Chapter3　ミニゲームの作成

ここではミニゲームの作成を通して、GameMaker:Studio の基本的な操作をマスターすることを目的とします。GameMaker:Studio は直感的にゲームを作成できますが、それでも多少の使い方を知っておく必要があります。最初はいろいろな機能や項目があって圧倒されてしまうかもしれませんが、いきなりすべてを理解する必要はありません。まずは最低限の使い方をここで説明しますので、少しずつ慣れていきましょう。

Section3-1　使用する素材のダウンロード

今回作成するミニゲームで使用する素材は、「GitHub」にアップしています。以下のページへ進み、右下にある「Download ZIP」からダウンロードできます（図3-1）。

・https://github.com/syun77/GameMaker_Sample

図 3-1

今回使用するデータの構成は以下のようになっています。

```
GameMaker_Sample-master
  Exe: サンプルゲームの実行ファイル
    MiniGame.exe: ミニゲーム完成の実行ファイル
    MiniGame_Ext.exe: ミニゲームを改造した実行ファイル

  Projects: GameMaker:Studioのプロジェクトファイル
    MiniGame.gmx: ミニゲーム完成プロジェクト
    MiniGame_Ext.gmx: ミニゲーム改造プロジェクト

  Materials: 素材フォルダ
    minigame: ミニゲーム用素材
    minigame_ext: ミニゲーム改造用素材
```

「Exe/MiniGame.exe」を実行して、どんなものを作るのか確認しておくとよいかと思います。

Section3-2　GUI の説明

起動直後の画面は図3-2のようになっていると思います。

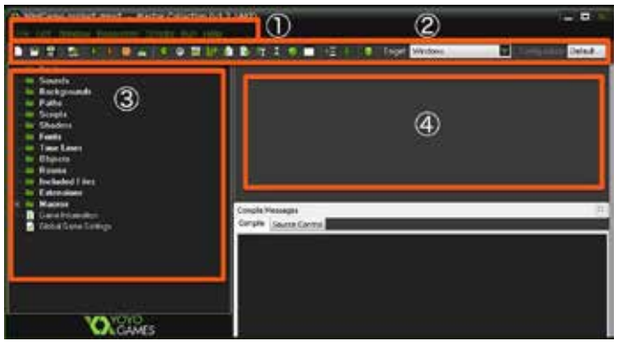

図 3-2

もし起動できていなければ、「Section2-2　インストール」の後半に書かれている起動方法・プロジェクトの作成方法を参考にしてプロジェクトを作成しておきましょう。

①メインメニュー
メインメニューは、すべての機能にアクセスできるメニューです。

②メインツールバー
メインメニューからよく使う機能に、直接アクセスできるアイコンです。

③リソースツリー
ここにゲームに必要なリソースを追加していきます。

④ワークスペース
ここでリソースの詳細な動作などを編集します。

他にもいろいろありますが、ひとまずはこの4つのカテゴリを覚えておけば大丈夫です。

Section3-3　ゲームを起動する

それではさっそくゲームを起動したいと思います。メインツールバーにある緑色の三角矢印 をクリックしてみてください。これがゲームの実行ボタンとなります。

すると、エラーダイアログが出て起動に失敗してしまいました。メッセージ内容は「ゲームを実行するには少なくとも1つの『ルーム』が必要です」というものです（図3-3）。

図 3-3

GameMaker:Studioは「ルーム」を1つの単位としてゲームを管理します。ルームというのは、複数のオブジェクトを配置し管理する空間のことです。

この説明はちょっと難しかったかもしれません。わかりやすく言うと、ルームというのはタイトル画面、オプション画面、メインゲーム画面、といった1つの「画面」の単位と考えることができます。

先ほどのエラーメッセージを翻訳すると「ゲーム画面（＝ルーム）がないから起動できない」という意味となります。つまりルームが必要、ということでさっそくルームを作成しましょう。

リソースツリーから「Rooms」を右クリックして「Create Room」選びます（図3-5）。すると、「room0」というルームが作られます。

図 3-4

これでゲームが起動可能となりました。メインツールバーにある緑色の三角矢印をクリック、もしくは［F5］キーを押してゲームを実行してみましょう。

何もない灰色の画面ですが、ゲームが起動できました（図3-5）。この画面が「ルーム」となります。起動が確認できたら、右上の×ボタンを押して灰色の画面を閉じます。

図 3-5

Section3-4　ルームの設定項目と保存

ルームの設定項目

ゲーム画面のサイズですが、デフォルトだと「1024x768」なのでちょっと大きいですね。ここではサイズを「640x480」に変更したいと思います。画面サイズを変更するには「Room Properties」画面の「setting」タブを選択します（図3-6）。

図 3-6

設定項目が表示されるので、以下のように変更します。

・Name: rm_main
・Width: 640
・Height: 480
・Speed: 60

サイズを640x480にするついでに他の項目も変更しました。「Name」というのはルームの名前です。この画面をメインゲーム画面とするので「rm_main」としています。「Speed」というのはゲームの実行速度です。これは1秒間に何回更新されるかを表す数値です。いわゆる「FPS」にあたるものです。一般的なアクションゲームでは「60」が基準となるのでここでは「60」に変更しました。

変更ができたら実行してみてください。実行すると画面が小さくなりましたね。確認できたら画面を閉じます。

そうしたら「Room Properties」画面の左上のチェックボタン を押して閉じます。すると、リソースツリーのルームの名前が「rm_main」に変わりました。GameMaker:Studioは、このようにチェックボタンを押して閉じたり、[OK]ボタンを押して閉じないとリソースに設定した内容が反映されないことあるので、不要なウィンドウはこまめに閉じたほうがよいです。

