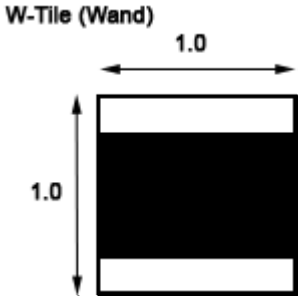
Blitzbasic Tutorial
Robert "bob" Chlewinski März 2006

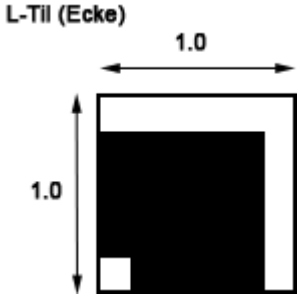
Levelerstellung aus Bitmaps bzw. Mini 3D Mapeditor
Schwierigkeitsgrad: Einsteiger
Ziel: Quick and dirty Level Erstellung für Games, Test- und Demozwecke
Download aller Resourcen unter: http://www.iomagic.de/bb/Bitmap_to_3dLevel.zip

Wir das o.G. Ziel schon andeutet geht es in diesem Tut. Um das möglichst schnelle erstellen von Levels. Wir Programmieren eine kleine Engine die uns anhand eines einfachen Bitmaps einen kompletten Level für Labyrinthartige Spiele wie z.B. Rollenspiele (RPG) generiert. Die Methode Farbwerte aus einem Bitmap auszulesen und an die entsprechenden Koordinaten 3D Objekte zu erstellen ist natürlich ein alter Hut. Oft werden dabei einfache Cubes mittels „CreateCube()“ erstellt und Positioniert. So simpel und praktische diese Methode auch ist, so groß sind auch ihre Nachteile. Wie wir ja wissen sind Cubes würfelartige Objekte die aus sechs Seiten bestehen. Zieht man nun beispielsweise eine Wand von 10 Cubes entlang einer Achse, ergeben sich insgesamt 60 Seiten. Die sich berührenden Seiten sowie die Unterseiten bekommt der Spieler aber eigentlich nie zu Gesicht. Das sind 28 Seiten insgesamt und somit fast 50% Geometriedaten mehr als benötigt. Wir verschwenden Speicherplatz und Rechenpower. Weiter Probleme können beim Texturieren Auftreten. Von der Legosteinartigen Optik die dabei herauskommt ganz zu schweigen. Was wir also für einen halbwegs ansprechenden Maze-Level brauchen ist ein kleines, Optimiertes 3D Tileset.

Um also eine größeren Anreiz zu geben das Tutorial auch durchzuziehen, habe ich mir erlaubt dieses 3D Tilset im b3d Fileformat zu erstellen. Die Tiles sind, bis auf die Bodenplatte, allesamt Mauerteile die im Zentrum eines Bodentiles stehen. Ein Bodentile entspricht dabei der BlitzBasic 3D Standarddimension von 1*1. Die Mauerteile haben jedoch eine geringere Tiefe (dicke) was erfordert die Tiles entsprechend auszurichten. Um dies besser zu verdeutlichen hier die Tiles (von oben gesehen) auf einer Bodenplatte stehend.

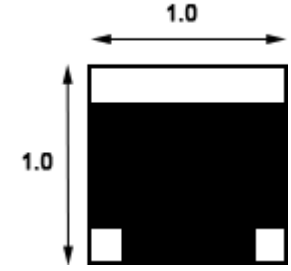


Das W-Tile (Wand) erstreckt sich über die gesamte Breite einer Bodenplatte. Brauch nur entweder um 90° rotiert werden oder bleibt ohne Rotation.



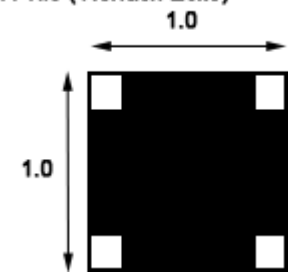
Das L-Tile ist eine Ecke. Es wird entsprechend um -90,90 oder 180° gedreht.

T-Till (Doppelecke)



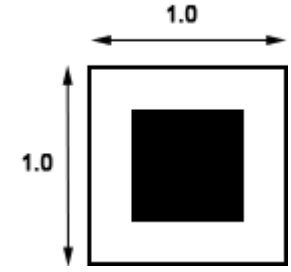
Das T-Tile muss ebenfalls um -90,90 oder 180° gedreht werden. Es wird dann verwendet wenn 2 Ecken benötigt werden

X-Tile (Vierfach Ecke)



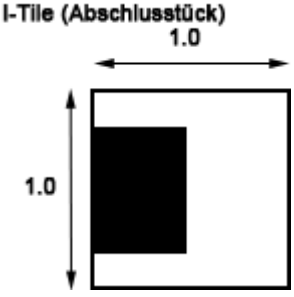
Das X-Tile findet immer bei 4er Ecken Verwendung. Es muss natürlich nie ausgerichtet werden sonder wird ausschließlich positioniert.

