auf der Komponentenpalette von Lazarus, also erstellen wir sie im Code im `OnCreate` Handler des Datenmoduls und zerstören sie im `OnDestroy` Handler:

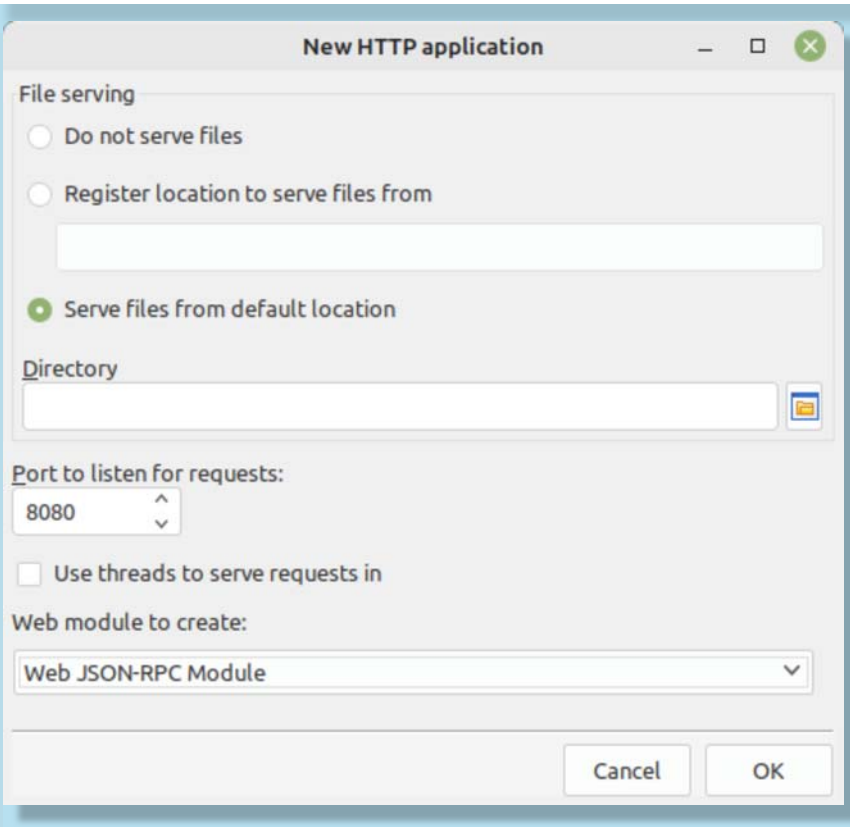


Abbildung 1: Der Start der Serveranwendung

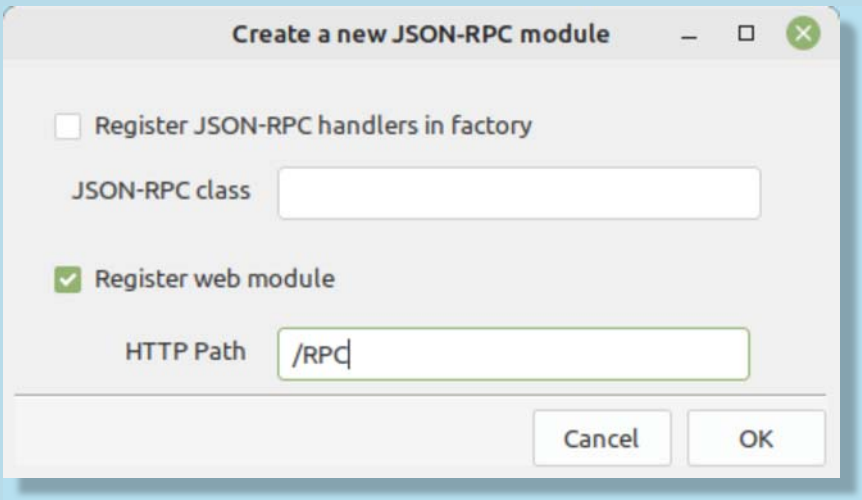


Abbildung 2: Erstellen des JSON-RPC-Webmoduls

```
procedure Tprocesscontrol.DataModuleCreate(Sender: TObject);
begin
  Capture:=TProcessCapture.Create(Self);
end;

procedure Tprocesscontrol.DataModuleDestroy(Sender: TObject);
begin
  FreeAndNil(Capture);
end;
```

Letzteres ist streng genommen nicht notwendig, da die Komponente zum Datenmodul gehört und zerstört wird, wenn das Datenmodul zerstört wird, aber der Übersichtlichkeit halber zerstören wir sie trotzdem manuell.

Im `OnExecute`-Ereignis des `StartProcess`-Handlers sammeln wir die beiden Argumente A und B und starten das Testprogramm:

```
const
  LongProcess = 'longprocess' {$ifdef windows} + '.exe' {$endif} ;

var
  arr : TJSONArray absolute Params;
  a, b, Exe, PID : string;
begin
  Res:=Nil;
  a:=Arr.Strings[0];
  b:=Arr.Strings[1];
  Exe:=ExtractFilePath(ParamStr(0))+longprocess;
  PID:=Capture.Execute(Exe,[a,b]);
  Res:=TJSONString.Create(PID);
```

Wie Sie in diesem Code sehen können, verwenden wir die **Execute**-Methode der `TProcessCapture`-Klasse, um den Prozess zu starten.

Für den Aufruf von `GetStatus` ist der Code ein wenig länger, aber nicht so sehr. Der Code beginnt mit dem Abrufen der Argumente und der Überprüfung, ob der Prozess noch läuft. Wenn der Prozess nicht mehr läuft, wird der **Exit-Status** abgefragt.

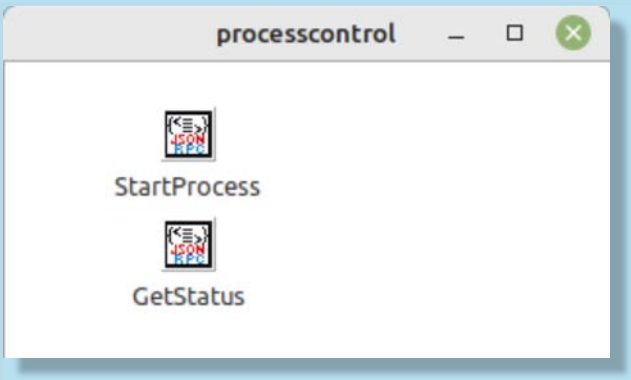


Abbildung 3: Das fertige JSON-RPC-Webmodul

```
procedure Tprocesscontrol.GetStatusExecute(Sender: TObject;
  const Params: TJSONData; out Res: TJSONData);
var
  arr : TJSONArray absolute Params;
  PID,aOutput : string;
  Offset,Status : Integer;
begin
  Res:=Nil;
  PID:=Arr.Strings[0];
  OffSet:=Arr.Int64s[1];
  if Capture.IsProcessRunning(PID) then
    Status:=-1
  else
    Status:=Capture.GetProcessExitStatus(PID);
  aOutput:=Capture.GetProcessOutput(PID,Offset);
  Res:=TJSONObject.Create(['status',Status,'output',aOutput,'offset',offset]);
end;
```

Unabhängig davon, ob der Prozess noch läuft oder nicht, wird schließlich die verfügbare Ausgabe abgerufen und alle 3 Elemente (**Status, Ausgabe, neuer Offset**) werden in einem **JSON-Objekt** an den Client zurückgegeben.

Das Datenmodul wird wie in *Abbildung 3 auf Seite 6* dieses Artikels aussehen. Bevor das Programm verwendet werden kann, sind noch zwei letzte Dinge zu erledigen, wenn die Release-Version von **FPC** unter **Linux** verwendet wird. Die HTTP-Verbindung, über die die Anfragen eintreffen, wird an den **Task-Helper** übergeben, und folglich wird die Verbindung nicht geschlossen wenn der StartProcess-Aufruf zurückkehrt, so dass der Browser wartet, bis der Prozess beendet ist.

