Scripting

Die Spieltisch-Logik wird über das Scripting mittels der Programmiersprache Visual Basic realisiert. Visual Basic ist - zumindest in den Bereichen, die Sie hier benötigen - relativ einfach zu erlernen, da die Spieltisch-Logik im Wesentlichen auf "wenn ... dann ..."-Abfragen oder zu durchlaufende Schleifen basiert. Dabei ist das Scripting objektorientiert, das heist: jedes Objekt in Future Pinball wird über eine eigene Sub-Routine angesprochen, die nur den Code enthält, der für die Abfrage oder Steuerung bezüglich dieses Objektes nötig ist.
Es würde den Rahmen dieses Tutorials sprengen, wenn ich die Syntax des gesamten Scriptings im Detail abhandeln würde. Deshalb beschränke ich mich hier darauf, nur dort detailierter auf das Script einzugehen, wo dieses für den Anfänger schwieriger nachzuvollziehen ist. Das Script ist im übrigen komplett durchkommentiert, so dass dieses gleich an Ort und Stelle erklärt wird. Öffnen Sie den Spieltisch und lassen Sie sich durch Klick auf die entsprechende Schaltfläche in der linken Werkzeugleiste das Script anzeigen.
Da ich bei den Erklärungen im Folgenden auf die Zeilennummern Bezug nehme, sollten Sie, sofern noch nicht geschen, die Ansicht der Zeilen-Nummern einschalten (Menü ***View*** / ***Line Numbers***). Desweiteren wird der Spielablauf eines Spieltisches über weite Bereiche über Timer gesteuert. Sie sollten sich deshalb also mit diesem Objekt vertraut machen. Die Beschreibung des Timer-Objektes finden Sie **hier**. Timer werden immer dann eingesetzt, wenn zwischen einem und dem nächsten Schritt ein zeitlicher Zwischenraum bestehen soll.
Entgegen der Reihenfolge der einzelnen Routinen im Standard-Script eines neuen Spieltisches habe ich diese bei meinem Beispiel-Spieltisch im Sinne des logischen Ablaufes geändert, so dass wir dieses hier Block für Block durchgehen können.
Das Script beginnt in der Zeile 14 mit der Anweisung ***Option Explicit***, die sagt, dass alle Variablen deklariert (angemeldet) werden müssen. Der Sinn liegt darin, dass Sie auf einen Laufzeitfehler, dessen Ursache beispielsweise in einem Tipfehler begründet ist, durch das Debuggen aufmerksam gemacht werden können. Sie bekommen dann eine Fehlermeldung, die Aufschluß über die relevante Zeile und den Fehler informiert. Darüber hinaus behalten Sie damit auch eine Übersicht über die mitunter große Anzahl an Variablen und Flags. Deshalb sollten Sie diese Anweisung immer beibehalten.

Deklaration der Variablen - ab Zeile 16:

Hier werden die verwendeten **Variablen deklariert** und mit beschreibenden Kommentaren versehen. Sie sehen hier zwei verschiedene Varianten: Mit ***Const*** können Sie der Variablen dierekt einen bestimmten Wert zuweisen, der sich während des Spieles nicht mehr verändert, mit ***Dim*** wird die Variable einfach angemeldet und deren Inhalt im Verlauf des Spieles über das Script bestimmt. Feldvariablen (wir kommen später darauf zurück) müssen mit der Anzahl der enthaltenen Felder in Klammern angegeben werden.

Sub FuturePinball_BeginPlay() - ab Zeile 80:

Diese Sub-Routine wird als erstes aufgerufen, wenn Sie einen Spieltisch für ein Spiel starten. Hier werden die diversen **Initialisierungen** der Displays, Lichter und globalen Variablen durchgeführt.

Die Initialisierung der Lichter (Zeile 123 bis Zeile 131) möchte ich hier näher erklären: Sowohl die Bonus-Lichter als auch die anderen Lichter sollen/müssen im Script während des Spieles "automatisiert" angesprochen werden können, wenn z. B. bei Ball-Ende der Bonus heruntergezählt wird. Aus diesem Grund packen wir die Bezeichnung der 10 Bonuslichter in eine Feldvariable, die aus 10 einzelnen Feldern besteht, die einfach nummerisch angesprochen werden können: Mit der Anweisung

```
Set Bb_Bonus(1) = Bb_Bonus1
```

weisen wir dem ersten Speicherplatz der Feldvariablen ***Bb_Bonus()*** das erste der 10 Bonuslichter zu, nämlich ***Bb_Bonus1***. Mit

```
Set Bb_Bonus(2) = Bb_Bonus2
```

belegen wir den zweiten Speicherplatz der Feldvariablen ***Bb_Bonus()*** mit dem zweiten Bonuslichtes ***Bb_Bonus2***, usw. Mit den anderen Lichtern verfahren wir gleichermaßen, wobei hier die Feldvariable einfach ***Bulb()*** heißt. Auf die praktische Anwendung für dieses Vorgehen werden wir später zurückkommen.

Sub FuturePinball_KeyPressed(ByVal KeyCode) - ab Zeile 151:

Diese Routine behandelt die **Tastatur-Eingaben** während des Spieles beim Drücken einer Taste. Als erstes habe ich da einige Tasten so belegt, dass mit diesen bestimmte Objekte auf dem Spieltisch angesprochen werden können. Dies simuliert das Hit-Ereignis, welches beim Treffen der Kugel auf ein Ziel auftritt und kann zum Testen verwendet werden:

```
If KeyCode = 23 Then LeftInLaneTrigger_Hit()
```

KeyCode 23 ist der Tasten-Code für die ***i-Taste***. Wenn diese gedrückt wird, wird die Sub-Routine aufgerufen, die das Hit-Ereignis des linken Inlane-Triggers behandelt. So können Sie diese Funktion testen, ohne dass Sie während des Spieles darauf warten müssen, bis die Kugel die Inlane passiert. Damit diese Test-Tasten funktionieren, müssen Sie die ***constTest***-Variable oben bei der Deklaration auf ***TRUE*** setzen. Die Tasten-Codes für die einzelnen Tasten finden Sie im Future Pinball Manual unter ***Scripting*** / ***Keycodes*** aufgelistet.
Erklärungsbedarf besteht wohl auch für die Zeilen 163 bis 167: In dieser kleinen Funktion spreche ich die Droptargets der 4er Droptarget-Bank einen nach dem anderen durch Tastendruck an. Dabei mache ich mir zunutze, dass jeder Droptarget in der Bank eine interne ID-Nummer besitzt, bei meiner Droptarget-Bank sind das die IDs 1 bis 4. ***KeyCode 32*** ist die Taste für das **D**. Mit

```
x = x + 1: If x > 4 Then x = 1
```

bekommt x zunächst den Wert 1. Mit jedem Tastendruck wird dieser um 1 erhöht. Wenn der Wert größer als 4 ist, soll er wieder 1 betragen. Wir können x somit als Platzhalter für die ID nehmen und so die Droptargets ansprechen. Mit der Zeile

```
DropTarget4Bank.Popdown(x)
```

wird der Droptarget mit der ID x 'angeschossen' so dass dieser versenkt wird. Danach wird mit der Zeile

```
DropTarget4Bank_Hit()
```

der Code abgearbeitet, der das eigentliche Handling beim Treffen des Targets bestimmt. Mit dieser Funktion haben wir somit auch die Möglichkeit zu testen, ob das Handling bei den verschiedenen Resets der Bank richtig funktioniert. Alle anderen Tasten-Abfragen betreffen die bei den Einstellungen bzgl. der Tastenbelegung definierten Spieltasten und sind in einer Funktion vordefiniert. An diesen brauchen Sie in der Regel nichts zu verändern.
Im einzelnen betrifft dies das Einwerfen einer Münze und der damit verbunden Vergabe des Credits, die sich auf das entsprechende Display auswirkt, so dass dieses aktualisiert werden muss. Auch wenn der Spieler die Taste für das Zurückziehen des Plungers drückt, wird hier abgehandelt. Alsdann wird unterschieden, ob sich der Flipper gerade in einem Spiel befindet und nicht getilt ist. Hier befinden sich die Anweisungen, die beim drücken der Flipper-Tasten erfolgen sollen und was passieren soll, wenn der Spieler die Taste zum Starten des Spieles drückt.

Sub FuturePinball_KeyReleased(ByVal KeyCode) - ab Zeile 151:

Analog zur KeyPressed-Routine wird hier das Loslassen einer gedrückten Taste behandelt. Dies betrifft die Taste für den Plunger und die beiden Flipper-Finger.