S-Tile (Säule)



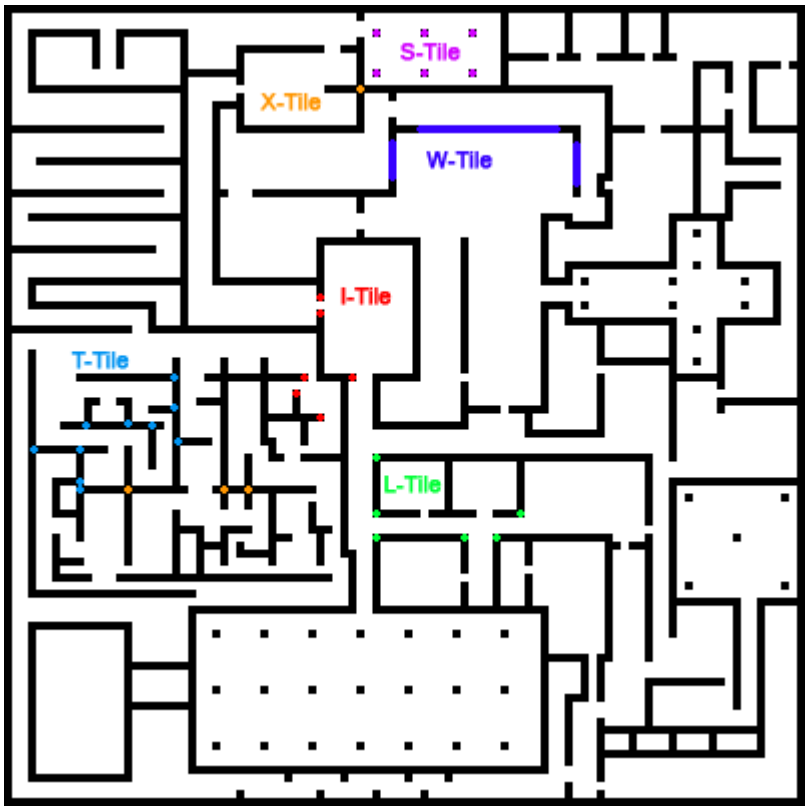
Auch das S-Tile (Säule) Wird nur positioniert. Es findet immer dann Verwendung wenn ein schwarzer Pixel keine schwarzen „Nachbarn“ hat.

I-Tile (Abschlusstück)



Dem I-Tile (Abschlusstück) kommt besondere Bedeutung zu. Da bei allen Tiles außer dem I-Tile die Stirnseite(n) entfernt wurde, muss die Geometrie abgeschlossen werden da man sonst sieht dass alle Tiles hohl sind. Das I-Tile findet aber nur dann Verwendung wenn normalerweise
1. Eine Wand gesetzt wurde
2. Die Wand nur einen Nachbarn hat.
Dies hat auch zur Folge das schmale Durchlässe auf unserer Map mindestens 2 Tiles Breit werden.

Hier der Demolevel der generiert werden soll und wie die Tiles Verwendung finden sollen
(Level1.bmp original 100*100 Pixel hier vergrößert)



An allen schwarzen Pixelpositionen sollen die Passend en Tils gesetzt werden.
(Die Map wurde mittlerweile leicht modifiziert. Es existieren nun noch ein paar Durchgänge mehr)

Um sich die Tiles in Ruhe alleine für sich zu betrachten liegt im Ordner „Media“ des Zip Files ein kleines Programm, auf das ich hier nicht näher eingehen möchte, im Quellcode vor. „media_browser.bb“ (Selbsterklärend nach dem Start.)

Keep it Simpel:

Wir fangen mal ganz einfach an. Wir Initialisieren BB, Laden unsere Demolevel Bitmap und bringen diese zur Anzeige.

Info: Im weiteren Verlauf werde ich immer nur neue hinzuzufügenden Code hier einfügen und besprechen. Ihr findet zu jedem Stadium einen separaten Quellcode der immer alle bis dahin enthaltenen Teile beinhaltet.