プロジェクトの保存

では、さっそくゲームキャラクターの配置を……、と先へ進む前に「保存」の方法を覚えておきましょう。

保存をするには、メインツールバーからフロッピーディスクのアイコン をクリックします。すると「Saving...」というダイアログが一瞬表示され、セーブが完了します。セーブをせずに何時間も作業を続け、強制終了でデータを失うとそれまでの作業が無駄になってしまいます。そのため、こまめにセーブするとよいです。[Ctrl] + [S] キーで保存することもできるので、定期的にこのショートカットキーを押すのもよいでしょう。

Section3-5　スプライトの作成

GameMaker:Studioではゲームキャラクターなどで使用する画像を表示するために、「スプライト」という仕組みが用意されています。これを使って画像を読み込んでみます。

読み込む画像は「Material/minigame」にある「tako.png」を使用します（図3-7）。では、スプライトを作成します。

図 3-7

リソースツリーにある「Sprites」を右クリックして「Create Sprite」を選択します（図3-8）。

図 3-8

「Sprite Propeties」というスプライトエディタが表示されます。まず左上の「Name」を選択して「sprite0」を「spr_tako」に変更します。たこ焼きの画像のスプライトなので、このような名前にしています。次に、その下にある［Load Sprite］ボタンを押します（図3-9）。

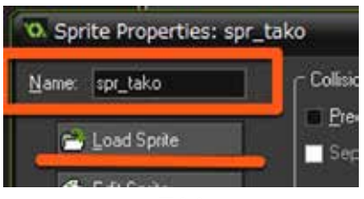

図 3-9

画像読み込みダイアログが表示されるので、ダウンロードした素材フォルダのなかの「Materials/minigame」フォルダから「tako.png」を選び、［開く］ボタンで画像を読み込みます（図3-10）。

図 3-10

画像を読み込むと右側に読み込んだ画像が表示されます。読み込まれたことを確認して、左下にある［OK］ボタンを押してスプライトエディタを閉じます（図3-11）。

図 3-11

Section3-6　オブジェクトの作成

スプライトを表示するために、オブジェクトを作成します。オブジェクトとは、ゲームを制御するためのインスタンスの元となるものです。ひとまずは「ゲームキャラクター＝オブジェクト」であると考えても問題ありません。

リソースツリーから「Objects」を右クリックして「Create Object」を選択します（図3-12）。すると、「Object Properties」が表示されます。ここでオブジェクトの設定を行います。

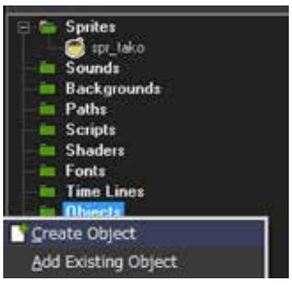

図 3-12

まずは名前を設定します。左上にある「Name」を「object0」から「obj_tako」を変更します（図3-13）。

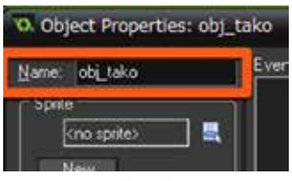

図 3-13

次に、オブジェクトとスプライトを関連付けします。Nameの下にある「Sprite」の中の「<no sprite>」をクリックするとスプライト選択のポップアップが表示されるので、「spr_tako」を選択します（図3-14）。

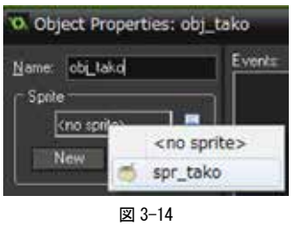

図 3-14

「spr_tako」スプライトをオブジェクトに設定できました。そうしたら、左下の［OK］ボタンを押してスプライト設定を閉じます（図3-15）。

図 3-15

では、［F5］キーを押すなどしてゲームを実行してみましょう。しかし灰色の画面のままですね。これはゲームオブジェクトを定義しただけではルームに配置されないためです。

Section3-7　オブジェクトをルームに配置する

リソースツリーから「Rooms」の中にある「rm_main」をダブルクリックして開きます。

すると、左上の「objects」タブの下にたこ焼きオブジェクトが表示されているはずです（図3-16）。もし表示されていなければ、別のタブが選択されている可能性があります。その場合は「objects」タブを選択すると表示されます。

図 3-16

たこ焼きをルームに配置します。右のグリッドの好きなところを左クリックして配置します。

ひとまず4つほど好きな位置に配置しましょう（図3-17）。配置しすぎた場合は、配置したオブジェクトを右クリックするとポップアップが表示されるので、そこから「Delete」を選ぶと削除できます。