Dies macht natürlich den Zweck der ganzen Übung zunichte.
Um dies zu beheben, müssen wir das `Close-On-Exec`-Flag auf dem Socket-Handle setzen. Dies lässt sich leicht bewerkstelligen, indem wir den `OnAllowConnect`-Handler des HTTP-Servers verwenden.
Dazu fügen wir Folgendes in die Projektdatei ein:

```
THTTPApplication = Class(fphttpapp.THTTPApplication)
  constructor Create(aOwner : TComponent); override;
private
  procedure DoConnect(Sender: TObject; ASocket: Longint; var Allow: Boolean);
end;

{ THTTPApplication }

constructor THTTPApplication.Create(aOwner: TComponent);
begin
  inherited Create(aOwner);
  OnAllowConnect:=@DoConnect;
end;

procedure THTTPApplication.DoConnect(Sender: TObject; ASocket: Longint;
                                     var Allow: Boolean);
{$IFDEF UNIX}
const
FD_CLOEXEC = 1;
 {$ENDIF}
begin
 {$IFDEF UNIX}
FpFcntl(aSocket, F_SETFD, FD_CLOEXEC);
 {$ENDIF}
  Allow:=True;
end;
```

Um schließlich die Dateien des Client-Programms bereitzustellen, setzen wir das Basisverzeichnis für das **Dateiservicemodul** auf das Verzeichnis mit den Dateien des Client-Programms:

```
Function GetBaseDir : String;
begin
  Result:=ExtractFilePath(ParamStr(0));
  Result:=Result+'..'+PathDelim+'client';
  Result:=ExpandFileName(Result);
end;
```

(*dieser Code geht davon aus, dass es 2 Verzeichnisse gibt: eines für den Server, eines für den Client*). Schließlich laden wir alle bekannten Mime-Typen und erstellen unsere eigene HTTP-Anwendung:

```
Var
 Application:THTTPApplication;
begin
 MimeTypes.LoadKnownTypes;
 TSimpleFileModule.BaseDir:=GetBaseDir;
 TSimpleFileModule.RegisterDefaultRoute;
 Application:=THTTPApplication.Create(Nil);
 Application.Title:='Process server';
 Application.Port:=8060;
 Application.Initialize;
 Application.Run;
 Application.Free;
end;
```

Beachten Sie, dass wir den HTTP-Port auf Port 8060 festgelegt haben.

❻ THE BROWSER CLIENT-SIDE COMPONENT

Im Browser wird die Komponente TProcessCapturePoller verwendet, um die Arbeit mit der Komponente TProcessCapture auf dem Server zu unterstützen. Sie startet nicht den eigentlichen Prozess, sondern fragt lediglich den Server nach dem Status des gestarteten Prozesses ab und löst eine Reihe von Ereignissen auf der Grundlage der Ergebnisse aus. Sie verwaltet auch den Status des **Ausgabeoffsetparameters**. Es gibt Eigenschaften, die steuern, wie oft und wie lange der **Abfragemechanismus** versuchen muss und wie viele Fehler toleriert werden können, bevor die Abfrage abgebrochen wird.
Um unabhängig vom tatsächlich verwendeten **RPC-Mechanismus** zu sein, wird die eigentliche Abfrage ebenfalls über ein Ereignis durchgeführt. Es liegt in der Verantwortung des Programmierers, dieses Ereignis zu implementieren und den **ReportProgress**-Mechanismus zu verwenden, um die **Serverergebnisse** an die Komponente zu übermitteln.
Diese Komponente hat die folgende Deklaration:

```
Type
 TProcessStatus = (psRunning, // Process still running
                   psExited, // Process has exited
                   psError // Too many errors
                   );

 TOnGetProcessStatusEvent =
          Procedure (Sender : TObject; aProcessID : String; aOffset : NativeInt)
 TOnProcessDoneEvent =
          Procedure (Sender : TObject; aStatus : TProcessStatus; aExitCode : Integer)
 TOnProcessOutputEvent =
          Procedure (Sender : TObject; aOutput : String) of object;
 TOnStatusFailEvent =
          Procedure (Sender : TObject; aError : String) of object;

TProcessCapturePoller = class(TComponent)
```

```
Public
  Procedure Start;
  Procedure Cancel;
  Procedure ReportProgress(aStatus : TProcessStatus;
                           aOutput : String;
                           aExitCode : Integer;
                           aOffSet : NativeInt);
  Procedure ReportProgressFail(const aMessage : string);
  Property Canceled : Boolean ;
  Property FailCount : Integer;
  Property StatusCheckCount : Integer;
  Property OutputOffset : NativeInt;
Published
  Property ProcessID : String;
  Property OnGetProcessStatus : TOnGetProcessStatusEvent;
  Property OnProcessDone : TOnProcessDoneEvent;
  Property OnProcessOutput : TOnProcessOutputEvent;
  Property OnStatusFail : TOnStatusFailEvent;
  Property LinebasedOutput : Boolean;
  Property PollInterval : Integer;
  Property MaxFailCount : Integer;
  Property MaxCheckCount : Integer;
end;
```

Die Methoden erfüllen die folgenden Aufgaben

- **`Start`** - startet den Abfragevorgang.
- **`Cancel`** - bricht den Abfragevorgang ab.
- **`ReportProgress`**

- diese Methode muss verwendet werden, wenn der `OnGetProcessStatus`-Eventhandler den Status des Prozesses vom Server erhalten hat. Der Parameter `aStatus` ist einer der verfügbaren Status, `aOutput` ist die Ausgabe des Prozesses. Der Parameter `aExitCode` ist der **Exit-Code** (*falls der Status* `psExited` *ist*) und `aOffset` ist der neue Offset (*wie vom Server gemeldet*).

- **`ReportProgressFail`**

- diese Methode muss verwendet werden, wenn der **Serveraufruf** zum Abrufen des Prozessstatus fehlgeschlagen ist. Der Statusparameter `aMessage` kann verwendet werden, um anzugeben, was genau fehlgeschlagen ist.
Die folgenden Ereignisse können behandelt werden:

- **`OnGetProcessStatus`**

- Dies ist das einzige Ereignis, das implementiert werden muss. Es wird in regelmäßigen Abständen ausgelöst, wenn der Poller den Status des **Serverprozesses** abfragen muss. Der Poller übergibt die Prozess-ID und den aktuellen Ausgabe-Offset an das Ereignis, so dass der Benutzer den Status dieser Parameter nicht zu verfolgen braucht.

- **`OnProcessDone`**

- Dieses Ereignis wird aufgerufen, wenn der Prozess beendet oder die Abfrage abgebrochen wurde. Es meldet den Status (`psError` *im Falle eines Fehlers*) und den **Exit-Code** des Prozesses.

- **`OnProcessOutput`**

- Dies wird aufgerufen, wenn die Ausgabe des Prozesses empfangen wurde: Der Parameter `aOutput` enthält die gemeldete Ausgabe. Dieses Ereignis wird mehrfach aufgerufen.