(bitmap to Level_01.bb)

```
Const ScreenX = 1024
Const ScreenY = 768
Graphics3D ScreenX,ScreenY,16,1

Dim Map(0,0)
Global MapW
Global MapH
Global LevelImg = LoadImage("media/level1.bmp")
Load_level()

;Mainloop ***************************************************************************
While Not KeyHit(1)
      RenderWorld
      DrawBlock LevelImg,screenx-200,0
      Flip
Wend
End

Function Load_level()
   SetBuffer ImageBuffer(levelimg)
   mapw = imagewidth(levelimg)-1
   maph = imageheight(levelimg)-1
   Dim map(MapW,MapH)
   For y=0 To maph
      For x=0 To mapw
         GetColor x,y
         If ColorRed() = 0   And ColorGreen() = 0   And ColorBlue() = 0   Then  map(x,y)=1
      Next
   Next
  SetBuffer BackBuffer()
End Function
```

Ich denke dass keiner ernsthafte Schwierigkeiten mit dem Code hat.
Die Auflösung kann jeder Einstellen wie er möchte. (ScreenX, ScreenY). Bitte verwendet aber dazu die beiden Konstanten da wir später noch einige Berechnungen damit durchführen müssen. Die Funktion Load_Level() liest das Bitmap ein und befüllt das Dimfeld Map() mit einer 1 an X,Y Position wenn ein schwarzes Pixel mittels „GetColor“ ermittelt wurde.
Da dies ein Einsteiger Tut. ist habe ich hier auf ReadPixel bzw. ReadPixelFast verzichtet. Die Funktion ist nicht Zeitkritisch da sie nur einmal, am Anfang des Programms, ausgeführt wird.
Etwas verwundern mag uns vielleicht „RenderWorld“ im Mainloop. Da wir noch keine 3D Objekte verwenden ist der Aufruf eigentlich vollkommen Überflüssig. Er wurde aber mit Absicht nicht entfernt um noch mal zu zeigen das 2D Grafikaufrufe, wie der nachgestellte „DrawBlock „ Befehl, nach RenderWorld erfolgen müssen da sonst 2D Inhalte überschrieben werden. Wen der Code ausgeführt wird, mag es zu leichtem Flackern kommen. Dies liegt ebenfalls an RenderWorld das nichts zu rendern hat da weder Kamera noch eine „World“ existieren. Das soll uns aber vorerst nicht weiter stören. Mehr als 1 Sekunde brauch man eh nicht um zu sehen wo das Bitmap auf dem Screen platziert wird. Die etwas merkwürdige Position hat einen Grund. Später dazu mehr. Drückt nun ESC um das Programm zu beenden.

Tiles laden. (bitmap to Level_02.bb)

Um alle Tiles zu laden verwenden wir die Funktion „Load_Tiles()“. Vorher aber deklarieren wir einige Globale Variablen die die Mesh’s aufnehmen sowie 2 Dim Hilfsfelder die uns später bei der „Nachbarschaftspflege“ noch dienlich sein werden.
Die Deklaration kann nach z.B. Graphics3D erfolgen.

```
Dim dirMX(3) ;Dimfeld für X-Verschiebung der 4 Himmelsrichtungen
Dim dirMZ(3) ;Dimfeld für Z-Verschiebung der 4 Himmelsrichtungen
Global  X_Tile , L_Tile , W_Tile ,  T_Tile , I_Tile , S_Tile ,Ground
Load_Tiles ;Funktionsaufruf nach der Deklaration


Function Load_Tiles()
	X_Tile = LoadMesh("media\X_tile.b3d")
	L_Tile = LoadMesh("media\L_tile.b3d")
	W_Tile = LoadMesh("media\W_tile.b3d")
	T_Tile = LoadMesh("media\T_tile.b3d")
	I_Tile = LoadMesh("media\I_tile.b3d")
	S_Tile = LoadMesh("media\S_tile.b3d")
	Ground = LoadMesh("media\ground.b3d")

	f# = 1.0
	YScale# = 1.5
	ScaleMesh X_Tile, f,YScale#,f
	ScaleMesh L_Tile, f,YScale#,f
	ScaleMesh W_Tile, f,YScale#,f
	ScaleMesh T_Tile, f,YScale#,f
	ScaleMesh I_Tile, f,YScale#,f
	ScaleMesh S_Tile, f,YScale#,f
	ScaleMesh Ground, f,0,f

	HideEntity X_Tile
	HideEntity L_Tile
	HideEntity W_Tile
	HideEntity T_Tile
	HideEntity I_Tile
	HideEntity S_Tile
	HideEntity Ground

	;Norden
	dirMX(0) = 0
	dirMZ(0) = 1

	;Osten
	dirMX(1) = 1
	dirMZ(1) = 0

	;Süden
	dirMX(2) = 0
	dirMZ(2) = -1

	;Westen
	dirMX(3) = -1
	dirMZ(3) = 0

End Function
```

Wie man sieht passiert hier nichts was weiter spannend wäre. Meshes werden geladen, etwas nach oben skaliert (nur Y Achse) und anschließend mit HideEntity versteckt.