Sub PlungerTimer_Expired() - ab Zeile 350:

Dieser Timer dient dazu, dass der Plunger beim Drücken der enter-Taste langsam zurückgezogen wird. Er wird in der KeyPressed-Routine in Zeile 192 über den Befehl

```
If (KeyCode = GetKeyCode(PlungerKey)) Then
PlungerTimer.Enabled = True
```

gestartet. Alsdann wiederholt er den Code, so lange die enter-Taste gedrückt bleibt. Dabei zieht er den Plunger 5 %-weise zurück, bis er maximal 100 % erreicht hat. Zuerst wird mit

```
PlungerPercentage = PlungerPercentage + 5
```

der Weg definiert, um den der Plunger bei einem Timer-Durchlauf zurückgezogen wird. Die nächste Zeile stellt sicher, dass 100 % dem maximalen Weg entsprechen.

```
If PlungerPercentage > 100  Then PlungerPercentage = 100
```

Die Zeile 353 gibt dann die Anweisung, den Plunger um den Weg zurückzuziehen, der mit ***PlungerPercentage*** definiert wurde:

```
Plunger.Pull(PlungerPercentage)
```

Da der Plunger durch das loslassen der enter-Taste die Kugel abschießt, wird der Timer logischer weise in der KeyReleasd-Routine gestopt (Zeile 318).

Sub FuturePinball_Tilted() - ab Zeile 359:

Wenn die globale Variable ***fpTilted*** den Wert TRUE hat, wird diese Sub-Routine ausgeführt. Sie spielt einen entsprechenden Sound, lässt die Flipper 'abfallen' und lässt den Flipper mit dem Ausschalten aller Lichter reagieren. Gleichzeitig werden Bonus und Bonuspunkte durch das Zurücksetzen auf '0' gelöscht, da diese im Tilt-Status des Flippers nicht gezählt werden. Als letztes wird ab Zeile 385 der TiltRecovery-Timer gestartet, der, nachdem die Kugel im Drain angekommen ist, den Flipper wieder in den Normalzustand zurücksetzt.

Sub TiltRecoveryTimer_Expired() - ab Zeile 394:

Der Wiederherstellungs-Timer, der den Flipper aus dem TILT-Status zurücksetzt, wartet bis die letzte Kugel im Drain gelandet ist, bevor das Spiel fortgesetzt wird. Er wird gestartet, sobald der Flipper in den TILT-Status gerät.

Sub FuturePinball_NameEntryComplete(ByVal Position, ByVal Special) - ab Zeile 409:

Diese Sub-Routine wird automatisch aufgerufen, nachdem die Initialien bei einem eventuellen Highscore eingegeben wurden. Hier wird auch ein evtl. neuer Highscor in der globalen Variablen nvR1 gespeichert, der im Attract-Modus im Wechsel mit den letzten Punktesständen wechselt (Zeile 413).

Sub ResetForNewGame() - ab Zeile 431:

In dieser Routine wird der Spieltisch für ein neues Spiel initialisiert. Hier wird zunächst durch das Ausschalten aller Lichter der Attract-Modus gestopt. Bei diesem Vorgang kommen die Feldvariablen zum Einsatz, die ich bei den Initialisierungen angesprochen habe. Da ich die Namen aller Lichter incl. Bumper der Veldvariable ***Bulb()*** zugeordnet habe, kann ich die Lichter jetzt über 3 Zeilen Code einfach ausschalten. Die Anweisung in Zeile 440 gibt vor, dass die folgende Schleife 18 mal durchlaufen wird. Dabei werden in der Variablen ' ***i*** ' die Durchläufe gezählt und dieser Wert bei jedem Durchlauf um 1 erhöht.

```
For i = 1 to 18
```

Dadurch kann ' ***i*** ' verwendet werden, um den entsprechenden Speicherplatz in der Feldvariablen und demit das diesem Platz zugeordnete Licht anzusprechen. Next läßt die Schleife anschließend wiederholen, bis ' ***i*** ' den Wert 18 hat.

```
For i = 1 to 18
  Bulb(i).State = BulbOff
Next
```

Die gleiche Vorgensweise kommt auch für die Bonus-Lichter zum Einsatz, die der Feldvariablen ***Bb_Bonus()*** zugeordnet wurden.

Im nächsten Schritt werden ab Zeile 448 die Displays für die Punkte vorbereitet. Diese werden auf 0 zurückgesetzt und zunächst im Display des ersten Spielers und im HUD-Display der Text '00' angezeigt (mehrere Befehle können durch einen : (Doppelpunkt) getrennt in einer Zeile stehen. Die globale Variable ***nvTotalGamesPlayed*** zählt in Zeile 455 die Anzahl der insgesamt gespielten Spiele auf diesem Spieltisch. Sie können sich diese anzeigen lassen, indem Sie den Spieltisch im Debug-Modus starten und den Wert über

```
AddDebugText "gesamt gespielte Spiele " & nvTotalGamesPlayed
```

ausgeben. Dieser wird dann im Debug-Fenster gezeigt. Es folgen verschiedene Initialisierungen, deren Bedeutung im Script gut kommentiert sind und deshalb hier nicht weiter erläutert werden müssen. Zuletzt wird in der Zeile 477 mit der Anweisung

```
FirstBallDelayTimer.Interval = 1000
```

die Wartezeit zwischen dem Starten des Spieles bis zum Auswurf der Kugel definiert, in diesem Fall 1000 Millisekunden (= 1 Sekunde) und mit

```
FirstBallDelayTimer.Enabled = TRUE
```

der Timer gestartet.

Sub FirstBallDelayTimer_Expired() - ab Zeile 485:

Nach dem Abluaf dieses Timers werden die Routinen zum Reset für einen neuen Ball und das erstellen einer neuen Kugel aufgerufen.

Sub CreateNewBall() - ab Zeile 494:

Der PlungerKicker befindet sich unter dem Apron seitlich zur Plungerlane. Dieser erstellt nun eine neue Kugel und wirft diese anschließend über die ***.SolenoidPuls*** Methode aus. Die Variable ***BallsOnPlayfield*** erhält hier den Wert **1**. Dies wird benötigt, wenn der Flipper getilt wurde und der Wiederhestellungs-Timer darauf wartet, dass die Kugel im Drain gelandet ist, womit diese Varible auf 0 gesetzt wird, so dass dieser das Tilt-Flag zurücksetzen kann.

Sub PlungerKicker_Hit() - ab Zeile 504:

Hin und wieder kommt es vor, dass eine Kugel beim Verlassen der Spielfläche über den Drain durch den Kanal bis in den PlungerKicker läuft und darin liegen bleibt. Damit die Kugel dann aber nicht darin verweilen muss, wird diese über die .SolenoidPulse Methode des Kickers in dieser Sub-Routine wieder in dei Plungerlane ausgestoßen.

Sub PlungerLaneTrigger_Hit() - ab Zeile 511:

Diese Sub setzt einzig und alleine das Flag bBallInPlungerLane auf den Wert TRUE. Dieses Flag wird nur benötigt, damit ein Sound abgespielt werden kann, wenn der Plunger losgelassen wird und sich eine Kugel in der Lane davor befindet (TRUE).

Sub ResetForNewPlayerBall() - ab Zeile 518:

Jedes mal wenn ein neuer Ball startet, wird diese Sub-Routine durchlaufen, dies gilt bei meinem Beispiel-Flipper auch für den Extra Ball. In der ***Select Case*** Abfrage (ab Zeile 525) werden zuerst die Punkte-Displays für die verschiedenen Spieler initialisiert. Dabei wird unterschieden, ob der Spieler bereits Punkte erspielt hat oder ob es sich um dessen ersten Ball handelt. Der Sinn dieser Initialisierung liegt darin, dass das Display des aktuellen Spielers blinkt, bis die ersten Punkte addiert werden. Dies dient letztlich dazu, zu signalisieren, welcher der Spieler die nächste Kugel spielt. Die Anweisung

```
Display1.QueueText "00", seBlink, 999999, 0, False, ""
```

veranlasst das Display des ersten Spielers, die Anzeige mit '00' blinkend für einen Zeitraum von ca. 16 Minuten (999999 Millisekunden) darzustellen. Dabei wird entsprechend der Parameter der Effekt sofort gestartet und die Anzeige nach Beendigung des Effektes nicht gelöscht. Die Dauer des Blinkens besteht jedoch nur so lange, bis die ersten Punkte gezählt werden.
Wenn bereits Punkte erspielt wurden, enthält der Befehl anstelle des '00'-Textes die globale Variable nvScore(CurrentPlayer) des betreffenden Spielers, die dessen erspielte Punkte enthält:

```
Display1.QueueText nvScore(CurrentPlayer), seBlink, 999999, 0,
False, ""
```