- **`OnStatusFail`**

- Dieses Ereignis wird aufgerufen, wenn `ReportProgressFail` aufgerufen wurde, um ein Scheitern des Aufrufs zum Abrufen des Prozessstatus zu signalisieren. Es kann mehrfach aufgerufen werden, abhängig vom Wert von `MaxFailCount`.
Das Verhalten der Komponente wird gesteuert durch die folgenden Eigenschaften gesteuert:

- **`LinebasedOutput`**

- Wenn auf True gesetzt, teilt die Komponente die empfangene Ausgabe in Zeilen, und ruft `OnProcessOutput` für jede Zeile auf, anstatt die gesamte empfangene Ausgabe in einem Aufruf zu melden (*wenn auf False gesetzt*)

• **`PollInterval`** die Zeitspanne (in Millisekunden), nach der das Ereignis OnGetProcessStatus ausgelöst wird. Standard ist 500ms. HINWEIS: Das Ereignis wird erst dann erneut ausgelöst, wenn das Ergebnis (Erfolg oder Misserfolg) des vorherigen Ereignisses gemeldet wurde. Dies geschieht, um sich überschneidende getstatus-Aufrufe zu vermeiden.

• **`MaxFailCount`** Die maximale Anzahl von Fehlern, die gemeldet werden können bevor die Abfrage abgebrochen wird. Die Voreinstellung ist 1.

• **`MaxCheckCount`** Die maximale Anzahl von Abfragen, die die Komponente durchführt, bevor sie eine Zeitüberschreitung meldet.
Schließlich können die folgenden Eigenschaften verwendet werden, um einige Informationen über den Abfrageprozess zu erhalten:

• **`Canceled`** Der Abfragevorgang wurde abgebrochen.

• **`FailCount`** - Die Anzahl der Fehlschläge seit Beginn der Abfrage.

• **`StatusCheck`** - Count Die Anzahl, wie oft der Status noch geprüft wird.

• **`OutputOffset`** - Der aktuelle Ausgabeoffset.

Es mag seltsam erscheinen, die Ereignisse `OnProcessDone, OnStatusFail` und `OnProcessOutput` zu haben, wenn die Abfrage des Prozessstatus in einem Ereignis implementiert werden muss: sicherlich kann der Ereignishandler die Ausgabe anzeigen, entscheiden, wann der Prozess beendet ist usw.

Dafür gibt es zwei Gründe:
Erstens kann die Zustandslogik für die Ausgabe von der Komponente gehandhabt werden, aber noch wichtiger ist, dass die Komponente durch die Verfügbarkeit dieser Ereignisse leicht als Elternteil für nachfolgende Komponenten verwendet werden kann, die den **Polling-RPC-Mechanismus** in die Komponente integrieren. (*wie unten gezeigt wird*).

❼ DAS CLIENT PROGRAM

Mit dieser Komponente können wir nun das clientseitige Programm starten.
Im Dialogfeld "Projekt - Neues Projekt" wählen wir den Punkt "Webbrowser-Programm" und geben die richtigen Einstellungen ein, wie in Abbildung 4 auf Seite 13 dargestellt.
Die HTML-Datei wird am besten als index.html gespeichert.
Die HTML-Datei benötigt 5 Elemente:

❶ Ein Button zum Starten des Prozesses.
❷ Eine Button zum Abbrechen des **Abfragevorgangs**.
❸ Eine Eingabe für den Parameter A für das gestartete Programm.
❹ Eine Eingabe für den Parameter B für das gestartete Programm.
❺ Ein HTML-Element, in dem die Ausgabe des Programms angezeigt wird.
Wir werden dafür den **Ausgabemechanismus** der Browser-**Konsoleneinheit** verwenden:
eine einfache Writeln-Anweisung führt zum Anhängen der Ausgabe an dieses Element.

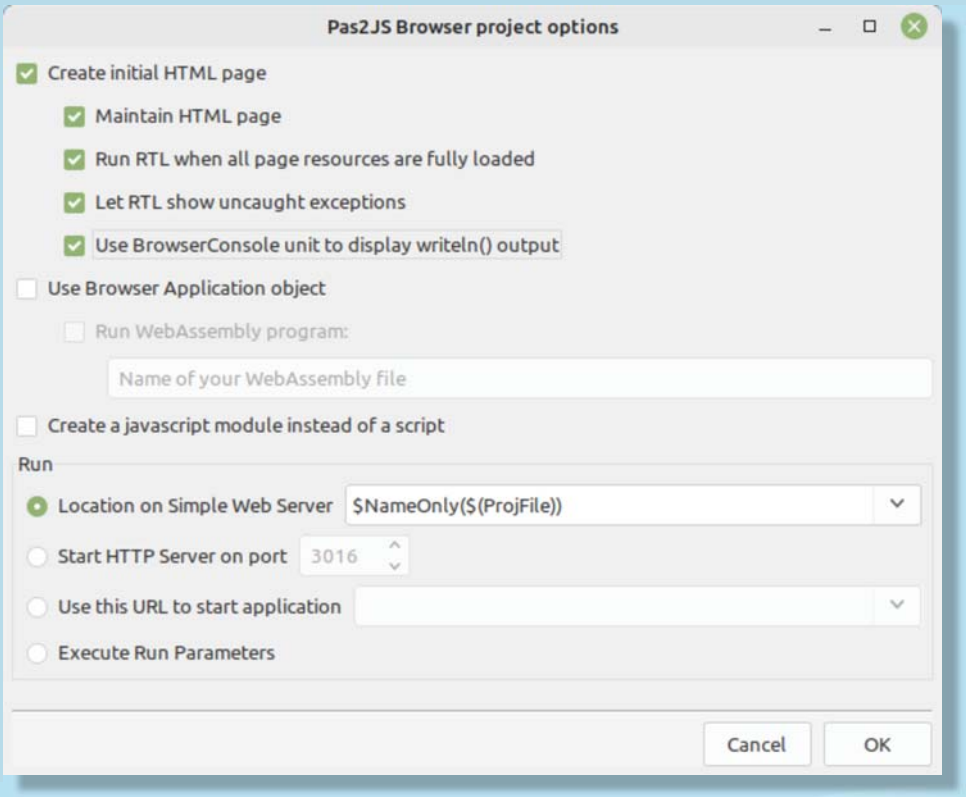


Abbildung 4: Erstellen des Client-Programms

The following simple HTML (*using Bulma CSS*) will do the job just fine:

```
<h3 class="title is-3">Process output demo</h3>
<div class="box">
  <h4 class="title is-4">Start parameters</h4>
  <div class="field">
    <label class="label">Argument A</label>
    <div class="control">
      <input id="edtA" type="text" class="input"
             placeholder="Enter argument A">
    </div>
  </div>
<div class="field">
  <label class="label">Argument B</label>
  <div class="control">
    <input id="edtB" type="text" class="input"
      placeholder="Enter argument B">
  </div>
</div>
<div class="field is-grouped">
  <div class="control">
    <button id="btnStart" class="button is-primary">
      Start process
    </button>
  </div>
  <div class="control">
     <button id="btnCancel" class="button is-warning is-light">
        Cancel
     </button>
    </div>
  </div>
</div>
<div class="box">
  <h4 class="title is-4">Process output</h4>
  <div id="pasjsconsole">
  </div>
</div>
```