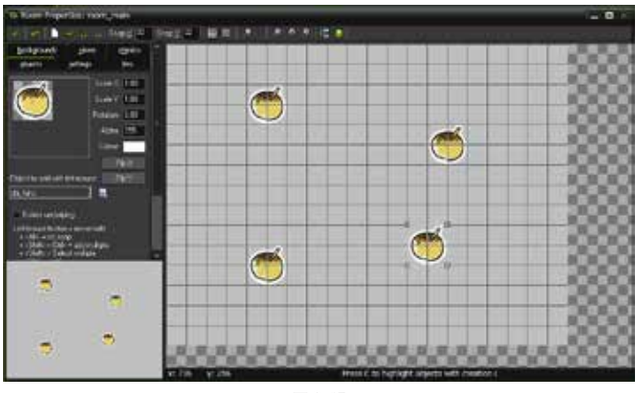

図 3-17

配置ができたら実行してみてください。たこ焼きが配置できているのが確認できます。なお、このようにルームにオブジェクトを配置することを「インスタンス化」と呼びます。

これでようやくキャラクターが画面に表示できたので、おさらいです。

まず、用意した画像データを読み込むにはスプライトを作成し、そこからロードします。スプライトは画像データ、当たり判定の管理を行います。次にオブジェクトを用意し、スプライトを登録します。オブジェクトは登録したスプライトを表示します。また、オブジェクトはキャラクターの移動処理などの制御を行います。そして、オブジェクトをルームに配置します。これによりインスタンス化が行われ、ルームにキャラクターが表示されることとなります（図3-18）。

図 3-18

Section3-8　たこ焼きを動かす

たこ焼きが止まったままなので動かしてみます。ルームエディタをいったん閉じます。

閉じるには、左上のチェックボタン をクリックします。閉じたら、リソースツリーの「Objects」から「obj_tako」をダブルクリックしてオブジェクト設定を開きます。

ここで簡単にオブジェクト設定画面の説明をします。真ん中にある「Events」というのがイベントの設定です。右の「Actions」というのがイベントに対応するアクションの設定です（図3-19）。イベントやアクションが何か、という説明は置いておいて、ひとまずオブジェクトを動かすイベントとアクションを設定してみます。

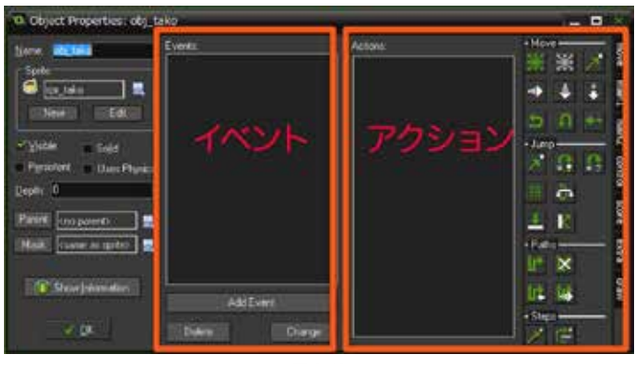

図 3-19

まず、「Events」カテゴリの下にある［Add Event］ボタンを押します。するとイベントの追加画面が表示されますので、ここから「Create」を選びます（図3-20）。

図 3-20

Eventsの枠の中に「Create」が追加されます（図3-21）。「Create」とは日本語で「生成」という意味で、これにより生成イベントを定義したことになります。生成イベントとはオブジェクトが作られた（インスタンス化した）ときに1度だけ発生するイベントです。もし間違ったイベントを選んでしまったら［Delete］ボタンを押して削除し、「Add Event」でやり直してください。

図 3-21

続いて、右にあるアイコンの中から「緑色の8方向の矢印」のものを「Actions」の枠の中にドラッグ＆ドロップします（図3-22）。

図 3-22

設定画面が表示されます（図3-23）。項目は以下のように設定します。

・Applies to: Self
・Directions: 右上・右下・左下・左下の矢印をクリック（赤くする）

・Speed: 2
・Relative: チェックなし

図 3-23

これは移動方向を決めるアクションで、斜め方向にランダムに2の速度で移動する設定となります。設定できたら［OK］ボタンを押します。なお、間違ったアイコンを設定してしまったら、配置したアイコンを選択して［Delete］キーで削除することができます。

そうしたら［F5］キーなどでゲームを実行します。実行するとたこ焼きが斜めに移動して画面外に出ていきます。

イベントとは

たこ焼きを動かすことができたので、その仕組みを説明したいと思います（図3-24）。

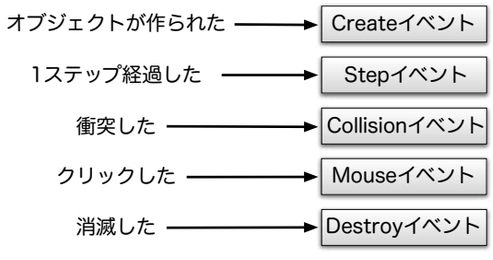

図 3-24

イベントとは、オブジェクトに対して外部から何らかの操作があった際、その通知を受け取る入り口となります。ここでは「Create」イベントを定義しましたが、これはオブジェクトをインスタンス化した際に呼び出されるイベントとなります。「Create」とは生成を意味しますが、オブジェクトが「出現」するときに発生するイベント、と覚えておいてもいいかもしれません。イベントにはこれ以外に、オブジェクトがクリックされたとき、オブジェクトが消滅したとき、別のオブジェクトと衝突したときなど、さまざまなものが用意されています。他のイベントについては、ミニゲームを作りながら少しずつ説明したいと思います。

アクションとは

アクションとは、発生したイベントに対応する動作を定義するものです。ここでは「Create」イベントの発生に合わせて、「Move Fixed」というアクションを使用しました。これは8方向移動するアクションです。これにより生成時のみ移動速度を設定する、という処理を実装できます。