Als nächstes wird ab Zeile 565 das Display für die Anzeige des Balles im Spiel initialisiert. Dabei wird auf die Variable ***BallsRemaining*** (ausstehende Kugeln) des aktuellen Spielers zurückgegriffen. Dieser Variablen wurde beim Reset für ein neues Spiel der Wert der globalen Variablen ***nvBallsPerGame*** übergeben. Da ***BallsRemaining*** nach jedem Ball-Ende um 1 reduziert wird, kann deren Wert über eine ***Select Case*** Abfrage für die Anzeige herangezogen werden. Wenn ***BallsRemaining*** beispielsweise den Wert 1 hat, und der Flipper mit drei Bällen pro Spiel gespielt wird, dann wurden schon zwei Bälle gespielt und der das Spiel befindet sich beim dritten Ball und im Display wird '3' angezeigt.

```
Select Case BallsRemaining(CurrentPlayer)
   Case 3
     DisplayBall.Text = 1
   Case 2
     DisplayBall.Text = 2
   Case 1
     DisplayBall.Text = 3
End Select
```

Ab Zeile 574 werden dann die Lichter über eine **FOR** ... **NEXT** Schleife zunächst ausgeschaltet, wie schon beim Reset für ein neues Spiel erklärt und anschließend die benötigten Lichter eingeschaltet. Dies sind die beiden Lichter der oberen Lanes, die beiden Lichter der oberen linken und rechten Droptarget-Bänke und die 3 Bumper. Zuletzt werden noch die Droptarget-Bänke resettet und ab Zeile 600 einige

Variablen und Flags zurückgesetzt, deren Bedeutung im Script hinreichend dokumentiert sind.

Sub Drain_Hit() - ab Zeile 629:

Diese Sub wird dann aufgerufen, wenn der Spieler die Kugel über den Drain verliert. Hier passiert nichts weltbewegendes: Die Kugel wird "vernichtet", die Variable ***BallsOnPlayfield*** auf '0' gesetzt und der zum _Hit-Eereignis passende Sound gespielt. Abschließend erfolgt der Aufruf der Sub ***EndOfBall***.

Sub EndOfBall() - ab Zeile 629:

Hier wird zunächst das Flag ***bOnTheFirstBall*** auf ***FALSE*** gesetzt. Dies resultiert darin, dass von diesem Zeitpunkt kein neuer Spieler zum Spiel hinzugefügt werden kann, da der erste Ball für diesen nicht mehr spielbar ist.
Die ab Zeile 638 folgenden Befehle werden nur ausgeführt, wenn sich der Flipper nicht im TILT-Status befindet. Hier werden zuerst die Bonuspunkte für den aktuellen Spieler bestimmt. Da ich die Variable ***Bonus*** lediglich zum Behandeln der Bonuslichter verwende und diese somit nur einen Werte von 1 bis 10 enthalten kann, muss dieser zuerst mit 1000 multipliziert werden, um die Bonuspunkte für das addieren zu den Punkten zu erhalten. Das Ergebnis wird der Variablen ***BonusPoints(CurrentPlayer)*** übergeben, was als Code so aussieht:

```
BonusPoints(CurrentPlayer) = (Bonus * 1000)
```

Nun erfolgt in Zeile 644 das Starten des ***CheckPointsTimers***, der, bevor der Bonus gezählt wird, prüft, ob noch ausstehende Punkte gezählt werden müssen. Gleichzeitig wird der Variablen ***BonusDelayTime*** die Wartezeit zwischen Ballverlust und dem Zählen des Bonus zugeordnet.
Falls der Flipper getilt wurde, wird der Timer, der das Fortsetzen des Spieles nach dem Zählen des Bonus behandelt, sofort gestartet, so dass das Zählen des Bonus übersprungen wird.

Sub CheckPointsTimer_Expired() - ab Zeile 662:

Der Code dieses Timer wird so lange durchlaufen, bis keine ausstehenden Punkte mehr gezählt werden müssen. Durch die Anweisung

```
CheckPointsTimer.Enabled = True
```

wird dieser immer wieder neu gestartet. Erst wenn die Inhalte der Variablen ***ScoreCurrent1000***, ***ScoreCurrent100*** und ***ScoreCurrent10*** = 0 sind

```
If ScoreCurrent1000 = 0 And ScoreCurrent100 = 0 ... Then
  CheckPointsTimer.Enabled = False
  BonusTimer.Enabled = True
End If
```

(die Bedeutung dieser Variablen werden später erkärt), wird der ***CheckPointsTimer*** gestopt und der Timer zum Abzählen des Bonus gestartet.

BonusTimer_Expired() - ab Zeile 673:

In dieser Sub-Routine wird der gesammelte Bonus zum Punktestand des aktuellen Spielers addiert. Die erste Zeile der Sub sorgt zunächst dafür, dass der Timer nach dem abarbeiten des Codes immer wieder neu gestartet wird. Dann wird in Zeile 676 überprüft, ob noch Bonus zu zählen ist: Wenn nicht (BonusPoints(CurrentPlayer) = 0), dann wird der Timer gestopt, die nächste Routine aufgerufen und der Timer verlassen.
Über die Select Case Abfrage ab Zeile 682 wird der weitere Ablauf zunächst danach unterschieden, ob der Bonus einfach oder doppelt gezählt werden soll, da die Handhabung leicht unterschiedlich ist. Wenn es sich um einfachen Bonus handelt (BonusMultiplier(CurrentPlayer) = 1), wird zuerst der Interval für die Wiederholung des Zählvorgangs bestimmt und es werden 1000 Punkte addiert, die anschließend von der Bonussumme abgezogen werden.

```
BonusTimer.Interval = 200
AddScore(1000)
PlayMusic 2, "1000"
BonusPoints(CurrentPlayer) = BonusPoints(CurrentPlayer) – 1000
```

Nun folgt die Lichtsteuerung: Wie anfangs bei der Initialisierung der Lichter schon beschrieben, sind die Namen aller 10 Bonuslichter in einer Feldvariablen namens ***Bb_Bonus()*** abgelegt, wobei die Zahl innerhalb der Klammern die Feldnummer angibt, die angesprochen werden soll. Da der Bonus insgesamt 19000 Punkte betragen kann und dies mit 2 Lichtern auf der Bonusleiter dargestellt wird, erfolgt zunächst eine Abfrage nach der Höhe des Bonus. Ist dieser höher als 10000 Punkte, wissen wir, dass 2 Lichter leuchten und es muss zuerst das unterste der beiden Lichter behandelt werden.
Angenommen ***Bonus*** hat den Wert 16, dann leuchten das 10000er Licht und das 6000er Licht. Mit dem Befehl

```
Bb_Bonus(Bonus - 10).State = BulbOff
```

wird in der Feldvariablen ***Bb_Bonus()*** der Speicherplatz 6 angesprochen (16 - 10 = 6). Dies entspricht dem 6000er Bonuslicht, welches dann ausgeschaltet wird. Anschließend wird ***Bonus*** um 1 reduziert.

```
Bonus = Bonus – 1
```

Er beträgt somit 15. Mit der Abfrage und Anweisung

```
If Bonus > 10 Then Bb_Bonus(Bonus - 10).State = BulbOn
```

wird zunächst sichergestellt, dass der Bonus immer noch größer als 10 ist, da sonst das Ergebnis Bonus - 10 negativ ausfällt und zu einem Fehler führt. Anschließend

wird das 5000er Licht eingeschaltet: (Bonus - 10) entspricht 15 - 10 = 5. Wenn der Bonus 10 oder weniger beträgt, kommt der Code nach dem ***Else***-Teil zur Anwendung. Die Handhabung dabei ist im Prinzip die gleiche wie vor beschrieben. Im Fall eines Doppelbonus ist die Hanhabung bezüglich der Lichter die gleiche, jedoch unterscheidet sich das Vorgehen beim Vergeben der Punkte: Damit pro 1000 Punkte des gesamten Bonus 2000 Punkte vergeben werden, habe ich mich eines Flags namens ***FlagMultiplier*** bedient, dessen Inhalt nach jeder Addition von 1000 Punkten um 1 erhöht wird. Erst wenn der Wert dieses Flags 2 beträgt, wird der Bonus dann um 1000 Punkte reduziert, das aktuelle Licht ausgeschaltet und das nächst niedrigere Licht eingeschaltet. Danach wird der Bonus-Timer gestopt und der ***BonusBreakTimer*** aufgerufen (Zeile 727), der eine kleine Pause zwischen 2000 und den nächsten 2000 Punkten herstellt.