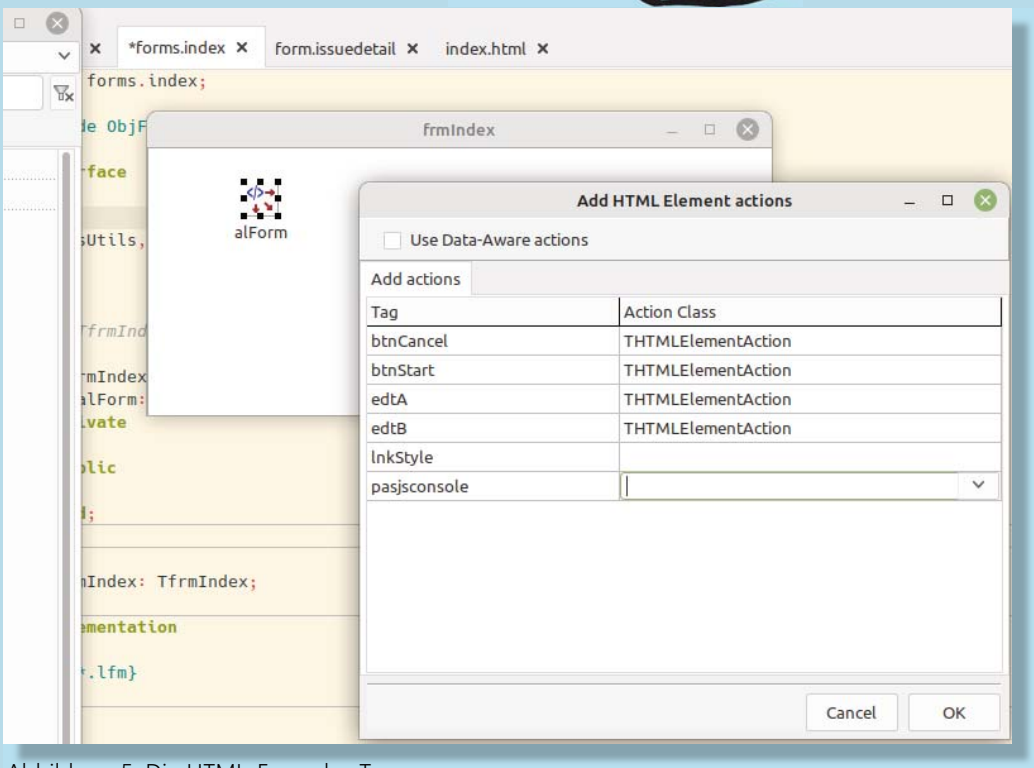


Abbildung 5: Die HTML-Formular-Tags

Um mit diesem HTML-Code zu interagieren, erstellen wir zunächst ein **HTML-Fragmentmodul** mit Hilfe des Dialogs "**Datei - neu**". Wir nennen es '`frmIndex`' und setzen die Eigenschaft '**UseProjectHTML**' auf True. Auf dieses Modul legen wir eine `THTMLElementActionList`-Komponente aus der Komponentenpalette ab.
Über das Komponenten-Kontextmenü "**Aktionen für HTML-Tags erstellen**" können wir Aktionen für alle Tags im obigen HTML erstellen, wie in *Abbildung 5 auf Seite 12* des Artikels gezeigt.
Wir brauchen einen `TPas2jsRPCClient` aus der `Pas2JS-Registerkarte` in der Komponentenpalette: Diese Komponente wird die RPC-Anfragen bearbeiten, und wir nennen sie kurz RPC. Die Komponente kann ihre Arbeit nur dann korrekt ausführen, wenn sie weiß, wo sich der Server befindet: Wir müssen die Eigenschaft URL eingeben.
Wie in einem früheren Artikel gezeigt, können wir nun einen Dienst-Proxy erzeugen: Das ist eine Klasse mit korrekten Methodendefinitionen, die die in unserem RPC-Server definierten Methoden widerspiegeln. Durch den Aufruf dieser Dienstmethoden werden die Methoden auf dem Server tatsächlich ausgeführt. Wenn Sie mit der rechten Maustaste auf die RPC-Komponente klicken und "**Dienst-Client-Komponente erstellen**" wählen, wird das Dialogfeld für die Dienstgenerierung angezeigt, wie in *Abbildung 6 auf Seite 13* des Artikels dargestellt. Wir nennen die Unit '`processservice`' und weisen die IDE an, sie dem Projekt hinzuzufügen.

Jetzt können wir mit der Codierung der Anwendung beginnen. Wir werden den TProcessCapturePoller und den Service-Client im OnCreate-Ereignis unseres Index-Formularmoduls erstellen:

```
procedure TfrmIndex.DataModuleCreate(Sender: TObject);
begin
  Service:=TprocesscontrolService.Create(Self);
  Service.RPCClient:=RPC;
  FPoller:=TProcessCapturePoller.Create(Self);
  FPoller.OnProcessOutput:=@DoDoutput;
  FPoller.OnGetProcessStatus:=@DoGetStatus;
  FPoller.OnProcessDone:=@DoProcessDone;
  FPoller.OnStatusFail:=@DoStatusFail;
end;
```

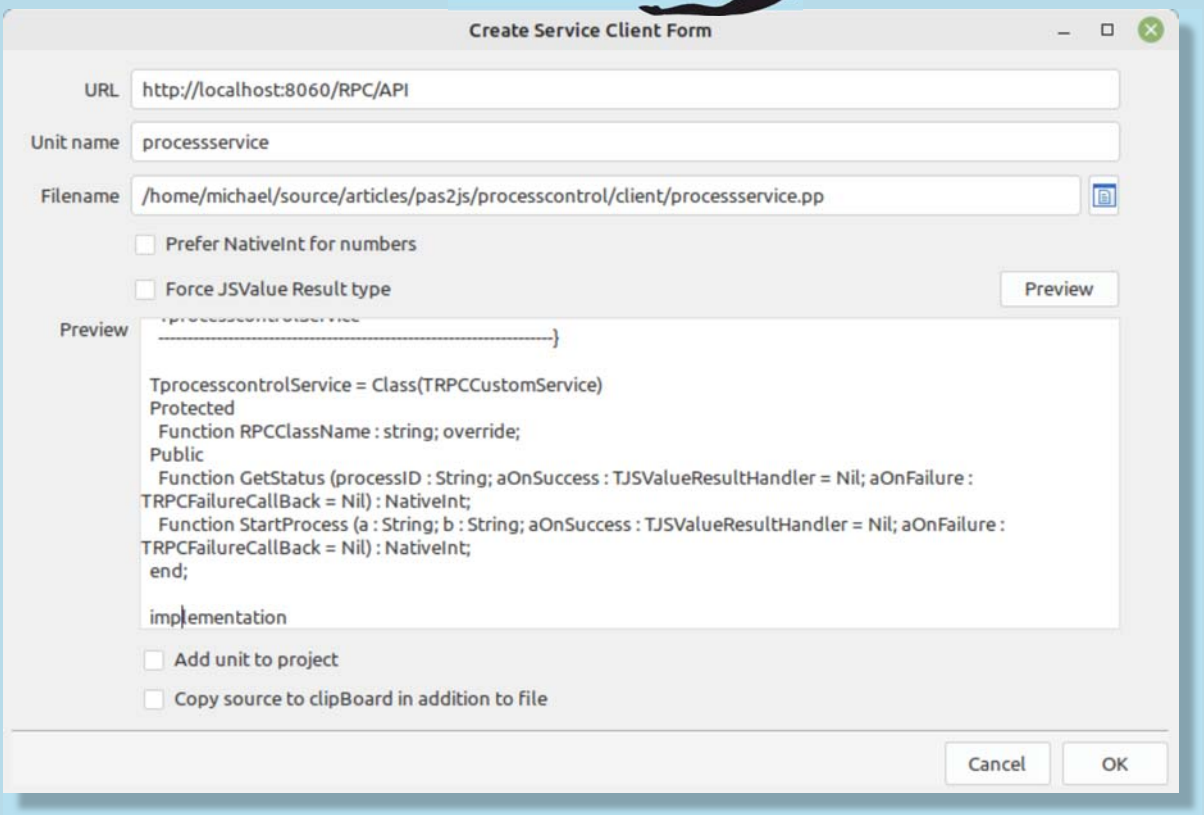


Figure 6: The service generation dialog

Beachten Sie, dass wir den RPC-Client unserer **Dienstdefinition** zuweisen und dass wir allen Event-Handlern der Poller-Komponente Ereignisse zuweisen.