アクションには、他にもオブジェクトの移動や生成、スプライトやフォントの描画やサウンド再生などがあります。そのあたりも作りながら説明したいと思います。

Applies toとは

「Move Fixed」アイコンの設定にあった「Applies to」という項目は、このアクションで動かす対象を選択します。

・Self: 自分自身を動かす
・Other: 衝突イベントのときのみ有効。ぶつかられた相手を動かす

・Object: 特定のオブジェクトを動かす

「Self」というのは自分自身を動かします。通常は自分を動かすので、たいていはこの項目を選びます。「Other」は他人という意味です。衝突イベントの際、ぶつかられたオブジェクトを操作したい場合に使用します。「Object」は特定のオブジェクトをまとめて動かすときに使用します。

memo エラー画面について

アクションに間違った設定をすると、図3-25のようなエラー画面が表示されることがあります。

図 3-25

これは「Speed」の値に「2」という数値ではなく、「abc」という文字を指定してしまったときのものです。英語ばかりで読むのが大変かもしれませんが、このメッセージにはエラーとなった重要な手がかりがあるので、しっかり読むと原因がある程度つかめます。

```
FATAL ERROR in
action number 1
of Create Event
for object obj_tako:


Push :: Execution Error - Variable Get -1.abc(100000, -2147483648)
at gml_Object_obj_tako_CreateEvent_1 (line 1) - action_move( "101000101", abc );
```

エラーの原因を見つけるための情報としては3行目の「of Create Event」というものです。「このエラーはCreateイベントで発生しました」ということを表しています。4行目の「for object obj_tako」は、「そのイベントはobj_takoのオブジェクトにある」となります。つまり「obj_takoのCreateイベントでエラーが発生した」という情報が読み取れるわけです。

そして、「Push :: Execution Error - Variable Get -1.abc(100000, -2147483648)」のところに注目します。「Variable」というのは変数という意味です。変数「abc」を取得しようとして何か問題が起きた、ということを表しています。最後の行は、エラーが起きたスクリプトの行の情報です。action_move関数を呼び出しています。この関数はマニュアルにもない非公開関数です。おそらくなんらかの移動値を設定するための関数と思われます。

細かいところまでは理解する必要がありませんが、どのオブジェクトのどのイベントで問題があるかは、このエラー情報をもとに推測できます。ですので、エラーメッセージを読まずにすぐ閉じるのではなく、落ち着いてエラーメッセージを読むと原因の特定が素早くできる可能性があります。エラー画面は大事なメッセージを伝えてくれる友達です。怖いものではありません。

Section3-9　ゲーム管理オブジェクトの作成

たこ焼きを動かすことができましたが、すぐに画面外に出てしまうので壁を作りましょう。と、その前にデバッグ機能を作りたいと思います。

リソースツリーの「Objects」を右クリックして「Create Object」を選び、名前を「obj_gameMgr」に変更します。

イベントを「Add Event」で追加します。イベントには「Key Press」を選び、「Letters > R」を選択します（図3-26）。このイベントは［R］キーを押したときに発生するイベントとなります。

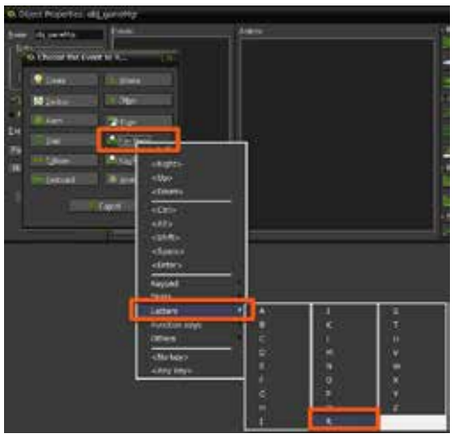

図 3-26

アクションは右のアイコンタブから「main2」タブを選択し、「Game」カテゴリから「Restart Game」をアクションにドラッグ＆ドロップします（図3-27）。このアイコンはゲームを最初からやり直すアクションとなります。

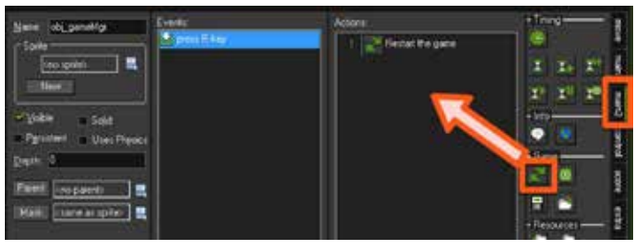

図 3-27

これで［R］キーを押すと、ゲームを始めからやり直すことができます。ただし、オブジェクトはルームに配置しないと動作しないのでしたよね。［OK］ボタンでオブジェクト設定画面を閉じます。

そして、リソースツリーの「Rooms」から「rm_main」を開きます。配置できるオブジェクトが「obj_tako」になっていますので、この場所をクリックして「obj_gameMgr」を選び、変更します（図3-28）。

図 3-28

そうしたらルームの左下あたり、X:64 Y:384に配置します（だいたいでかまいません）。座標は、ルームエディタの一番下にX座標とY座標が表示されています（図3-29）。この位置にしたのは、画面の隅にしてしまうと、この後に配置する壁オブジェクトに隠れてしまうからです。なお、スプライトが割り当てられていないオブジェクトは「？」のアイコンになり、ゲーム画面には表示されません。

図 3-29

配置できたら実行します。実行して［R］キーを押すたびに、たこ焼きが初期位置に戻るのを確認してください。

なぜこのようなデバッグ機能を作るかというと、何か問題が起きたときに、何度もゲームを起動し直すのは効率が良くないです。そこで、このようなデバッグ機能を先に作っておくと、繰り返し動作を確認することが簡単にできますので、問題の原因を探りやすくなります。なお、［R］キーにしたのは「Restart」の頭文字をとってこのキーに割り当てています。どのキーにするのかは好みの問題ですので、別のキーに割り当てても OK です。