Sub BonusBreakTimer_Expired() - ab Zeile 721:

Wie schon gesagt, dient dieser Timer im wesentlichen dazu, eine kleine Pause zwischen jeweils 2000 Punkten beim Doppelbonus einzufügen. Dabei wird das ***FlagMultiplier*** wieder auf 0 zurückgesetzt und anschließend der Bonus-Timer neu gestartet.

Sub EndOfBallTimer_Expired() - ab Zeile 729:

Nachdem der Bonus abgezählt wurde, aber auch wenn sich der Flipper im TILT-Status befindet, wird der weitere Spielverlauf hier fortgesetzt. Der Timer wird zunächst mit EndOfBallTimer.Enabled = False daran gehindert, den folgenden Code zu wiederholen. Alsdann wird die Future Pinball Variable ***fpTilted*** auf ***FALSE*** zurückgesetzt, damit sich der Flipper wieder im Normalzustand befindet, falls er getilt wurde.
Ab Zeile 742 weicht das Verhalten meines Beispiel-Flippers von dem anderer mir bekannter Future Pinball Spieltischen ab, die beim Auswurf des Extra Balls in die Plungerlane auch gleich das ShootAgain-Licht ausschalten. Dies entspricht jedoch nicht dem realen Verhalten von Flippern. Bei echten Flippern blinkt das ShootAgain-Licht, um zu signalisieren, dass der selbe Spieler einen weiteren Ball spielen kann. Das Licht erlischt erst, wenn die ersten Punkte gezählt werden. Dieses Verhalten habe ich auch für meinen Beispiel-Flipper implementiert.
Wenn es sich bei dem neuen Ball um einen Extra Ball handelt,

```
If (ExtraBallsAwards(CurrentPlayer) = 1) Then
   FlagExtraBall = True
```

wird lediglich ein Flag namens ***FlagExtraBall*** auf ***TRUE*** gesetzt, welches später benötigt wird, um das ShootAgain-Licht blinken zu lassen bzw. auszuschalten. In den folgenden Zeilen wird der Spieltisch dann durch den Aufruf der Reset-Routine ganz normal für einen neuen Ball zurückgesetzt und die neue Kugel in die Plungerlane ausgestoßen. Allerdings blinkt das ShootAgain-Licht nun, da beim Reset für einen neuen Ball das ***FlagExtraBall*** abgefragt und entsprechend reagiert wurde. Liegt kein Extra Ball vor, wird über die Zeile

```
BallsRemaining(CurrentPlayer) = BallsRemaining(CurrentPlayer)
- 1
```

die Anzahl der verbleibenden Bälle um 1 reduziert, damit beim neuen Ball das Display 'Ball in Play' angepasst werden kann. Danach wird überprüft, ob dies der letzte Ball im Spiel für den aktuellen Spieler war. Trifft dies zu, wird die Routine zum Eingeben der Initialien für einen eventuellen neuen Highscore aufgerufen. Wenn weitere Kugeln ausstehen, wird das Spiel mit dem Aufruf der Sub ***EndOfBallComplete()*** fortgesetzt.

Sub EndOfBallComplete() - ab Zeile 768:

Diese Sub-Routine wird zum Abschließen des Balles aufgerufen. Als erstes wird ermittelt, ob 1 oder mehrere Spieler ein Spiel spielen. Wenn ja, dann ist die Nummer des nächsten Spielers um 1 höher als die des aktuellen Spielers:

```
If (PlayersPlayingGame > 1) Then
   NextPlayer = CurrentPlayer + 1
```

Wenn allerdings die Nummer des nächsten Spielers höher ist als die Anzahl der spielenden Spieler, ist der erste Spieler wieder an der Reihe. Dies wird mit dem folgenden Code realisiert:

```
If NextPlayer > PlayersPlayingGame Then
   NextPlayer = 1
End If
```

Wird der Spieltisch nur von 1 Person gespielt, kommt der ***Else***-Teil der Abrage nach der Anzahl der Spieler zur Ausführung und die Nummer des nächsten Spielers ist die gleiche wie die des aktuellen Spielers:

```
NextPlayer = CurrentPlayer
```

In Zeile 788 verzweigt sich der Code wieder, je nach dem, ob weitere Kugeln für weitere Spieler ausstehen oder nicht.

```
If ((BallsRemaining(CurrentPlayer) <= 0) And
BallsRemaining(NextPlayer) <= 0)) Then
```

Stehen keine weiteren Kugeln mehr aus, weder für den ***CurrentPlayer***, noch für den ***NextPlayer***, wird in Zeile 793 die Match-Zahl zwischen 0 und 10 zufällig generiert und mit 10 multipliziert, damit diese für die letzten beiden Stellen der Punkte angewendet werden kann:

```
MatchZahl = (Int(10 * Rnd)) * 10
```

Anschließend wird diese in der Variablen nvR2 gespeichert, damit diese beim erneuten laden und starten des Speiltisches als letzte Match-Zahl angezeigt werden kann:

```
nvR2 = MatchZahl
```

Da die Match-Zahl '0' auch nach der Multiplikation mit 10 immer noch 0 ist, die Match-Zahl aber im Display immer 2-stellig dargestellt wird, wird in diesem Fall in Zeile 796 statt der Match-Zahl '0' der Text '00' im Display angezeigt.

```
If MatchZahl = 0 Then
   DisplayBall.Text = "00"
```

Ist Die Match-Zahl jedoch > 0, dann wird diese angezeigt wie sie ist.

```
Else
   DisplayBall.SetValue(MatchZahl)
End If
```

Ab Zeile 806 wird eine Schleife durchlaufen, die für alle spielenden Spieler prüft, ob die letzten beiden Stellen des Punkte-Displays mit der produzierten Match-Zahl übereinstimmen. Wenn ja

```
If MatchZahl = nvScore(x) Mod 100 Then Replay = Replay + 1
```

dann wird die Variable Replay, die die Anzahl der zu vergebenden Freispiele enthält, um 1 erhöht. Im Folgenden wird in Zeile 810 geprüft, ob es einen neuen Highscore gibt. In diesem Fall werden den Replays 3 weitere Freispiele hinzugefügt.

```
If nvR1 > HighScore Then
   Replays = Replays + 3
```

Wenn die Anzahl der Freispiele (Replays) größer als 0 ist, wird abschließend der Timer, der die Freispiele letztlich vergibt, gestartet und die Sub ***EndOfGame()*** für das Ende Spieles aufgerufen.
Stehen noch weitere Kugeln zum Spielen aus - für den aktuellen oder nächsten Spieler - dann wird (für den Fall, dass mehr als 1 Spieler spielt) nach dem ***Else***-Teil (Zeile 817) der nächste Spieler zum aktuellen Spieler und die Sub zum den Reset für einen neuen Ball gestartet und abschließend der neue Ball generiert.

Sub KnockerTimer_Expired() - ab Zeile 828:

In dieser Sub wird zunächs dafür gesorgt, dass der Code des Timers wiederholt und erst dann gestopt wird, wenn alle Freispiele vergeben wurden. Ist dies der Fall, wird die Routine durch `ExitSub` verlassen.

```
KnockerTimer.Enabled = True
If Replays = 0 Then
   KnockerTimer.Enabled = False
   Exit Sub
End If
```

Beim Durchlauf des Timers wird dann geprüft, ob die maximal mögliche Anzahl gespeicherter Credits bereits erreicht ist, oder nicht. Falls dies nicht der Fall ist

```
If nvCredits < ConstMaxCredits Then
   nvCredits = nvCredits + 1
   PlaySound "Knocker"
End If
```

werden die Credits um 1 erhöht und dazu der entsprechende Sound gespielt. Das reduzieren der Replays um 1 erfolgt außerhalb der Abfrage nach der Anzahl der möglichen Credits, da die Replays ja in jedem Fall am Ende wieder auf '0' stehen müssen.
Abschließend wird ab Zeile 842 noch das Credit-Display aktualisiert. Ist die Anzahl der Credis kleiner als 10 (also 1-stellig), dann wird der Wert durch eine vorangestellte '0' 2-stellig dargestellt, wie dies bei den Credit-Displays üblich ist.

```
If nvCredits  < 10 Then
   DisplayCredit.Text = "0" & nvCredits
```

Sub EndOfGame() - ab Zeile 851:

Hier wird dem Spieltisch durch ***EndGame()*** zunächst signalisiert, dass das Spiel zu Ende ist. Desweiteren wird der Sound für das Spielende gespielt. Abschließend wird dafür gesorgt, dass die beiden Flipper-Finger in die 0-Stellung zurückfallen und über ***SetAllLightsForAttractMode()*** der Attract-Modus gestartet.