Um den Prozess zu starten, fügen wir der Aktion actbtnStart einen `OnClick-Ereignishandler` hinzu, in dem wir die Werte für die Parameter A und B aus den jeweiligen Eingabefeldern sammeln und diese für den Aufruf von `StartProcess` in unserer **Dienstkomponente** verwenden.

Wir achten darauf, die `OnSuccess`- und `OnFail`-Handler dieser Methode zu behandeln - denken Sie daran, dass die Aufrufe an den Server asynchron sind:

```
procedure TfrmIndex.actbtnStartExecute(Sender: TObject; Event: TJSEvent);

  procedure DoStartFail(Sender: TObject; const aError: TRPCError);
  begin
    Writeln('Failed to start process : ',aError.Message);
  end;

  procedure DoStartOK(aResult: JSValue);
  begin
    FJobID:=String(aResult);
    FPoller.ProcessID:=FJobID;
    FPoller.Start;
  end;

var
  a,b : string;

begin
  a:=actedtA.Value;
  b:=actedtB.Value;
  Service.StartProcess(A,B,@DoStartOK,@DoStartFail);
end;
```

Wenn der Startaufruf fehlschlägt, wird dies einfach protokolliert. Wenn der Startaufruf erfolgreich war, wird das Ergebnis (*eine Prozess-ID*) in der Eigenschaft **ProcessID** des Pollers aufgezeichnet und der Poller gestartet. Der Onclick-Handler für den Button "**Abbrechen**" ist viel einfacher:
Wir müssen nur den Poller abbrechen.

```
procedure TfrmIndex.actbtnCancelExecute(Sender: TObject; Event: TJSEvent);
begin
  Writeln('Canceled wait for process.');
  FPoller.Cancel;
end;
```

Nun müssen nur noch die 4 Ereignisse der Komponente `TProcessCapturePoller` Komponente.
Wir beginnen mit den einfachen, den Ereignissen `OnProcessOutput` und `OnStatusFail`.
Darin, müssen wir nur die Nachrichten ausgeben, die an den Ereignishandler übergeben werden:

```
procedure TfrmIndex.DoStatusFail(Sender: TObject; aError: String);
begin
  Writeln('Error getting status: ',aError);
end;

procedure TfrmIndex.DoDoutput(Sender: TObject; aOutput: String);
begin
  Writeln(aOutput);
end;
```

Der `OnProcessDone`-Ereignishandler ist ebenso einfach:
Wir drucken den Status und den Exit-Code (*falls es einen gibt*)

```
procedure TfrmIndex.DoProcessDone(Sender: TObject;
                                  aStatus: TProcessStatus;
                                  aExitCode: Integer);
Const
    Exits : Array[TProcessStatus] of string
          = ('Running','Exited','Error');
begin
    Write('Process ',Exits[aStatus]);
    if aStatus=psExited then
        Writeln(' with exit code ',aExitCode)
    else
        Writeln();
end;
```

Zu guter Letzt müssen wir das Ereignis `OnGetProcessStatus` behandeln.
Dieses ruft einfach die `GetStatus`-Prozedur unserer Dienstkomponente auf und behandelt die Ergebnis-Handler: in jedem Handler wird die entsprechende Methode der `TProcessCapturePoller`-Komponente mit dem erhaltenen Ergebnis aufgerufen:

```
procedure TfrmIndex.DoGetStatus(Sender: TObject;
                                aProcessID: String;
                                aOffset: NativeInt);

procedure DoStatusFail(Sender: TObject; const aError: TRPCError);
begin
  FPoller.ReportProgressFail(aError.Message);
end;

procedure DoStatusOK(aResult: JSValue);

const statuses : array[Boolean] of TProcessStatus
             = (psError,psRunning);

Var
  D : TJSObject absolute aResult;
  aExitCode : Integer;
  aNewOffset : NativeInt;
  aOutput   : string;
  aStatus   : TProcessStatus;

begin
  aOutput   :=String(D['output']);
  aExitCode :=NativeInt(D['status']);
  aNewOffset :=NativeInt(D['offset']);
  aStatus   :=Statuses[aExitCode=-1];
  FPoller.ReportProgress(aStatus,
           aOutput,aExitCode,aNewOffset)
end;

begin
  Service.GetStatus(FJobID,aOffset,
           @DoStatusOK,@DoStatusFail);
end;
```

Damit ist die Logik unserer Anwendung fertig. Bleibt noch, das Hauptprogramm zu schreiben, die in der Tat sehr kurz ist: Alles, was wir tun müssen, ist unser Modul zu erstellen und `Show` aufzurufen:

```
var
  frm : TfrmIndex;
begin
  MaxConsoleLines:=15;
  frm:=TfrmIndex.Create(Nil);
  Frm.Show;
```

Wenn Sie `MaxConsoleLines` auf 15 setzen, können Sie sicherstellen, dass die Meldungen über den Bildschirm rollen, wenn die Ausgabe des Serverprozesses eintrifft. Das Ergebnis dieses Codes ist in *Abbildung 7 auf Seite 15* des Artikels dargestellt.

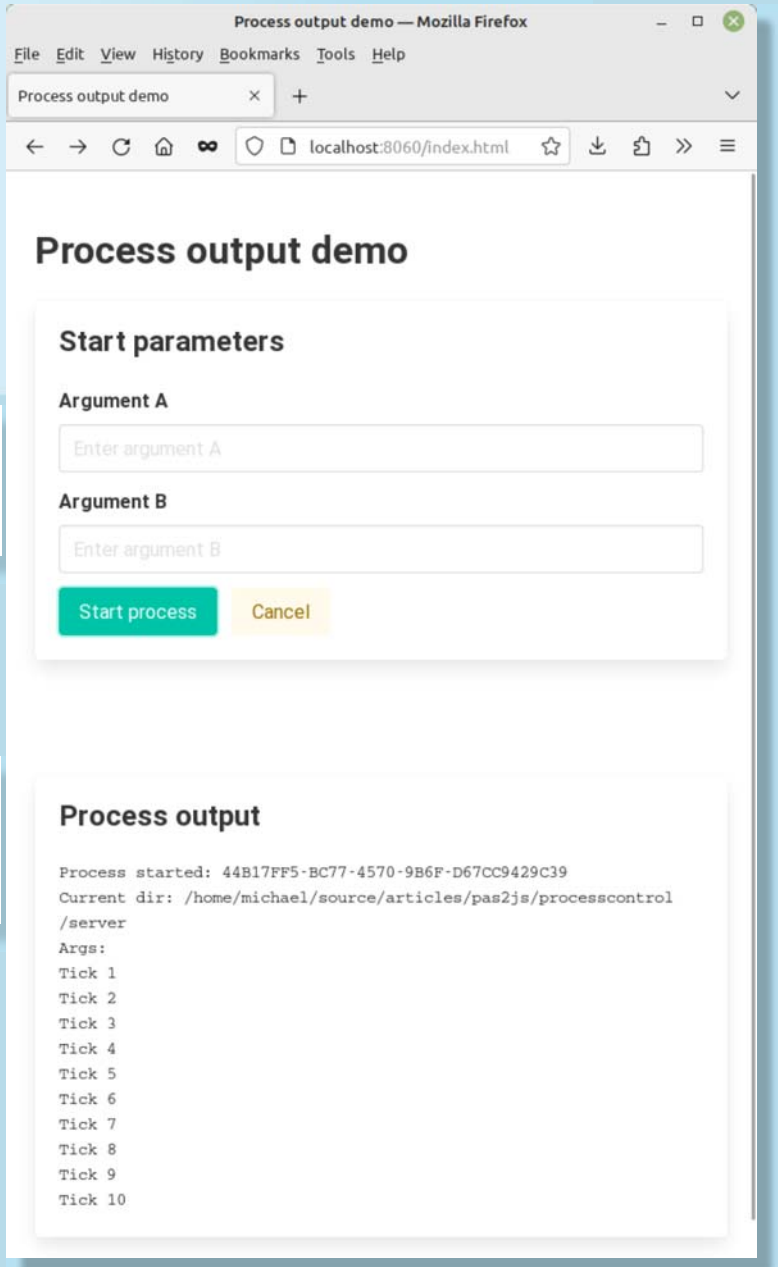


Abbildung 7: Das Programm in Aktion

❽ ERSTELLUNG EINER KOMPONENTE ZUR AUSFÜHRUNG VON SERVERPROZESSEN

Weiter oben im Artikel haben wir erwähnt, dass es seltsam erscheinen könnte, dass es Ereignisse gibt, die den Status und die Ausgabe melden, wenn der eigentliche Aufruf zum Abrufen des **Status im Formularmodul** ausgeführt wird: Zu diesem Zeitpunkt kennen Sie den Status bereits, warum sollten Sie ihn also noch an die Komponente melden?