Die Befüllung der beiden Dimfelder mag das einzige erklärenswerte sein.
Im spätern Verlauf werden wir zu jedem 1er Wert des Map(x,y) Feldes (Entspricht einem schwarzen Pixel) in alle vier Himmelsrichtungen schauen ob weitere 1er Werte vorhanden sind. Durch Kombination der Anzahl wie viel „Nachbarn“ und wo diese gefunden wurden lässt sich eindeutig klären welches Till wo positioniert und rotiert werden muss um nahtlos mit den andern abzuschließen. Wen wir beispielsweise nach Osten schauen wollen fragen wir
dirMX(1) und dirMY(1) ab da wir uns vielleicht beim auswerten auf der Map bei x29, y31 befinden wissen wir dann das unsere Nachbar Prüf Koordinaten 29+1 und 31+0 sind. Dies lässt sich natürlich auch „Hart“ Codieren macht den Code aber nicht unbedingt Lesbarer.
Das Programm 02 zu starten könnt ihr euch im Übrigen schenken. Es wird sich auf dem Bildschirm nichts ändern.

Der Spas beginnt: (bitmap to Level_02.bb)

```
Global Pivot = CreatePivot()
Cam = CreateCamera(Pivot)
Global Lab = CreateMesh()
Create_3D_Level()
```

Kamera erstellen und an einen Pivot hängen.
Danach Ein Mesh erstellen (Lab, global) das später unseren ganzen Level oder Teile davon aufnehmen kann.
Anschließend die Funktion Create_3D_Level aufrufen. Ich persöhnlich versehe Funktionsaufrufe immer mit klammern auch wen das nicht notwendig ist. Man kann einen Funktionsaufruf so noch einfacher als solche identifizieren.

```
Function OffMap(X,Y)
    If x >MapW Or x < 0 Or y > mapH Or y < 0 Then Return True
    Return False
End Function
```

Eine Hilfsfunktion die dafür sorgt das wir bei Koordinaten Überprüfungen nicht versehentlich auf einen „Array Index out of Bounds“ Error laufen. Da die Funktion „OffMap“ heist liefert sie Wahr (True) wen die Koordinaten die an die Funktion Übergeben werden NICHT auf der Map sind.

```
Type Block
    Field mesh
    Field Name$
    Field x,z
End Type
```