Sub SetAllLightsForAttractMode() - ab Zeile 867:

In dieser Sub-Routine werden die Lichter in den Attract-Modus geschaltet. Zuerst werden in einer Schleife allen runden Lichtern (ausgenommen die Bonuslichter) ausgeschaltet und anschließend in den Blink-Status versetzt.

```
For i = 1 to 18
   Bulb(i).State = BulbOff
```

Der zeitliche Abstand für das Blinken ist bei den Einstellungen der Lichter über den Blink Interval definiert. Das Blinkmuster wird über die Blink-Pattern definiert. In meinem Beispiel-Flipper sieht das Muster so aus: AN - AUS - AN - AN - AUS- AUS - AUS - AN - AUS. Dem entsprechend stellt sich das Blink-Pattern folgendermaßen dar:

```
    Bulb(i).BlinkPattern = "101100010"
    Bulb(i).State = BulbBlink
```

Dieses Muster wird wiederholt, so lange der BulbBlink-Status besteht. Bei den Bonuslichtern kommt im Attract-Modus ein Lauflicht zur Anwendung. Dieses wird realisiert, indem man jedem der 10 Bonuslichter ein um 1 Stelle versetztes Blinkmuster zuweist. Beachten Sie, dass die einzelnen Zeilen das jeweils nächst höhere Licht in der Leiter repräsentiert:

```
Bb_Bonus1.BlinkPattern = "01111111111000000000000000001"
Bb_Bonus2.BlinkPattern = "00111111111100000000000000010"
Bb_Bonus3.BlinkPattern = "00011111111110000000000000100"
Bb_Bonus4.BlinkPattern = "00001111111111000000000001000"
```
usw. ...

Das Ergebnis davon sieht so aus, dass zunächst von unten nach oben ein Licht nach dem anderen eingeschaltet und, wenn alle 10 Lichter leuchten, dann von unten nach oben ein Licht nach dem anderen wieder ausgeschaltet wird. Anschließend läuft ein einziges angeschaltetes Licht von oben nach unten (sieht richtig schön aus). In Zeile 892 werden dann noch die birnchen für "GAME OVER" eingeschaltet. Zeile 893 setzt dann das ***FlagHighScore*** auf ***TRUE***. Dieses Flag wird für das im Attract-Modus übliche hin und her schalten zwischen Punktezahl und Highscore benötigt. Zuletzt wird dann der Timer für das hin und her schalten gestartet.

Sub HighScoreTimer_Expired() - ab Zeile 900:

Der Code in diesem Timer wird wiederholt bis er durch den Start zu einem neuen Spiel gestopt wird. Nachdem der Timer abgelaufen ist, wird geprüft, ob ***FlagHighScore TRUE*** oder ***FALSE*** ist. Ist der wert **TRUE**, dann wird der ***Else***-Teil der Abrage abgearbeitet und somit in allen 4 Displays der HighScore gezeigt und abschließend das Flag auf ***FALSE*** gesetzt, damit beim nächsten Durchlauf die zuletzt erspielten Punkte angezeigt werden.

```
Else
  Bb_HighScore1.State = BulbOn: Bb_Highscore2.State = BulbOn
  Display1.SetValue(nvR1)
  Display2.SetValue(nvR1)
  Display3.SetValue(nvR1)
  Display4.SetValue(nvR1)
  HudDisplay.SetValue(nvR1)
  FlagHighscore = False
```

Ist der Wert von ***FlagHighScore*** jedoch ***FALSE*** dann zeigen die Displays die jeweils zuletzt erspielten Punkte. Sind noch keine Punkte erspielt, erscheint '00' in den Displays. Abschließend wird ***FlagHighScore*** auf ***TRUE*** gesetzt, damit im folgenden Ablauf wieder der Highscore gezeigt wird:

```
If FlagHighscore = False Then
   Bb_HighScore1.State = BulbOff
   Bb_Highscore2.State = BulbOff
   If nvScore1 = 0 Then
      Display1.Text = "00"
      HudDisplay.Text = "00"
   Else
      Display1.SetValue(nvScore1)
      HudDisplay.SetValue(nvScore(CurrentPlayer))
   End If
   If nvScore2 = 0 Then
      Display2.Text = "00"
      ...
```

Während der Code aus den gerade angesprochenen Sub-Routinen den Spielablauf steuern, wenden wir uns im Folgenden den Routinen zu, die aufgerufen werden, wenn die Kugel ein Ziel trifft (Targets, Triggers, Bumper, usw.) und Punkte vergeben werden. Bei der Vergabe und Wertung der Punkte unterscheiden sich meine Spieltische wiederum von anderen mir bekannten Spieltischen, bei denen die Punkte in der Reihenfolge vergeben, wie sie erspielt wurden. Angenommen die Kugel trifft einen Target, bei dem 5000 Punkte vergeben werden und während die gezählt werden dann einen der 500 vergibt, dann werden zuerst die 5 1000er Punkte gezählt und dann die 5 100er Punkte.
Bei realen Flippern - zumindest bei meinem KISS der Firma Bally aus dem Jahr 1978 - werden die Punkte jedoch nach einer Hirarchie vergeben. Bei diesem werden vorrangig die 10er Punkte gezählt, dann die 100er Punkte und zuletzt die 1000er Punkte. Entsprechend dem Beispiel zählt er zuerst die 1000er Punkte der 5000, aber nachdem der 500er Target getroffen wurde, fährt der Flipper mit dem Zählen der 100er Punkte fort, und erst nachdem diese komplett gezählt sind, werden die restlichen 1000er Punkte gezählt. Dabei wird auch ein fester Rhytmus beim zählen eingehalten. So werden beispielsweise Spinner-Punkte nicht in der Drehgeschwindigkeit des Spinners gezählt, sondern in einem Zwischenspeicher abgelegt, wenn die Drehbewegung des Spinners schneller ist als der vorgegebene Zähl-Rhytmus.
Aus diesem Grund werden bei allen meinen Spieltischen die Punkte entsprechend der Hirarchie und nicht in der Reihenfolge des Erspielens gezählt, wobei die Hirarchie auch umgekehrt sein kann, also zuerst 1000er, dann 100er und dann 10er Punkte.

Sub LeftTrigger_Hit() - ab Zeile 949:

Diese Routine wird aufgerufen, wenn die Kugel über den Trigger von einer der beiden oberen Durchlaufbahnen rollt. Zunächst wird festgelegt, dass die Sub verlassen werden soll, falls sich der Flipper im Tilt-Status befindet, was übrigens bei allen folgenden Routinen so festgelegt ist.
Danach wird das Lane-Licht ausgeschaltet und das korrespondierende Trigger-Licht im unteren Spielfeld eingeschaltet.

```
Bb_LeftUpperLane.State = BulbOff
Bb_TriggerButton1.State = BulbOn
```

Ab Zeile 955 erfolgt nun die Punktevergabe, bei der unterschieden wird, ob das Flag namens ***FlagWait*** den Wert ***TRUE*** oder ***FALSE*** enthält. Werden gerade Punkte auf dem Display gezählt, enhält das Flag den Wert ***TRUE***, während es den Wert ***FALSE*** enthält, wenn sich die Punkte-Zählung im Stillstand befindet, also alle zu vergenen Punkte gezählt wurden. Ist ***FlagWait*** also ***FALSE***, dann können die Punkte sofort gezählt werden und entsprechend der oben angesprochenen Hirarchie der Variablen ***ScoreCurrent1000*** zugeordnet. Dabei wird auch gleich das Flag auf ***TRUE*** gesetzt und der ***ScoreTimer*** zum Zählen mit ***ScoreTimer_Expired()*** ausgeführt, indem diese Sub sofort aufgerufen und nicht der Timer gestartet wird, da das Zählen natürlich ohne die Wartezeit des Timer-Intervals starten muss (Wenn Sie den ScoreTimer mit .Enabled = True starten, beginnt dieser ja erst mit dem Zählen, wenn die mit TimerInterval definierte Zeit abgelaufen ist).

```
If FlagWait = False Then
   FlagWait = True
   ScoreCurrent1000 = 3000
   ScoreTimer_Expired()
```

Steht das ***FlagWait*** jedoch bereits auf ***TRUE***, da gerade Punkte gezählt werden, werden die zu vergebenden Punkte in der Variablen ***ScoreNext1000*** zwischengespeichert.

```
Else
   ScoreNext1000 = ScoreNext1000 + 3000
End If
```

Im Folgenden wird die ***CheckExtraBall()*** Sub aufgerufen, die prüft ob eventuell das ShootAgainLicht ausgeschaltet werden muss, und zuletzt wird der Bonus über den Aufruf der Sub ***IncrementBonus()*** vergeben.
Diese Vorgehensweise wird auch auf die recht Lane angewendet, nur dass sich dort das Handling auf die jeweils rechte Variante bezieht.