Ein Teil der Antwort liegt darin, dass das oben Gezeigte nur eine Möglichkeit ist, die Komponente zu verwenden. Eine zweite Möglichkeit besteht darin, dass Sie auch einen Abkömmling dieser Komponente erstellen können, der das Abrufen des Status selbst übernimmt.

In diesem Fall sind die Ereignisse die einzige Möglichkeit, Benachrichtigungen über den Status des Prozesses zu erhalten. Im Folgenden zeigen wir, wie man einen solchen Nachkommen erstellt.
Die Komponente `TProcessCapturePoller` ist eigentlich ein einfacher Abkömmling der Komponente `TCustomProcessCapturePoller`, die lediglich die Methode zum Abrufen des Prozessstatus über ein Ereignis implementiert.

Wir können einen Nachkommen der Komponente `CustomProcessCapturePoller` erstellen, in den die Klasse `TprocesscontrolService` integriert ist. Diese Komponente weiß selbst, wie man einen Prozess auf dem Server ausführt. Diese Komponente würde wie folgt aussehen:

```
TRemoteExecutor = class(TCustomProcessCapturePoller)
Protected
   procedure DoStatusCheck; override;
Public
   Procedure Execute(a,b : String);
Published
   Property RPCClient : TRPCClient Read GetClient Write SetClient;
   Property OnProcessDone;
   Property OnProcessOutput;
   Property OnStatusFail;
   Property LinebasedOutput;
   Property PollInterval;
   Property MaxFailCount;
   Property MaxCheckCount;
end;
```

Wir haben den Konstruktor und den **Destruktor** weggelassen, die einfach den TprocesscontrolService erzeugen und zerstören.

```
constructor TRemoteExecutor.Create(aOwner: TComponent);
begin
  inherited Create(aOwner);
  FService:=TprocesscontrolService.Create(Self);
end;

destructor TRemoteExecutor.Destroy;
begin
  FreeAndNil(FService);
  inherited Destroy;
end;
```

Das Feld Service wird verwendet, um die Eigenschaft `RPCClient` abzurufen und zu setzen:

```
function TRemoteExecutor.GetClient: TRPCClient;
begin
 Result:=FService.RPCClient;
end;

procedure TRemoteExecutor.SetClient(AValue: TRPCClient);
begin
  FService.RPCClient:=aValue;
end;
```

Die Execute-Methode übernimmt die richtigen Parameter und tut im Wesentlichen das, was in unserem ursprünglichen Code in der Form getan wurde:

```
procedure TRemoteExecutor.Execute(a, b: String);

  procedure DoStartFail(Sender: TObject; const aError: TRPCError);
  begin
    SetFailCount(MaxFailCount);
    ReportProgressFail(aError.Message);
  end;

  procedure DoStartOK(aResult: JSValue);
  begin
    ProcessID:=String(aResult);
    Start;
  end;

  begin
    Service.StartProcess(A,B,@DoStartOK,@DoStartFail);
  end;
```

Wenn der Prozess nicht gestartet werden konnte, wird der Fehlerzähler auf den Höchstwert gesetzt, was dazu führt, dass die Methode `ReportProgressFail` keine neue Prüfung plant. Die Methode `DoStatusCheck` enthält lediglich den Code, der in unserer ersten Implementierung im Formular vorhanden war:

```
procedure TRemoteExecutor.DoStatusCheck;

  procedure DoStatusFail(Sender: TObject; const aError: TRPCError);
  begin
    ReportProgressFail(aError.Message);
  end;

  procedure DoStatusOK(aResult: JSValue);

  const statuses : array[Boolean] of TProcessStatus
                 = (psError,psRunning);
Var
    D : TJSObject absolute aResult;
    aExitCode : Integer;
    aNewOffset : NativeInt;
    aOutput   : string;
    aStatus   : TProcessStatus;
  begin
    aOutput   :=String(D['output']);
    aExitCode :=NativeInt(D['status']);
    aNewOffset :=NativeInt(D['offset']);
    aStatus   :=Statuses[aExitCode=-1];
    DoReportProgress(aStatus,aOutput,aExitCode,aNewOffset)
  end;
begin
  service.GetStatus(ProcessID,OutputOffset,@DoStatusOK,@DoStatusFail);
end;
```

Der **Formularcode** ist jetzt viel einfacher. Wir müssen nur noch die Komponente `TRemoteExecutor`-Komponente erstellen und ihre 3 Ereignisse einstellen:

```
procedure TfrmIndex.DataModuleCreate(Sender: TObject);
begin
  FRemote:=TRemoteExecutor.Create(Self);
  FRemote.OnProcessOutput:=@DoDoutput;
  FRemote.OnProcessDone:=@DoProcessDone;
  FRemote.OnStatusFail:=@DoStatusFail;
end;
```

Der Event-Handler für den Button "Start" ist jetzt ein einfacher Einzeiler:

```
procedure TfrmIndex.actbtnStartExecute(Sender: TObject; Event: TJSEvent);

begin
  FRemote.Execute(actedtA.Value,actedtB.Value);
end;
```