Section3-10　壁オブジェクトの作成

デバッグ機能を組み込んだので壁を作ります。リソースツリーから「Sprites」を右クリックして「Create Sprite」からスプライトを作成します。名前は「spr_wall」とします。そうしたら［Load Sprite］ボタンを押して、壁の画像ファイルをダウンロードした素材のフォルダ「Materials/minigame」から「wall2.png」を読み込みます。

名前の設定ができて画像を読み込めたら、［OK］ボタンで閉じます（図3-30）。

図 3-30

次に、壁スプライトを使って壁オブジェクトを作成します。リソースツリーから「Objects」を右クリックして「Create Object」からオブジェクトを作成します。名前は「obj_wall」とし、スプライトには「spr_wall」を割り当て、［OK］ボタンを押します（図3-31）。

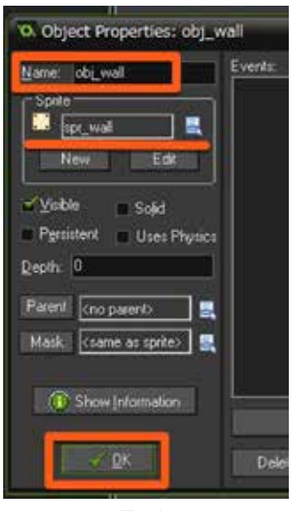

図 3-31

Section3-11　壁の配置と衝突判定の実装

壁の配置

リソースツリーの「Rooms」から「rm_main」を開きます。「obj_wall」を選択して外周を囲むように配置します（図3-32）。

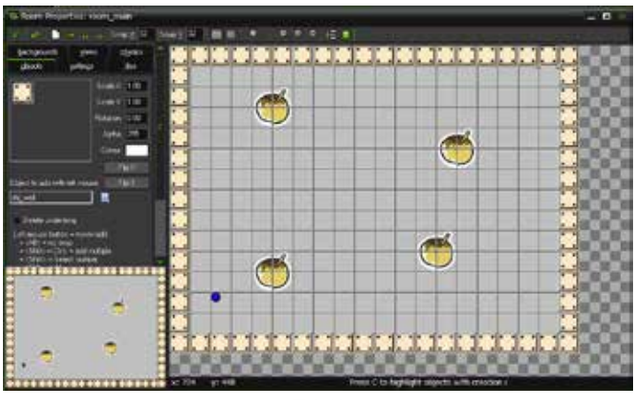

図 3-32

[Shift] + [Ctrl] キーを押しながら左ドラッグで連続して配置できます。間違えてしまったら、[Ctrl] キーを押しながら右ドラッグで連続で削除できます。

配置できたら実行します。ですが、衝突判定がないのでたこ焼きが壁をすり抜けていってしまいます。

衝突判定の実装

衝突判定を実装します。まずは「obj_wall」を開きます。そして、左の真ん中少し上あたりにある「Solid」にチェックを入れます（図3-33）。私はこのソリッドという単語を見るたびに某スネークさんを思い出してしまうのですが、特に関係はなく「固体」という意味となります。オブジェクトにこの属性を付けると、衝突判定が簡単にできるようになります。

図 3-33

続けて「obj_tako」を開きます。「Add Event」から「Collision」を選びます。これが衝突イベントとなります。すると衝突対象の選択がポップアップするので、「obj_wall」を選びます（図3-34）。これで「obj_wall」との衝突イベントが定義できました。間違ったイベントや対象を選んでしまった場合は、下にある [Delete] ボタンで削除できます。

図 3-34

次に、衝突イベントに対応するアクションを設定するのですが、「obj_wall」との衝突イベントを選択できている（青色になっている）でしょうか？（図3-35）

図 3-35

「Create」イベントが選択状態となっていると、生成イベントのアクションを定義することになります。そのため、どのイベントに対するアクションを定義しているかどうかは常に意識する必要があります。

衝突イベントには「moveタブ > Jumpカテゴリ」にある「Bounce」をアクションにドラッグ＆ドロップして指定します（図3-36）。これは衝突方向に対して反射する動きをするアクションとなります。

図 3-36

設定項目は以下のように指定します。

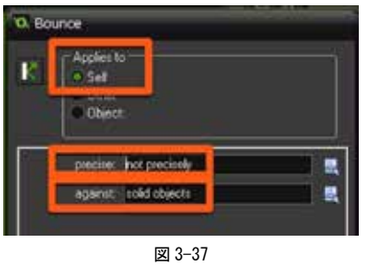

図 3-37

・Applies to: Self
・precise: not precisely
・against: solid objects

上から順に説明すると、「Applies to」の対象者には、現在のオブジェクトを意味する「Self」を指定しています。「precise」は厳密な衝突を行うかどうかです。例えば斜面との衝突判定をする際に、これを「precise」にすると正確に反射するようになります。ただし計算量が増えるので処理が重たくなります。ひとまず、ここでは「not precisely」としています。「against」は反射する対象のオブジェクトです。「solid object」にすることでソリッドオブジェクトのみ反射することとなります。「all」を選ぶとすべてのオブジェクトと反射することになります。