Ein Benutzerdefinierter Type der unseren einzelnen Tiles aufnimmt. (Der Name Block ist dabei für mich historischer Natur da ich zuerst auch mir „LegoBlocks“ (Cubes) gearbeitet habe.

```
Function Create_3D_Level()
    For z = 0 To MapH
    For x =  0 To MapW
        If map(x,z) = 1  ;Block setzen
            B.Block = New Block
            ;auf Nachbar Block Prüfen
            nCount = 0
            tString$ = ""
            xCount = 0
```

```
            zCount = 0
            For N = 0 To 3
                NX = x+dirMX(N)
                NZ = z+dirMZ(N)
                If offMap(nx,nz) = False Then
                    If map(NX,NZ) = 1 Then ;NAchbar
                        ncount = ncount +1
                            Select N
                                Case 0
                                    tString = tstring + "N"
                                    zCount = zCount +1
                                Case 1
                                    tString = tstring + "O"
                                    xCount = xCount +1
                                Case 2
                                    tString = tstring + "S"
                                    zCount = zCount +1
                                Case 3
                                    tString = tstring + "W"

                                    xCount = xCount +1
                            End Select
                        EndIf
                    EndIf
                Next
DebugLog "Tile " + x + " " + z + " hat Nachbarn " + nCount +"  " +tstring +" x z count "+ xcount + " " +zcount
                ;Auswertung des Tiles
                Select nCount ;Anzahl der Nachbarn
                    Case 0                  ;Eine Säule = S Tile
                        b\mesh = CopyMesh(S_Tile)
                        b\name = "Saule"
                    Case 1                  ;Ein Endstück = I  Defaultausrichtung ist w
                        b\mesh = CopyMesh(I_Tile)
                        b\name = "I Tile"
                        If tString = "N" Then Rotatemesh b\mesh,0,-90,0
                        If tString = "O" Then Rotatemesh b\mesh,0,180,0
                        If tString = "S" Then Rotatemesh b\mesh,0,+90,0
                    Case 2                  ;Eine Ecke oder Mauer
                        If xCount = 1 And ZCount = 1 Then
                            b\mesh = CopyMesh(L_Tile)    ;default =sw
                            b\name = "L Tile"
                            If tString = "NO" Then Rotatemesh b\mesh,0,180,0
                            If tString = "OS" Then Rotatemesh b\mesh,0,90,0
                            If tString = "SW" Then Rotatemesh b\mesh,0,0,0
                            If tString = "NW" Then Rotatemesh b\mesh,0,-90,0
                        Else
                            b\mesh = CopyMesh(W_Tile)
                            b\name = "W Tile"
                            If tString = "NS" Then Rotatemesh b\mesh,0,90,0
                        EndIf
                    Case 3                  ;Ein T-Tile
                        b\mesh = CopyMesh(T_Tile)  ;Defaultrichtung ist "OSW"
                        b\name = "T Tile"
                        If tString = "NOS" Then Rotatemesh b\mesh,0,90,0
                        If tString = "NOW" Then Rotatemesh b\mesh,0, -180,0
                        If tString = "OSW" Then Rotatemesh b\mesh,0, 0,0
                        If tString = "NSW" Then Rotatemesh b\mesh,0,-90,0
                    Case 4                  ;Ein X Tile
                        b\mesh = CopyMesh(X_Tile)
```

```
                        b\name = "X Tile"
                End Select

                PositionMesh b\mesh,x,0,z
            EntityPickMode b\mesh,2
            b\x = x
            b\z = -z
            NameEntity b\mesh, Handle(b)
             EndIf
        Next
    Next

    For z = 0 To MapH
        For x =  0 To MapW
                Temp = CopyMesh(Ground)
                PositionMesh Temp,x,0,z
                AddMesh temp,lab
                FreeEntity temp
        Next
    Next

    For b.block = Each block
        Temp = CopyMesh(b\mesh)
        AddMesh temp,lab
        HideEntity b\mesh
        FreeEntity temp
    Next

    ;Startposition per Zufall bestimmen
.again
    pX = Rand(1,mapW-1)
    pZ = Rand(1,mapH-1)
    If map(px,pz) = 1 Then Goto again
    PositionEntity pivot,px,2,pz

End Function
```

```
Function Create_3D_Level()
    For z = 0 To MapH
    For x =  0 To MapW
        If map(x,z) = 1  ;Block setzen
            B.Block = New Block
            ;auf Nachbar Block Prüfen
            nCount = 0
            tString$ = ""
            xCount = 0
            zCount = 0
```

Die Funktion wird mit einer Doppelschleife (Z,X) eingeleitet. Nachfolgender Code wird danach nur ausgeführt wen auf der Map(X,Z) eine 1 ist. Trifft dies zu erstellen wir erst mal einen neuen Block.
Welcher dies sein wird und wie er positioniert wird ist abhängig von Nachfolgenden Kontrollvariablen,
nCount (Anzahl der gefundenen Nachbarn)
tString$ (Ein Stringvariable die uns in Kurzform Aufschluss über die Himmelsrichtungen gibt)
xCount (Anzahl der Nachbarn in X Richtung)

zCount (Anzahl der Nachbarn in Z Richtung)

Ursprünglich sah der Code dafür ganz anders aus. Ich habe in aber wieder abgeändert da ich der Meinung bin das Anfänger damit besser zurechtkommen werden. Die Variable nCount ist eigentlich auch überflüssig da es letztendlich die Summe von XCount+zCount ist.
Den ganz unbedarften sei hier erklärt dass wir die uhrsrprüngliche Y Koordinate des Bitmaps in Z Koordinate des 3D Raums umwandeln. Wir wollen unsern Level ja in Räumlich tiefe bauen und nicht nach oben. Dass diese Methode allerdings einen kleinen Schönheitsfehler aufweist soll uns hier noch nicht stören. Ich komme später auf das kleine Problem zurück. Hier aber dennoch ein Denkanstoss. Ist ein sich erhöhender Y Wert eines Bitmaps(wir lesen nach unten aus und die Zahl steigt also) gleich dem Z Wert von 3D Koordinaten wo wir bei steigenden werten nach „hinten“ arbeiten?