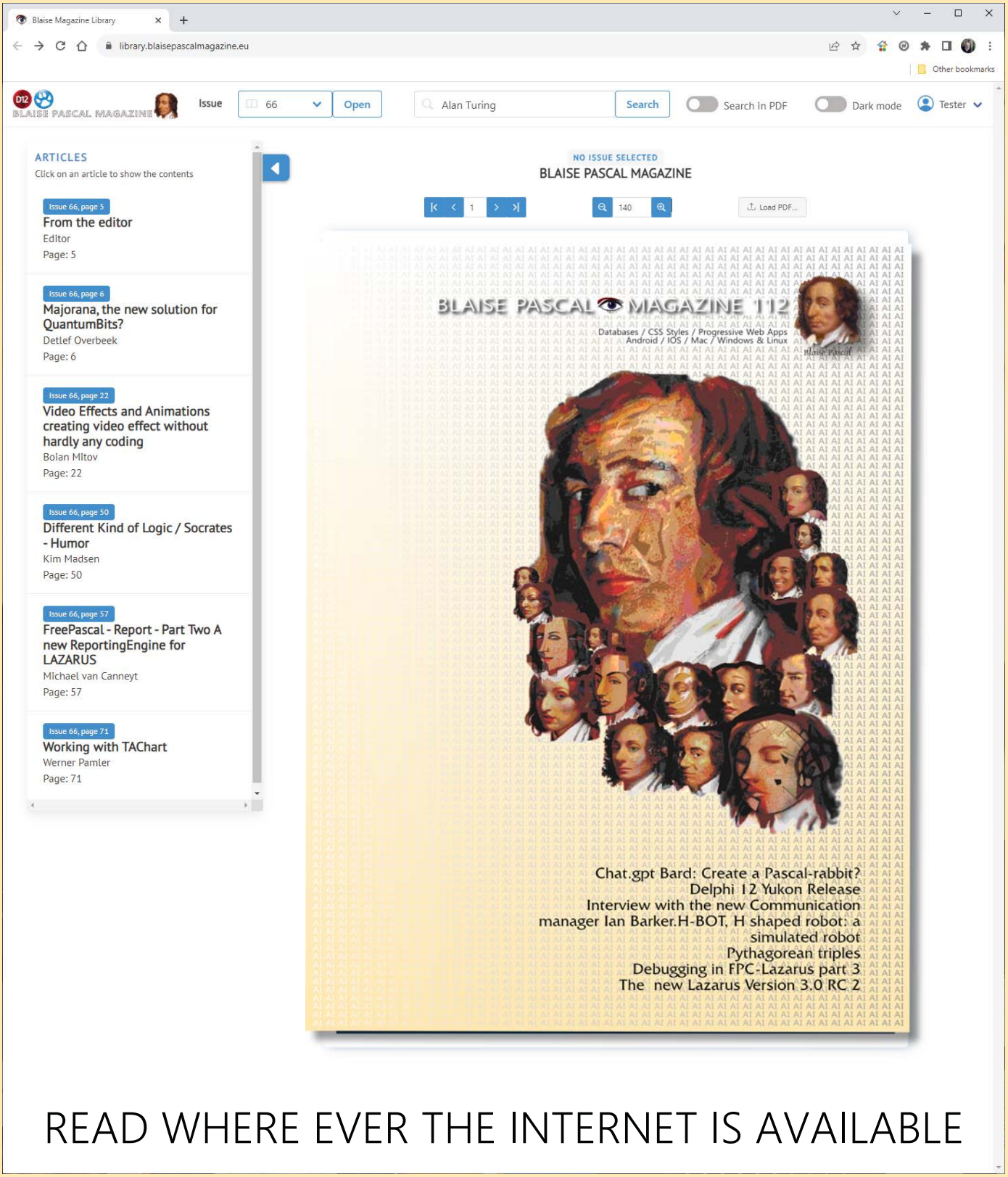
Der Event-Handler zum Abrufen des Status wird nicht mehr benötigt.
Die Funktionsweise des Programms ist nicht anders, aber wenn Sie viele Stellen in Ihrem Programm haben, an denen Stellen in Ihrem Programm haben, an denen Sie Programme auf dem Server ausführen müssen, macht es Sinn, die Remote-Ausführung auf diese Weise zu abstrahieren.

❾ SCHLUSSFOLGERUNG

In diesem Artikel haben wir gezeigt, dass die Ausführung von Programmen auf einem HTTP-Server aus einem Pas2JS-Programm heraus nicht schwierig sein muss. Die Komponente zur Automatisierung des Prozesses ist unabhängig von einem RPC-Mechanismus und kann als solche verwendet werden, oder sie kann als Elternteil für eine komplexere Komponente verwendet werden, die die gesamte Kommunikation selbst abwickelt.

THE NEW INTERNET LAZARUS PDF KIT READER

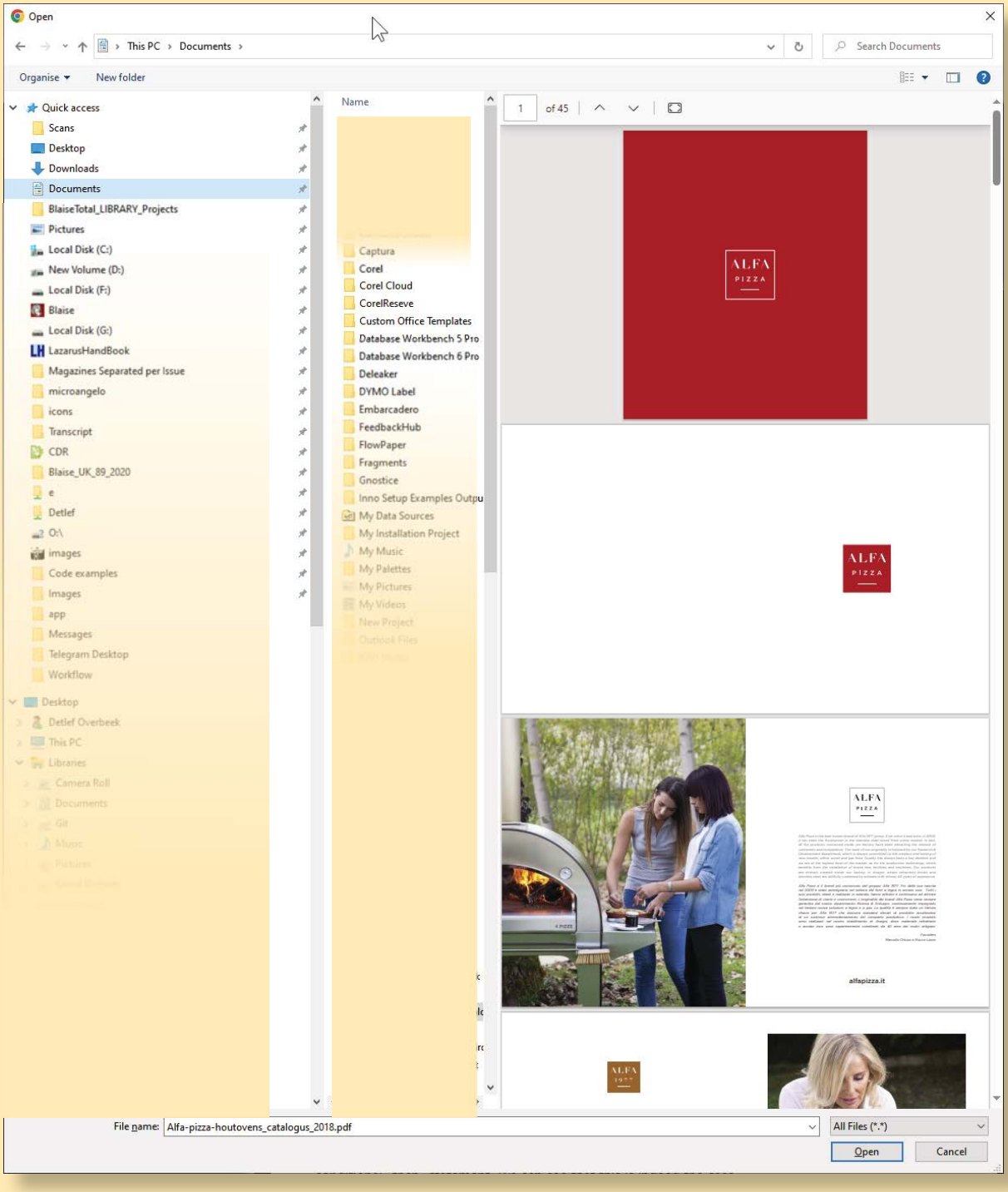
JUST OPEN ANY BROWSER (CHROME, SAFARI, EDGE, FIREFOX, OPERA, DUCKDUCKGO) AND LOGIN: IT WORKS UNDER LINUX / MAC / AND WINDOWS |
OPEN PDF FILES ON YOUR DESKTOP / NOTEBOOK / TABLET OR MOBILE
IF YOU ORDER ANY ITEM YOU WILL RECEIVE THIS READER FOR FREE.
IT IS ONLY A READER | IT CONTAINS NO CODE OR MAGAZINES