これで衝突判定が実装できました。実行してみると、たこ焼きが壁に跳ね返ってくれます。うまくいかない場合は、設定をどこかで間違えていないかどうか見直してみましょう。

・衝突イベントを「obj_wall」との衝突に設定できているかどうか
・衝突イベントに「Bounce」アクションを設定できているかどうか

Section3-12　クリックでたこ焼きを消す

それなりに動くようになりましたが、これだけではまだ見ているだけのアプリケーションです。プレイヤーからの入力を受け付けるようにしてみます。

「obj_tako」を開いて、「Add Event」でイベントを追加します。「Mouse」を選んでマウス入力を受け付けるようにします。

そして、ポップアップから「Left pressed」を選択します（図3-38）。これはたこ焼きが左クリックされたときに発生するイベントとなります。

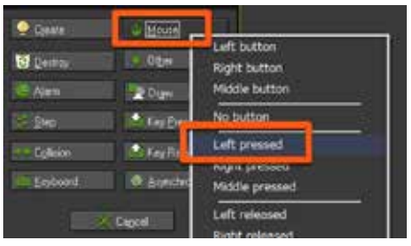

図 3-38

このイベントのアクションには消滅アクションを指定します。

「main タブ > Objects カテゴリ」からゴミ箱のアイコン「Destroy Instance」をドラッグ＆ドロップします（図3-39）。これでクリックするとたこ焼きが消滅します。ついでに消滅エフェクトを設定します。

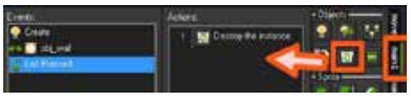

図 3-39

「draw タブ > Other カテゴリ」から花火のようなアイコン「Create Effect」をドラッグ＆ドロップします（図3-40）。

図 3-40

設定項目がいろいろありますが、ひとまず「color」の色を好きな色にします。ここではオレンジ色にしました（図3-41）。［OK］ボタンを押して設定を閉じます。

図 3-41

では、実行してみてください。たこ焼きをクリックすると消滅するようになりました。

Section3-13　エフェクトの設定とRelative

マウスクリックで消滅はしましたが、エフェクトが左上に出るというおかしな挙動をしています。しかも壁に隠れてしまっています。

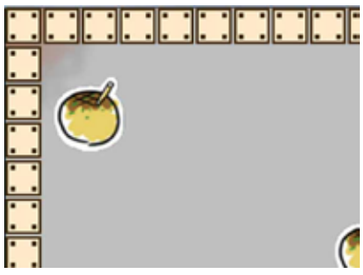

図 3-42

エフェクトは倒した敵のところにちゃんと出て欲しいですよね。この原因は先ほどの「エフェクト生成」アクションの設定にあります。「Create Effect」アイコンをダブルクリックして設定を開き、「Relative」にチェックを入れて［OK］ボタンを押して閉じます。実行すると、敵のあたりにエフェクトが出るようになります。

この「Relative」という設定は、アクションに何らかの「座標」を設定する場合に意識しなければならないものです。Relativeというのは相対的という意味です。これにチェックをしない場合は、指定した座標はルーム内の絶対的な座標となります。ルームでは左上が(0, 0)となるので、Relativeにチェックを入れないと、指定した座標が(0, 0)の場合、左上となるのです。

逆にRelativeにチェックを入れると、現在のオブジェクトの位置からの相対的な座標を基準に、出現座標が決定されます。

エフェクト生成の設定項目は説明は以下のようになります。

・type: エフェクトの種類
・x: 生成座標X
・y: 生成座標Y
・size: エフェクトのサイズ(small=小さい / medium=中 / large=大きい)
・color: 色
・where: 描画デプス(below objects=再背面 / above objects=最前面)

typeやsizeなどは好みで指定していいと思います。最後の「where」について説明すると、これはエフェクトの描画を画面の奥にするか手前にするかの設定となります。通常は最前面に表示したほうが違和感がないので、「above objects」にするとよいと思います。

Section3-14　エフェクトの出現位置の調整

Relativeの設定をしましたが、それでも出現位置がやや左上にずれているようです。これには原因があります。スプライトは標準では、左上を基準に描画するからです（図3-43）。

図 3-43

それを確認するためにリソースツリーから「spr_tako」を開きます。左下にOriginとあり X=0 Y=0 となっています。Originとは描画の基準座標で、それが(0, 0)となっているため、エフェクトが左上にずれてしまうのです。

これを基準座標を画像の中央に修正するには、Originカテゴリの Centerをクリックします。すると、X=32 Y=32 という値にセットされます。画像サイズは64x64なので、ちょうど中心の座標になりました（図3-44）。

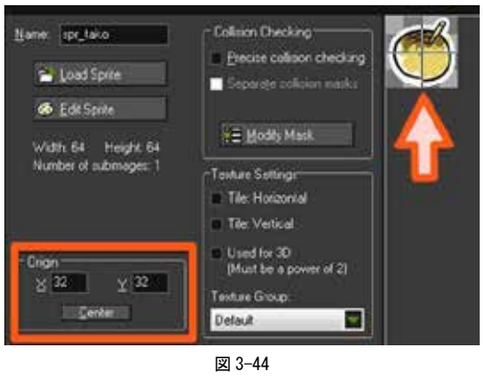

図 3-44

実行してみると、エフェクトが敵の中央から発生することが確認できます。

Section3-15　ゲームクリア判定を作る

ゲームクリア判定を作ります。リソースツリーから「obj_gameMgr」を開き、「Add Event」から「Step」イベントを選びます。

「Step / Begin Step / End Step」というポップアップが表示され、その中から「Step」を選びます（図3-45）。これは毎ステップ呼び出されるイベントとなります。ステップというのはオブジェクトの更新の最小単位となります。例えば、このミニゲームでのたこ焼きは、2ピクセルの移動を1秒間に60回行っています。この「60」という数値はルームで設定したSpeedの値となります。この60回の移動の1つが1ステップという単位となります。アニメーションの1コマが1ステップにあたる、というイメージです。