Nachbarn finden:

```
For N = 0 To 3
    NX = x+dirMX(N)
    NZ = z+dirMZ(N)
    If offMap(nx,nz) = False Then
        If map(NX,NZ) = 1 Then ;NAchbar
            ncount = ncount +1
            Select N
                Case 0
                    tString = tstring + "N"
                    zCount = zCount +1
                Case 1
                    tString = tstring + "O"
                    xCount = xCount +1
                Case 2
                    tString = tstring + "S"
                    zCount = zCount +1
                Case 3
                    tString = tstring + "W"
                    xCount = xCount +1
                End Select
        EndIf
    EndIf
Next
DebugLog "Tile " + x + " " + z + " hat Nachbarn " + nCount +"  " +tstring +" x z count "+ xcount + " " +zcount
```

Nun kommen unsere Hilfsfelder zum Einsatz. Eine weiter Schleife von 0-3 liest Werte zwischen -1 und 1 aus den Dim Felder dirMX(z) aus die addiert werden mit den aktuellen x,z Koordinaten (die Doppelschleife ist ja noch aktiv) . Das Ergebnis sind die zu überprüfenden 4 Himmelsrichtungen der aktuellen Koordinate. Nur wen diese neuen Koordinaten im gültigen Map Bereich liegen (offMap Funktion) findet die Befüllung der Kontrollvariablen statt. Als weiter Bedingung muss natürlich auch an den neuen Prüf Koordinaten eine 1 auf der Map sein. Ansonsten hätten wir eine weiße Stelle erwischt und somit kein Nachbarn gefunden. Der Nachbarzähler (nCount) wird alsdann um 1 hoch gesetzt.
Über eine Select Case Bedingung deren Kriterium der aktuelle N wert (0-3) sein kann. kann in Textform festgehalten werden in welcher Himmelsrichtung wir grade geschaut haben. 0 =Norden (entspricht einer Erhöhung der aktuellen X,Z Adresse um +0, -1 auf der Map(x,z) usw. die Variable tString kann somit werte von z.B. NOS (Norden Süden Osten) ausspucken. Das bedeutet dass wir an der aktuellen Position ein T-Tile verwenden müssen. (3 Nachbarn). Da die Abfrage im Uhrzeigersinn verläuft ist eine Kombination von z.B. „SWN“ nie möglich. Auf Norden wird immer zuerst geprüft und daher kann Norden auch immer ausschließlich die erste Himmelsrichtung sein (wen sie den überhaupt vorkommt).
Das Debuglog gibt genauen Aufschluss über alle gefunden Nachbarn.

```
    ;Auswertung des Tiles
        Select nCount ;Anzahl der Nachbarn
            Case 0                  ;Eine Säule = S Tile
                b\mesh = CopyMesh(S_Tile)
                b\name = "Saule"
            Case 1                  ;Ein Endstück = I  Defaultausrichtung ist w
                b\mesh = CopyMesh(I_Tile)
                b\name = "I Tile"
                If tString = "N" Then Rotatemesh b\mesh,0,-90,0
                If tString = "O" Then Rotatemesh b\mesh,0,180,0
                If tString = "S" Then Rotatemesh b\mesh,0,+90,0
            Case 2                  ;Eine Ecke oder Mauer
                If xCount = 1 And ZCount = 1 Then
                    b\mesh = CopyMesh(L_Tile)    ;default =sw
                    b\name = "L Tile"
                    If tString = "NO" Then Rotatemesh b\mesh,0,180,0
                    If tString = "OS" Then Rotatemesh b\mesh,0,90,0
                    If tString = "SW" Then Rotatemesh b\mesh,0,0,0
                    If tString = "NW" Then Rotatemesh b\mesh,0,-90,0
                Else
                    b\mesh = CopyMesh(W_Tile)
                    b\name = "W Tile"
                    If tString = "NS" Then Rotatemesh b\mesh,0,90,0
                EndIf
            Case 3                  ;Ein T-Tile
                b\mesh = CopyMesh(T_Tile)  ;Defaultrichtung ist "OSW"
                b\name = "T Tile"
                If tString = "NOS" Then Rotatemesh b\mesh,0,90,0
                If tString = "NOW" Then Rotatemesh b\mesh,0, -180,0
                If tString = "OSW" Then Rotatemesh b\mesh,0, 0,0
                If tString = "NSW" Then Rotatemesh b\mesh,0,-90,0

            Case 4                  ;Ein X Tile
                b\mesh = CopyMesh(X_Tile)
                b\name = "X Tile"
            End Select

            PositionMesh b\mesh,x,0,z
            EntityPickMode b\mesh,2
            b\x = x
            b\z = z
            NameEntity b\mesh, Handle(b)
        EndIf
    Next
```

Eine erste grobe Auswertung kann nun schon anhand der Anzahl der Nachbarn stattfinden.