Sub DropTargetLeft_Hit() - ab Zeile 992:

Diese Routine wird aufgerufen, wenn die Kugel einen Target der oberen linken Droptarget-Bank trifft. Das Handling mit den Punkten entspricht dem, wie wir dies oben besprochen haben, mit dem einzigen Unterschied, dass hier ***ScoreCurrent100*** bzw. ***ScoreNext100*** verwendet werden, da hier 100er Punkte (500) vergeben werden.
Da die Droptarget-Bank resettet werden muss, wenn alle Targets getroffen wurden, wird dies ab Zeile 1006 geprüft. Wenn dies der Fall ist, wird dem Flag ***LastSwitichHit*** der Name des zuletzt getroffenen Objektes, in unserem Fall ist dies ***DropTargetLeft***, zugewiesen.

```
If DropTargetLeft.Dropped = True Then
   Set LastSwitchHit = DropTargetLeft
```

Dieses Flag wird später benötigt, wenn die Dropttarget-Bank resettet wird. Nachdem das Flag gesetzt ist, wird dann auch der ***DropTargetResetTimer*** gestartet.

```
   DropTargetResetTimer.Enabled = True
End If
```

Die rechte obere Dropttarget-Bank (Zeile 1016) und ebenfalls die 4er Droptarget-Bank links (Zeile1039) erfahren die gleiche Behandlung, wie gerade abgehandelt, jedoch werden bei der 4er Droptarget-Bank noch über den Befehl

```
CountDTReset = CountDTReset + 1: If CountDTReset > 3 Then
CountDTReset = 3
```

die Anzahl der Resets gezählt.

Sub DropTargetResetTimer_Expired() - ab Zeile 1062:

Wenn jeweils alle Targets einer Droptarget-Bank getroffen wurde, wird diese Routine aufgerufen, um die Bank zurückzusetzen. Zunächst wird der Timer gestopt, damit der Code nach dem Durchlaufen nicht wiederholt wird

(`DropTargetResetTimer.Enabled = FALSE`).

Danach wird das Flag LastSwitchHit abgefragt, um die richtige Droptarget-Bank zurückzustellen. Mit

```
If LastSwitchHit.Name = "DropTargetLeft" Then
```

reagiert der Code des Timers auf das Dropped-Ereignis der linken oberen Droptarget-Bank. Dabei werden über die folgenden Zeilen zunächst das Licht der Droptarget-Bank ausgeschaltet und das Licht am Spinner eingeschaltet.

```
   Bb_LiteSpinner.State = BulbOff
   Bb_Spinner.State = BulbOn
```

Anschließend wird die Bank über die ***.SolenoidPuls*** Methode zurückgesetzt und der entsprechende Sound gespielt. Zuletzt wird das Flag ***LastSwitchHit*** wieder auf den ***DummyTrigger*** gesetzt.

```
   DropTargetLeft.SolenoidPulse
   PlaySound "DropTargetReset"
   Set LastSwitchHit = DummyTrigger
```

Die nächsten Fälle der Abfrage beziehen sich auf die rechte bzw. 4er Droptarget-Bank.

```
ElseIf LastSwitchHit.Name = "DropTargetRight" Then
   ...
ElseIf LastSwitchHit.Name = "DropTarget4Bank" Then
   ...
```

Die grundsätzliche Vorgehensweise ist hier die gleiche wie bei der linken Droptarget-Bank, einzig die 4er Droptarget-Bank links unterscheidet sich von den anderen, weil hier noch eine Abfrage nach der Anzahl der bereits vollzogenen Resets erfolgen muss, da entsprechend dieser Anzahl die Lichter für Doppelbonus, Extra Ball und Special geschaltet werden müssen. Dies erfolgt über

```
Select Case CountDTReset
```

da ***CountDTReset*** die Anzahl der Resets enthält. So wird, wenn es sich um den ersten Reset handelt, das Doppelbonus-Licht und der Bonus-Multiplier behandelt,

```
Case 1
   Bb_Lite2x.State = BulbOff
   Bb_2x.State = BulbOn
   BonusMultiplier(CurrentPlayer) = 2
   Bb_LiteExtraBall.State = BulbOn
```

während bei dem zweiten Reset Extra Ball und ShootAgain-Licht behandelt werden.

```
Case 2
   Bb_LiteExtraBall.State = BulbOff
   ShootAgainLight.State = BulbOn
   ExtraBallsAwards(CurrentPlayer) = 1
   Bb_LiteSpecial.State = BulbOn
```

Beim dritten Reset werden die Special-Lichter angesprochen und das ***FlagSpecialEnabled*** auf ***TRUE*** gesetzt, damit das linke und rechte Special-Licht der Outlanes über die Slingshots alternieren können.

```
Case 3 ' beim 3. Reset
   Bb_LiteSpecial.State = BulbOff
   Bb_SpecialLeft.State = BulbOn
   FlagSpecialEnabled = True
```

Sub Spinner1_Hit() - ab Zeile 1104:

Wenn der Spinner getroffen wird, wird für die Vergabe der Punkte unterschieden, ob das Licht am Spinner leutet (1000 Punkte) oder nicht (100 Punkte). Die Handhabung entspricht der bereits erklärten Vorgehensweise, weshalb ich die hier nicht noch einmal durchgehe.

Sub Kicker1_Hit() – ab Zeile 1131:

Diese Routine wird im TILT-Status nicht sofort verlassen, wie dies bei anderen Objekten der Fall ist, da die Kugel auch in diesem Fall ja ausgestoßen werden muss. Mit der Abfrage und den Anweisungen

```
If fpTilted = True Then
   KickerTimer_Expired()
   Exit Sub
End If
```

wird, sofern der Flipper getilt ist, zuerst der Timer für den Ausstoß der Kugel gestartet und dann die Sub verlassen.
Als nächstes wird auch hier das Flag ***LastSwitchHit*** belegt, da dieses später nach dem zählen der Kicker-Punkte für das weitere Fortfahren benötigt wird, und es wird ein Sound gespielt, der den Kontakt der Kugel mit dem Kicker wiedergibt (Zeile 1137 und 1138).
Im Folgenden wird auch hier unterschieden, ob das Licht am Kicker ein- oder ausgeschaltet ist, da unterschiedliche Punkte vergeben werden. Die Punkte werden hier jedoch einer einheitlichen Variablen ScoreKicker zugeordnet, da Kicker-Punkte beim zählen etwas anders behandelt werden als die anderen 100er bzw. 1000er Punkte, wie wir später sehen werden.

Sub KickerTimer_Expired() - ab Zeile 1156:

Dieser ***KickerTimer*** wird gestartet, wenn die Kicker-Punkte abgezählt wurden. Der Timer wird zunächst gestopt, dann wird die Kugel über die ***.SolenoidPuls*** Methode ausgestoßen und dazu der passende Sound gespielt. Abschließend wird der ***WaitTimer*** ausgeführt, auf den wir ebenfalls später noch zu sprechen kommen.

Sub Bumper1_Hit() - ab Zeile 1166:

Dieser Code zum Werten der Bumper-Punkte entspricht eins zu eins dem, wie er auch schon bei den anderen Objekten angesprochen wurde, weshalb hier nichts zusätzlich zu erklären ist.Da Bumper2 und Bumper3 nichts anderes machen als Bumper1, wird in diesen Sub-Routinen mit der Zeile

```
Bumper1_Hit()
```

auf Bumper1 verwiesen.

Sub LeftUpperSlingshot_Hit() - ab Zeile 1193:

In dieser Sub und auch in der ***RightUpperSlingshot_Hit()*** wird mit der Zeile

```
LeftSlingshotRubber_Hit()
```

auf den linken unteren Slingshot verwiesen, da der Code für die beiden oberen Slingshots der gleiche ist, wie der, mit dem der untere linke Slingshot behandelt wird.

Sub LeftBonusTrigger_Hit() - ab Zeile 1207:

Diese Sub wird aufgerufen, wenn die Kugel über den linken Trigger-Button im unteren Spielfeld-Bereich rollt. Auch der Code bei diesen beiden Triggers ***LeftBonusTrigger*** und ***RightBonusTrigger*** dürfte Ihnen keine großen Rätsel aufgeben. Hier wird wie beim Spinner oder Kicker ebenfalls unterschieden, ob das Licht an oder aus ist, da dem entsprechend unterschiedliche Punkte vergeben werden. Die Vergabe der Punkte selbst entspricht dann wiederum der Vorgehensweise wie bei den anderen Objekten.