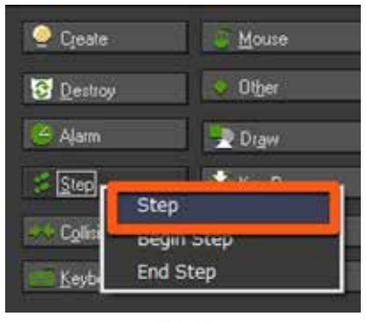

図 3-45

よくわからなければ、細かく何かを更新したりチェックしたい場合は「Step」イベントを使用する、と理解しておくとよいでしょう。

Stepイベントには「Controlタブ > Questionカテゴリ」の「Test Instance Count」アイコンをドラッグ＆ドロップします（図3-46）。これは特定のオブジェクトの存在する数をチェックするアクションです。

図 3-46

設定は以下のようにします（図3-47）。

・object: 対象となるオブジェクト。「obj_tako」を指定
・number: 比較対象となる値。「0」を指定
・operation: 符号。「equal to」を指定
・NOT: チェックしない

図 3-47

「operation」に指定した「equal to」というのは、「比較する値同士が一致したら」という意味となります。比較するのは「objectに指定したオブジェクト数」と「number」に指定した値です。これにより、「obj_takoの数」＝「0」（obj_takoが全滅した）の場合は何らかの処理をする、ということができるようになります。

今までは条件をイベントで実装してきましたが、敵が全滅した、というイベントはないので、このように「条件式のアイコン」を使い、独自の条件を定義してチェックを行います。

では、全滅したときの処理のアイコンを配置してみます。と、そのまえに「Controlタブ > Otherカテゴリ」の上向きの△「Start Block」と下向きの▽「End Block」を配置します（図3-48）。

図 3-48

このアイコンを配置すると、アイコンの位置がその区間だけ少し右にずれます。この三角のアイコンですが、「条件式アイコン」が成立した場合にその下の△～▽までの区間のみを実行するという意味となります。つまり、obj_takoが0になったら、この区間のアクションが実行されることとなります。

続けて、「main2タブ > Infoカテゴリ」から吹き出しアイコンの「Display Message」を上下の矢印の間に配置します（図3-49）。これはダイアログでメッセージを表示するアクションとなります。

図 3-49

「message」には「ゲームクリア！」というメッセージを指定します（図3-50）。

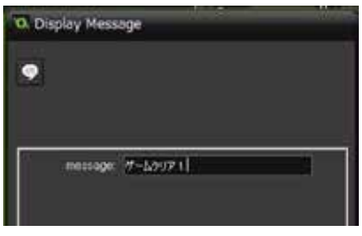

図 3-50

［OK］ボタンを押して設定できたら実行してみます。

敵を全滅させると「ゲームクリア！」というメッセージが表示されます。……ですが、困ったことにダイアログを閉じても、何度もダイアログが表示されてしまいます。こういうバグが発生すると焦ってしまいがちですが、落ち着いてGameMaker:Studioにある「停止」アイコン  をクリックして、クールに対処しましょう。この停止アイコンを押すと、実行中のゲームを強制停止できます。ふぅ。

さて、なぜこのようなことが起きたかというと、毎ステップ全滅をチェックし、全滅したら毎ステップの間、ダイアログメッセージを表示するようにしてしまったためです。これを回避するにはいろいろ方法があるのですが、今回はゲームクリア時にはゲームを終了することにします。

先ほどの「Display Message」アイコンの下に「End Game」アイコンを配置します。

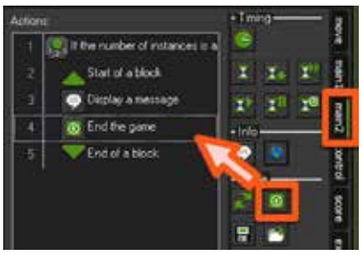

図 3-51

「End Game」アイコンは「main2タブ > Gameカテゴリ」にあります。実行すると、敵を全滅させるとダイアログが表示され、ゲームが終了します。

Section3-16　ゲームを配布できる形式に出力する

最後の敵をクリックした瞬間にゲームが終わってしまうので、若干の違和感がありますが、ひとまずゲームとしては完成です。完成したということで、これをGameMaker:Studioを持っていない人でも遊べる形式に出力したいと思います。

出力するには、メインツールバーから保存アイコンの隣にあるアイコン（Create Executable for Target） をクリックします。メインメニューの「Flie」から「Create Application...」を選んでも同じことができます。

すると、出力先と形式を選択するダイアログが表示されます。違う画面が出る場合は、いったんキャンセルして、メインツールバーの「Target」が「Windows」になっているかどうか確認してください。このダイアログの「ファイルの種類」から出力する形式を指定します。デフォルトだとインストーラ形式になっていますが、この形式は遊ぶ人に嫌がられることが多い（レジストリが汚れる）ため、別の形式がよいでしょう。1つのEXEファイルにまとまる「Single runtime executable (*.exe)」の形式が一番良いのでこれを選びます（図3-52）。