Select nCount ;Anzahl der Nachbarn

Haben wir keine Nachbarn, erstellen wir eine Säule
Haben wir 1 Nachbar, setzen wir ein Abschlussstück (I-Tile)
Bei 2 Nachbarn könnten wir auf eine Ecke oder ein durchgehende Mauer gestoßen sein.
Die Frage ist leicht geklärt. Ein durchgehende Mauer bedingt das beide Nachbarn sich entweder auf der X oder Z Achse befinden. Alles andere müssen Ecken sein. (Ich hab's hier aber andersrum gelöst und erstmal auf eine Ecke geprüft)

Bei 3 und 4 Nachbarn ist es wieder sehr einfach. 3 Nachbarn immer eine T-Tile, 4 Nachbarn ein X-Tile. Innerhalb der Select Anweisung wird natürlich auch direkt bestimmt in welche Richtung unser kopiertes Mesh rotiert wird. Da uns ja die Default Ausrichtungen bekannt sind (Siehe Tile Grafiken oben) ist es leicht Anhand der tString Variable genau zu bestimmen um wie viel Grad das Tile gedreht werden muss damit es an die Nachbarn anschließt.
Gibt uns der tString bei einem L-Tile z.B NO zurück müssen die Schenkel des „L's" auf Norden und Osten ausgerichtet werden. Da die Defaultrichtung nach Süden und Westen deutet, können wir das Tile einfach um -180 oder +180 Grad drehen.

```
    For z = 0 To MapH
        For x =  0 To MapW
                Temp = CopyMesh(Ground)
                PositionMesh Temp,x,0,z
                AddMesh temp,lab
                FreeEntity temp
        Next
    Next

    For b.block = Each block
        Temp = CopyMesh(b\mesh)
        AddMesh temp,lab
        HideEntity b\mesh
        FreeEntity temp
    Next
;Startposition per Zufall bestimmen
.again
    pX = Rand(1,mapW-1)
    pZ = Rand(1,mapH-1)
    If map(px,pz) = 1 Then Goto again
    PositionEntity pivot,px,2,pzEnd Function
```

Nachdem alle Nachbarn geprüft und alle Tiles in ein Field Mesh des Types Block kopiert und rotiert wurden, ist der Level Grundsätzlich Fertig. Was noch fehlt sind die Bodenplatten die Einfach sooft Kopiert werden wie das Bitmap Pixel hatte. Um nicht mit 10.000 Surfaces zu hantieren (100*100) werden die Bodenplatten direkt in das Trägermesh LAB per AddMes überführt.
Dasselbe Passiert auch mit jedem Mesh das in einem Block Type ist. Das Mesh des Blocks wird hierbei allerdings erstmal nur per Hideentity versteckt. Vielleicht wollen wir den Level später ja noch mal neu zusammensetzen weil ein wütender Bombenleger im Level ein großes Loch gesprengt hat.
Die Startposition des Players bzw. Kamera wir hier einfach per Zufall aus der Map(X,Z) gewählt.
Der Vorgang wird solange wiederholt bis die pX,pY Koordinaten auf ein Mapfeld zeigen welches nicht geblockt (wert 1) ist.

Das Tutorial sollte eigentlich an dieser Stelle zu Ende sein. Da wir uns aber z.Zt. weder bewegen können noch all zuviel sehen, habe ich mir erlaubt noch einige kleine goodies einzubauen die aber nicht Bestandteil des Tut's sind. Ich werde den Quellcode hier nicht Auflisten sondern nur kurz besprechen.