Sub LeftSlingshotRubber_Hit() - ab Zeile 1264:

Der linke untere Slingshot wurde getroffen. Die Code-Zeilen für die Punkte-Vergabe brauchen auch hier nicht weiter erklärt zu werden. Einzig die Abfrage des ***FlagSpecialEnabled*** ab Zeile 1279 bedarf weiterer Erklärung.
Dieses Flag wird auf *TRUE* gesetzt, nachdem die 4er Droptarget-Bank zum dritten mal resettet wurde und das Licht der linken Outlane eingeschaltet wurde. Anderfalls hat es den Wert ***FALSE***. Wenn dieses Flag den Wert TRUE besitzt, werden die beiden Special-Lichter der Outlanes wechselseitig ein- bzw. ausgeschaltet, wie im Script dokumentiert.

Sub RightSlingshotRubber_Hit() - ab Zeile 1293:

Hier wird der rechte untere Slingshot getroffen. Da in dieser Sub nichts anderes geschieht wie beim linken Gegenstück, wird hier über die Zeile

```
LeftSlingshotRubber_Hit()
```

verwiesen.

Sub LeftInLaneTrigger_Hit() - ab Zeile 1300:

Diese Sub wird aufgerufen, wenn die Kugel über den Trigger der linken Inlane rollt. Hier werden lediglich die zugehörigen Punkte so wie Bonuspunkte vergeben und die Sub ***CheckExtraBall()*** aufgerufen, um ein eventuell leuchtendes ShootAgain-Licht auszuschalten.

Sub RightInLaneTrigger_Hit() - ab Zeile 1318:

Da hier nichts anderes geschieht als beim linken Gegenstück wird hier die Routine der linken Inlane aufgerufen.

Sub LeftOutLaneTrigger_Hit() - ab Zeile 1325:

Diese Sub wird aufgerufen, wenn die Kugel über den Trigger der linken Outlane rollt. Auch hier verzichte ich auf die nochmaliger Erklärung der Punkte-Vergabe. Nachdem diese den entsprechenden Variablen zugeordnet wurden, der Bonus gewertet (***IncrementBonus()***) und auf ein eventuell leuchtendes ShootAgainLicht über ***CheckExtraBall()*** reagiert wurde, wird ein Freispiel vergeben, wenn das zugehörige Licht der Lane leuchtet:

```
If Bb_SpecialLeft.State = BulbOn Then
```

im relevanten Fall wird das Licht dann ausgeschaltet und das ***FlagSpecialEnabled*** auf ***FALSE*** gesetzt, da nur 1 Special-Freispiel pro Ball vergeben wird. Das Freispiel wird jedoch nur vergeben, wenn die maximal mögliche Anzahl an gespeicherten Credits noch nicht erreicht ist:

```
If nvCredits < ConstMaxCredits Then
```

Ist dies der Fall, wird die globale Variable nvCredits um 1 erhöht und der entsprechende Sound gespielt.

```
   nvCredits = nvCredits + 1
   PlaySound "Knocker"
End If
```

Danach wird das Display 'Credits' aktualisiert. Falls die Anzahl der Credits kleiner als 10 ist

```
If nvCredits  < 10 Then
```

wird der einstellige Wert durch eine vorangestellte '0' zweistellig dargestellt, wie dies hier üblich ist.

```
   DisplayCredit.Text = "0" & nvCredits
```

Andernfalls erfolgt die Anzeige im Display mit dem Wert von nvCredits.

```
Else
   DisplayCredit.SetValue(nvCredits)
End If
```

Sub RightOutLaneTrigger_Hit() - ab Zeile 1360:

Dieses Hit-Ereignis entspricht dem der linken Outlane, so dass hier nicht weiter darauf eingeganen werden muss.

Sub ScoreTimer_Expired() ab Zeile 1399:

Dieser Timer addiert die erspielten Punkte über die ***AddScore()***-Routine zum Punktestand des aktuellen Spielers. Über die Abfrage

```
If (fpTilted = True) Then
```

wird zunächst geprüft, ob sich der Flipper im TILT-Status befindet. Ist dies der Fall, werden alle ausstehenden und zwischengespeicherten Punkte gelöscht und die Sub anschließend verlassen, da die Punkte dann nicht gewertet werden.

```
   ScoreCurrent1000 = 0: ScoreNext1000 = 0
   ScoreCurrent100 = 0 : ScoreNext100 = 0
   ScoreCurrent10 = 0: ScoreNext10 = 0
   ScoreKicker = 0
   Exit Sub
End If
```

Die nächste Abfrage (ab Zeile 1409) prüft, ob alle ausstehenden und zwischengespeicherten Punkte gezählt wurden.

```
If ScoreCurrent1000 = 0 And ScoreCurrent100 = 0 And
ScoreCurrent10 = 0 And ScoreKicker = 0 Then
```

Ist dies der Fall, wird der ***ScoreTimer*** gestopt und der **WaitTimer** aufgerufen, der prüft, ob zwischenzeitlich eventuell weitere Punkte erspielt wurden. Dann wird die Sub verlassen.

```
   ScoreTimer.Enabled = False
   WaitTimer_Expired()
   Exit Sub
End If
```

Im nächsten Schritt werden angesammelte zwischengespeicherte Punkte auf die Variablen für die sofort zu vergebenden Punkte übertragen.

```
ScoreCurrent1000 = ScoreCurrent1000 + ScoreNext1000
ScoreNext1000 = 0
ScoreCurrent100 = ScoreCurrent100 + ScoreNext100
ScoreNext100 = 0
ScoreCurrent10 = ScoreCurrent10 + ScoreNext10
ScoreNext10 = 0
```

Erinnern Sie sich daran, dass Punkte in der Variablen ***ScoreNext...*** zwischengespeichert werden, wenn ***FlagWait*** = ***TRUE*** ist und gerade Punkte gezählt werden.
In den folgenden Blocks wird geprüft ob 1000er bzw. 100er, 10er oder KickerPunkte zum zählen anstehen. Da das Zählen nach einer bestimmten Hirarchie vonstatten geht, nämlich zuerst KickerPunkte, dann 10er Punkte, dann 100er Punkte und zuletzt 1000er Punkte, werden 1000er Punkte erst dann gezählt, wenn alle anderen ***ScoreCurrent...*** Variablen bzw. der ***ScoreKicker*** '0' enthalten.

```
If ScoreCurrent1000 > 0 And ScoreCurrent100 = 0 And
ScoreCurrent10 = 0 And ScoreKicker = 0 Then
   AddScore(1000)
   ScoreCurrent1000 = ScoreCurrent1000 – 1000
```

In diesem Fall werden 1000 Punkte als Parameter der Routine AddScore() übergeben und diese 1000 Punkte von den zu zählenden ScoreCurrent1000 abgezogen. Gleichfalls wird der zu diesem Wert gerhörende Sound gespielt.

```
   PlayMusic 2, "1000",,0.7
End If
```

Die Parameter von PlayMusic betreffen zuerst den Kanal, dann den Namen des Sounds, ob der Sound wiederholt werden soll (wird hier freigelassen) und zuletzt die Lautstärke, die im aktuellen Fall 0,7 der vollen Lautstärke bedeutet.
In der gleichen Vorgehensweise werden die 100er und 10er Punkte behandelt. Bei der verarbeitung der Kicker-Punkte wird abgefragt, wie hoch diese sind. betragen diese mehr als 500 Punkte, werden 1000er Punkte nach dem gleichen Schema addiert, wie es auch schon bei ScoreCurrent1000 der Fall ist. Ansonsten werden 100er Punkte vergeben.
Wenn der ScoreKicker '0' ist und das zuletzt getroffene Ziel der Kicker war

```
If ScoreKicker = 0 And LastSwitchHit.Name = "Kicker1" Then
```

dann wird das Flag wieder auf DummyTrigger gesetzt und anschließend der KickerTimer für den Auswurf der Kugel gestartet.

```
   Set LastSwitchHit = DummyTrigger
   KickerTimer.Enabled = True
End If
```

Als letzte Anweisung im ScoreTimer wird dieser erneut gestartet, damit sämtliche Punkte abgefragt und gezählt werden können.