READ WHERE EVER THE INTERNET IS AVAILABLE

THE NEW FREE LAZARUS PDF KIT READER

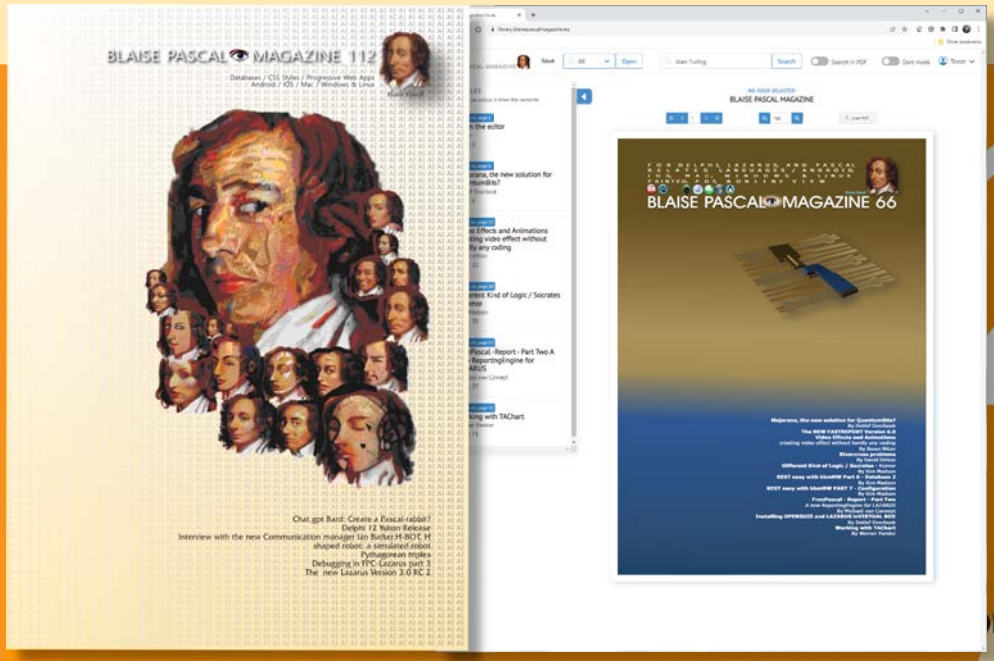
JUST OPEN ANY BROWSER (CHROME, SAFARI, EDGE, FIREFOX, OPERA, DUCKDUCKGO) AND LOGIN: IT WORKS UNDER LINUX / MAC / AND WINDOWS |
OPEN PDF FILES ON YOUR DESKTOP / NOTEBOOK / TABLET OR MOBILE
IF YOU ORDER A BOOK OR ANY ITEM YOU WILL RECEIVE THIS READER FOR FREE.
IT IS ONLY A READER - IT CONTAINS NO CODE OR MAGAZINES



⓬ You can also load any kind of PDF, from your Hard disk, your USB stick or any other source...

READ WHERE EVER THE INTERNET IS AVAILABLE

ADVERTISEMENT



1

2

LAZARUS HANDBOOK

3 POCKET Edition +shipment

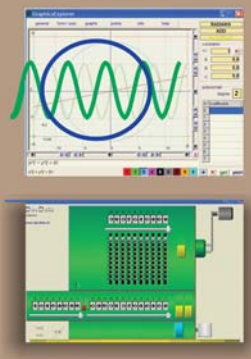
LAZARUS HANDBOOK PDF

4

5

LEARN TO PROGRAM USING LAZARUS
HOWARD PAGE-CLARK

DAVID DIRKSE
including 50 example projects



6

BLAISE PASCAL MAGAZINE
COMPUTER (GRAPHICS) MATH & GAMES IN PASCAL

SUPER PACK 6 ITEMS 2023

1. One year Subscription
2. The newest LIB Stick
   - All issues 1-111
   - On Credit Card
3. Lazarus Handbook Pocket
4. LH PDF including Code
5. Book Learn To Program
   - using Lazarus PDF including 19 lessons and projects
6. Book Computer Graphics Math & Games
   - PDF including ±50 projects

PRICE € 150

NORMAL PRICE € 275

Donate for Ukraine and get a free license at:
https://components4developers.blog/2022/02/26/donate-to-ukraine-humanitarian-aid/

If you are from Ukrainian origin you can get a free Subscription for Blaise Pascal Magazine, we will also give you a free pdf version of the Lazarus Handbook. You need to send us your Ukrainian Name and Ukrainian email address (*that still works for you*), so that it proofs you are real Ukrainian. please send it to `editor@blaisepascal.eu` and you will receive your book and subscription

BLAISE PASCAL MAGAZINE

Multi platform /Object Pascal / Internet / JavaScript / Web Assembly / Pas2Js /
Databases / CSS Styles / Progressive Web Apps
Android / IOS / Mac / Windows & Linux

Blaise Pascal

COMPONENTS 4 DEVELOPERS

Donate for Ukraine and get a free license at:
https://components4developers.blog/2022/02/26/donate-to-ukraine-humanitarian-aid/

COMPONENTS 4 DEVELOPERS

Donate for Ukraine and get a free license at:
https://components4developers.blog/2022/02/26/donate-to-ukraine-humanitarian-aid/

kbmMW Professional and Enterprise Edition v. 5.22.10
kbmMemTable v. 7.98.00 Standard and Professional Edition

5.22.00 is a release with containing new stuff, refinements and bugfixes.,**OpenSSL v3 support,** WebSocket support, further improvements to SmartBind, new high performance hashing algorithms, improved RemoteDesktop sample and much more.

This release requires the use of **kbmMemTable** v. 7.97.00 or newer.

- RAD Alexandria supported
- Win32, Win64, Linux64, Android, IOS 32, IOS 64 and OS X client and server support
- Native high performance 100% developer defined application server
- Full support for centralised and distributed load balancing and fail-over
- Advanced ORM/OPF support including support of existing databases
- Advanced logging support
- Advanced configuration framework
- Advanced scheduling support for easy access to multi thread programming
- Advanced smart service and clients for very easy publication of functionality
- High quality random functions.
- High quality pronounceable password generators.
- High performance LZ4 and J peg compression
- Complete object notation framework including full support for YAML, BSON, Messagepack, J SON and XML
- Advanced object and value marshalling to and from YAML, BSON, Messagepack, JSON and XML
- High performance native TCP transport support
- High performance HTTPSys transport for Windows.
- CORS support in REST/HTML services.
- Native PHP, Java, OCX, ANSI C, C#, Apache Flex client support!

kbmMemTable is the fastest and most feature rich in memory table for Embarcadero products.

- Easily supports large datasets with millions of records
- Easy data streaming support
- Optional to use native SQL engine
- Supports nested transactions and undo
- Native and fast build in M/D, aggregation/grouping range selection features
- Advanced indexing features for extreme performance

- New: full Web-socket support.
  The next release of kbmMW Enterprise Edition will include several new things and improvements. One of them is full Web-socket support.
- New I18N context sensitive internationalisation framework to make your applications multilingual.
- New ORM LINQ support for Delete and Update.
- Comments support in YAML.
- New StreamSec TLS v4 support (by StreamSec)
- Many other feature improvements and fixes.

Please visit http://www.components4developers.com for more information about kbmMW

- High speed, unified database access (35+ supported database APIs) with connection pooling, metadata and data caching on all tiers
- Multi head access to the application server, via REST/AJAX, native binary, Publish/Subscribe, SOAP, XML, RTMP from web browsers, embedded devices, linked application servers, PCs, mobile devices, Java systems and many more clients
- Complete support for hosting FastCGI based applications (PHP/Ruby/Perl/Python typically)
- Native complete AMQP 0.91 support (Advanced Message Queuing Protocol)
- Complete end 2 end secure brandable Remote Desktop with near realtime HD video, 8 monitor support, texture detection, compression and clipboard sharing.
- Bundling kbmMemTable Professional which is the fastest and most feature rich in memory table for Embarcadero products.

COMPONENTS 4 DEVELOPERS