Die Funktion Cam_Control:
Sorgt dafür das wir uns „Dungenmaster 3D" mäßig im Level Bewegen können. Das originale Bitmap zur Erstellung des Levels wird dabei auch noch gleichzeitig als MiniMap zur Orientierung verwendet. (Nebst Kopie). Ein paar Sounds werden, annähernd zu den Schritten die wir machen, abgespielt. Ein Kompass aus 2 Sprites zusammen gedengelt dient als weiters Orientierungsmittel. Er wird einfach auf die Kamera „geklebt"
Als einzige Lichtquelle dient uns ein Punktlicht das wir per Zufallsgenerator manipulieren um ansatzweise eine Fackel zu simulieren. Noch ein bisschen Kameravodoo und dann war's das auch schon.
Die Taste F12 macht uns ausnahmsweise keinen Screenshot dafür wird aber der ganze Level als B3D Objekt gespeichert. Kopiert ihn anschließend einfach in den Media Ordner und überprüft ihn mit dem media_browser.

Anmerkungen:

Achtet beim verwenden eigener Bitmaps zur Levelerstellung auf exakte schwarz (0,0,0) und Weiswerte (255,255,255).

Giles quittierte das Laden des Demolevels mit einem MA Fehler. Vielleicht ist der Level einfach zu groß. Tests mit kleineren Leveln funktionierten auch in Giles.

Wer mag kann die einzelnen Surfaces nachträglich im Code selber Texturieren.
Mesh Lab Surface 1 = Boden Surface
Mesh Lab Surface 2 = Mauer Surface

Eine Beispielanwendung für dieses Programm könnte z.B. sein kleine, einzelnen Abschnitte zu generieren die anschließend als b3d gespeichert werden um „Super Tiles“ zu erzeugen.
Man kann so komplette Gänge oder einzelne Räume generieren und separat texturieren, und beleuchten.

Wer mag kann ja noch ein Dach auf den Level setzen. Natürlich könnt ihr auch eigene Tiles mit z.B. noch geringerer Wandstärke erstellen. Achtet aber beim Texturieren darauf dass ihr eine bessere Textur verwendet als ich. (Sie passt nicht ganz Nahtlos aneinander da diese Textur nur ein Ausschnitt aus eine größeren Textur ist.)

Im der Hauptschleife (Mainloop) steckt der Code für die F12 Taste zum speichern des Levels als B3D.
Ich wechsele mit Absicht zuvor mittels ChangeDir(„media“) in den Media Ordner. Ansonsten würde die Export Funktion in das B3D File die Pfadangaben der Texturen mit einem vorangestellten „\media\“ versehen.
Bewahrt eure Texturen deshalb immer zusammen mit dem B3d File auf.

Dieses Tutorial liegt im Zipfiele als PDF Dokument dabei.

Steuerung:

(bitmap to Level_final.bb)

Esc = Ende
Cursor links, rechts und rauf
F12 Save Level to b3d

Viel Spaß und ich hoffe es bringt euch weiter.
Bob

Nachtrag:

Leider habe ich vergessen etwas wichtiges zu erklären. Beim Einlesen der Bitmap Datei in das Map(x,z) Feld wird ja mittels Doppelschleife von oben(Z oder besser Y) nach unten und von rechts nach Links (x) eingelesen. Das Dimfeld entspricht somit einem einfachen Bitfeld mit Ja/nein Zustände (0 oder 1) das sowohl für den Aufbau des Maze als auch zur Kollisionskontrolle verwendet werden kann. In der spätern Funktion „Create_3D_Level()“ wird in einem weitern durchlauf durch das DimFeld Map(x,z) eine Tile positioniert. Die Position entspricht dabei der aktuellen Adresse (X,Z). Dies hat zur Folge das am ende der Funktion das Maze in zwar in Positiven Z Koordinaten liegt, das aber die Tiefe vertauscht ist. Ist an Position x4,z4 z.B. eine Säule (ein einzelner schwarzer Pixel) hat dieser zwar die richtigen Koordinaten aber Blicken wir in Richtung Z Achse aus einer erhöhten Position auf das Maze erkennen wir das das Maze Spiegelverkehrt in Z Richtung erzeugt wurde. Um diesem Problem entgegenzuwirken seien hier 2 Tricks verraten.

1. Erstellt die Tiles an der Z Position aber setzt den Z Wert in negative indem ihr einfach ein Minus vor die Z Koordinate setzt. PositionEntity b\mesh,x,0,-z. Nach Erzeugung des Maze könnt ihr dann das ganze Maze per PositionEntity Lab,0,0,Meshdepth(LAB) so verschieben das der Level wieder mit der Map(X,Z) übereinstimmt.

2. Der in diesem Tut angewandte Trick besteht darin das Bitmap mittels ScaleImage LevelImg,1-1 in Y Richtung zu spiegeln. Ist das Dimfeld Map(x,Z) dann befüllt, kann die Skalierung rückgängig gemacht werden.