Sub AddScore(points) - ab Zeile 1468:

Die Anweisungen in dieser Sub-Routine finden nur statt, wenn sich der Flipper nicht im TILT-Status befindet. Über die Anweisung

```
nvScore(CurrentPlayer) = nvScore(CurrentPlayer) + points
```

werden die Punkte, die der Sub als Parameter ***Points*** übergeben wurden, zum Punktestand des aktuellen Spielers addiert. Eine Abfrage stellt sicher, dass dies auch auf dem richtigen Display geschieht.

```
Select Case (CurrentPlayer)
   Case 1: Display1.AddValue(points)
      HudDisplay.AddValue(points)
   Case 2: Display2.AddValue(points)
      HudDisplay.AddValue(points)
   ...
End Select
```

Im folgenden Block der Sub werden die Punkte-Freispiele gegeben. Das erste Freispiel gibt es bei 300.000 Punkten, das zweite bei 480.000 Punkten. Wenn die Punkte also gleich oder größer als diese Summe ist, aber kleiner als 480.000 Punkte und das erste Freispiel noch nicht gegeben wurde (***FlagFreispiel1 = FALSE*** ist), dann wird das erste Freispiel gegeben.

```
If nvScore(CurrentPlayer) >= 300000 AND nvScore(CurrentPlayer)
< 480000 And FlagFreispiel1(CurrentPlayer) = False Then
```

Aber auch hier wird das Freispiel nur gegeben, wenn die maximale Anzahl der möglichen Credits noch nicht erreicht wurde.

```
If nvCredits < constMaxCredits Then
```

Das Vergeben des Freispieles wurde weiter oben schon erklärt, weshalb ich hier nicht weiter darauf eingehe. Als letztes wird das ***FlagFreispiel1*** auf ***TRUE*** gesetzt, damit das Freispiel für die 300000 Punkte nicht mehrmals vergeben werden kann. Das Vergeben des zweiten Freispieles bei 480000 Punkten wird analog dem ersten behandelt, weshalb ich dies hier ebenfalls nicht noch einmal erklären muss.

Sub WaitTimer_Expired() - ab Zeile 1524:

Der WaitTimer stellt letztlich sicher, dass die Punkte in einem vorgegebenen Rhytmus gezählt werden, also das Zählen der einzelnen Beträge nicht abgeschnitten oder beschleunigt wird. Desweiteren werden hier noch eventuell ausstehende Punkte aus den ***ScoreNext...*** Variablen an die ***ScoreCurrent...*** Variablen übertragen und die ***ScoreNext...*** jeweils auf '0' gesetzt. In diesem Fall wird danach die Sub ohne das Zurücksetzen des FlagWait verlassen.
Falls keine wertbaren Punkte mehr existent sind, wird das FlagWait auf FALSE gesetzt, so dass die nächsten Punkte sofort gezählt werden können.

Sub IncrementBonus() - ab Zeile 1544:

Diese Sub wird von den Routinen aufgerufen, in denen beim ***_Hit()*** Ereignis Bonuspunkte vergeben werden. Das angewendete Prinzip dabei ist vergleichbar mit dem, nach dem wir den Bonus bei Ball-Ende heruntergezählt haben, nur in umgekehrter (aufsteigender) Reihenfolge.

```
If Bonus < 10 Then
```

Ist der angesammelte Bonus kleiner als 10, kommt der erste Block zur Anwendung. der Bonus wird um 1 erhöht und das 'neue' Licht durch ansprechen der Feldvariablen ***Bb_Bonus()*** mit der Nummer von ***Bonus*** eingeschaltet.

```
Bonus = Bonus + 1
Bb_Bonus(Bonus).State = BulbOn
```

Nur wenn der Bonus noch größer ist als 1, wird das 'alte' Licht ausgeschaltet, indem wir die Nummer (Bonus -1) verwenden

```
If Bonus > 1 Then Bb_Bonus(Bonus - 1).State = BulbOff
```

und danach die Sub verlassen. Nehmen wir an, der Bonus sei aktuell 6, dann leuchtet das 6000er Licht. Nach dem erhöhen des Bonus hat dieser den Wert 7. Mit ***Bb_Bonus(Bonus).State = BulbOn*** wird somit das 7000er Licht eingeschaltet. Nun muss jedoch das 6000er Licht noch ausgeschaltet werden, weshalb wir dies mit ***Bb_Bonus(Bonus - 1)*** realisieren (7 - 1 = 6).
Ist der Bonus größer als 10, aber kleiner als 19 (maximal möglicher Bonus), dann kommt der zweite Block zur Anwendung:

```
If Bonus > 10 And Bonus < 19 Then
```

In diesem Fall leuchtet das 10000er Licht bereits, weshalb wir dieses nicht beachten brauchen. Jedoch werden Bonuspunkte über 10000 mit einem zweiten Licht dargestellt, weshalb wir dieses hier behandeln müssen. Zunächst wird ebenfalls der Bonus um 1 erhöht. Das 'neue' Bonuslicht wird dann über

```
Bb_Bonus(Bonus -10).State = BulbOn
```

eingeschaltet und das 'alte' über

```
Bb_Bonus(Bonus - 11).State = BulbOff
```

ausgeschaltet. Zur besseren Verständlichkeit hier ein Beispiel: Angenommen der aktuelle ***Bonus*** hat den Wert ***14***, so dass das 10000er und das 4000er Licht leuchtet. Nach dem erhöhen des Bonus um 1 müssen nun 15000 Punkte dargestellt werden und deshalb als zweites Licht das 5000er Licht eigngeschaltet werden. Dies erreichen wir, in dem wir für ***Bonus*** den Wert (***Bonus - 10)*** heranziehen. 15 - 10 = 5. Somit wird mit ***Bb_Bonus(Bonus -10).State = BulbOn*** das 5000er Licht eingeschaltet. Nun muss das 'alte' Licht noch ausgeschaltet werden. Da dieses mit der nächst niedrigeren Nummer angesprochen werden muss, muss der Wert ***Bonus*** um 11 reduziert werden: 15 - 11 = 4. Somit wird mit ***Bb_Bonus(Bonus - 11).State = BulbOff*** das 4000er Licht ausgeschaltet.
Der dritte Block kommt zur Anwendung, wenn der Bonus = 10000 Punkte beträgt. In diesem Fall leuchtet das 10000er Licht bereits und es muss nach der erhöhung des

Bonus um 1 lediglich das zweite Licht noch eingeschaltet werden. Dies ist bei der Darstellung von 11000 Punkten logischer weise das 1000er Licht, was mit

```
Bb_Bonus(1).State = BulbOn
```

realisiert wird.

Sub CheckExtraBall() - ab Zeile 1566:

Dies ist die letzte Sub-Routine in unserem Script. Sie wird von jedem ***_Hit()*** Event eines Objektes aufgerufen, da sie die Aufgabe hat, ein eventuell blinkendes ShootAgain-Licht auszuschalten. Dies darf jedoch nur geschehen, wenn der Extra Ball als neuer Ball ausgeworfen und ins Spielfeld geschossen wird. Da die einfache Abfrage des Status hier nicht funktionieren würde, habe ich dies über ein Flag realisiert, welches nach dem Auswurf des Extra Balls in die Plungerlane auf True gesetzt wird. Dies signalisiert, dass es sich bei diesem um den Extra Ball handelt. In diesem Fall wird das Flag wieder zurückgesetzt

```
If FlagExtraBall = True Then
   FlagExtraBall = False
```

und anschließend der Extra Ball gelöscht, sowie das ShootAgain-Licht ausgeschaltet.

```
   ExtraBallsAwards(CurrentPlayer) = 0
   ShootAgainLight.State = BulbOff
End If
```

Das war's! Lassen Sie mich abschließend noch einige Tips zum Scripting generell geben: Vielleicht ist es Ihnen aufgefallen, dass wir grundsätzlich 3 verschiedene Typen von Sub-Routinen hier verwendet haben. Da sind solche, die beim Hit-Ereignis eines Objektes durchlaufen werden, andere beinhalten Code, der ausgeführt wird, wenn ein Timer abgelaufen ist und eine dritte Variante, auf die irgendwo in einer Sub verwiesen wird.
Diese letztere Variante dient dazu, die Code-Zeilen des Scriptings zu minimieren. In meinem Script wird z. B. bei jedem Hit-Ereignis geprüft, ob das ShootAgain-Licht auszuschalten ist. Statt die relevanten Zeilen in jeder dieser Subs mit aufzunehmen, habe ich diese in einer eigene Sub aufgenommen (***CheckExtraBall()***), auf die mit einer einzigen Zeile Code verwiesen wird. Gleiches gilt auch für die Sub ***IncrementBonus()***, deren Code aus einigen Subs aufgerufen wird. Ich hätte das gesamte Script noch weiter minimieren können, z. B. die Code-Zeilen für das Werten der Punkte, da sich diese ja auch recht oft wiederholen. Allerdings habe ich darauf verzichtet, um die Erklärungen zum Script nicht zu unübersichtlich werden zu lassen. Sie sehen aber, wenn Sie sich im Vorfeld über den Ablauf der verschiedenen Schritte reichlich Gedanken machen, können Sie sich viel Tip-Arbeit sparen. Ich hoffe, dass ich Ihnen die Vorgänge beim Scripting nun verständlich genug erklärt habe, so dass Sie die Entwicklung eines eigenen Spieltisches erfolgreich abschließen können und wünsche gutes Gelingen!