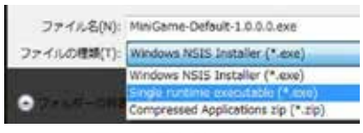

図 3-52

そうしたら好きな場所を選んで「保存」を選びます。すると黒い画面が表示され、何か危険なアプリが動いているでは、という気になりますが、圧縮しているだけなので特に問題はありません。しばらくすると処理が終わり、実行ファイルが作成されます。作成された実行ファイルをダブルクリックして、正常に動作することを確認してください。

これで配布できる実行ファイルが作成できました！……ですが、アイコンがGameMaker:Studioのものとなっています。アイコンはゲームの顔となるのものです。このアイコンで配布したら、残念なゲームと思われても仕方ありません。そこでオリジナルのアイコンを設定します。

図 3-53

リソースツリーから「Global Game Settings」をダブルクリックで開くとゲームの設定画面が表示されます。ここからWindowsタブを選び、Optionカテゴリの Game iconの隣にある［Update］ボタンを押します。するとファイル選択ダイアログが表示されるので、ダウンロードした素材の「Materials/minigame」フォルダから「tako.ico」を選びアイコンを設定します。アイコン画像がはみ出してしまいますが、とくに問題はありません。再び実行ファイルを作ると、アイコン画像がたこ焼きの画像に変更されています。

これで完成です。

Section3-17　ゲームルールの見直し

ゲームが完成したので、内容を一度見直してみます。今回のゲームのルールは4つです。

条件	結果
たこ焼きの出現時	ランダムで斜めに動く
たこ焼きが壁に衝突する	跳ね返る
たこ焼きをクリックする	消滅する
たこ焼きが全滅する	ゲームクリア

ルールは「条件」と「結果」により成り立ちます。「条件」はほぼイベントとして定義されているものですね。最後のクリア条件だけはStepイベントで実装しました。その条件に対応する結果は、アクションとして定義したものとなります。

このように、イベントによってシステムが駆動することを「イベントドリブン」と呼びます。GameMaker:Studioはこの仕組みを使ってゲームを作ることとなります。イベントによってルールが明確に分離されるので、見通しの良い作りにすることができるのです。

今回のミニゲームは、GameMaker:Studioに慣れてもらうための練習用なので、あまり面白いゲームとならなかったかもしれません。そこで、このゲームを面白くするためのルールを考えてみます。

条件	結果
10秒経過する	ゲームオーバー
たこ焼きが壁に衝突する	跳ね返り「移動速度が減少」する

それと視界を制限するための黒い枠を入れてみます。

このルールを採用したミニゲームの実行ファイルは「Exe/MiniGame_Ext.exe」をダブルクリックで実行できます。

図 3-54

今回紹介しなかった機能も使っていますが、ここでは簡単に説明するだけにとどめておきます。

・時間制限を実装するにあたって、アラームイベントを使用しています。
・制限時間はフォントを使って描画しています。
・視界を制限するためのオブジェクト（obj_view）はマウスの座標 (mouse_x, mouse_y) を中心に配置しています。

プロジェクトは「MiniGame_Ext.gmx」にあります。興味のある方はプロジェクトを開いて、中を見てもいいかもしれません。アクションにはコメントを書いているので、なんとなく理解できるようにしています。ここで使っている機能は次のシューティング作成のときに説明します。

ひとまずは「ルールを追加することでゲームが面白くなるんだ」ということを理解してもらえるといいかな、と思います。特に「ゲームオーバー」が存在するのは重要です。これにより緊張感が生まれ、面白さがアップします。

ということで、今後ゲームを作るとき「ゲームが面白くならない」と悩むことがあったら、とにかくルールを足していく、という方法をおすすめします。たくさんの遊べる要素を次々と組み込んでいけば、いつか「これは面白い」という要素を発見できます。もし結果としてそのゲームが面白くならなかったとしても、そういった挑戦は、次に作るゲームに応用できることも多々あります。失敗を恐れずチャレンジしていくことが、面白いゲームを作るための近道であると私は考え

ています。

Section3-18　まとめ

今回登場したリソースの種類についておさらいをします。

・ルーム: ゲーム画面。オブジェクトを配置するところ
・スプライト: 画像を読み込んで管理する
・オブジェク: ゲームキャラクターやゲームシステムの制御をする。

GameMaker:Studioは、他にもさまざまなリソースが存在しますが、ひとまずはこの3つがあれば、最低限のゲームは作れるようになります。

memo Relativeについて

Relativeはアクションアイコンを設定する上でとても重要な概念なので、もう一度ここで説明をします。

GameMaker:StudioではアクションアイコンにRelativeを設定できる場合、デフォルトではチェックされていません。Relativeにチェックがされていない場合、設定する値は絶対的なものとして扱われます。例えば座標を設定するアクションであるとし、現在のオブジェクトのX座標が「50」、設定する座標が「100」の場合には、

・Relativeにチェックがない→X座標は「100」
・Relativeにチェックをする→X座標は「50 + 100 = 150」

となります。つまりRelativeが無効だと、指定した値をそのまま設定し、Relativeを有効にすると、現在の値を基準にその値を足したり引いたりすることができる、ということになります。

memo オブジェクトとインスタンス

GameMaker:Studioを理解する上で重要な概念である「オブジェクト」と「インスタンス」の違いについて説明します。今回のミニゲームでは「たこ焼き」オブジェクト、「壁」オブジェクト「ゲーム管理」オブジェクトを作成しました。ですがオブジェクトを作るだけではゲーム上では動いてくれません。動かすためには、それらのオブエジェクトをルームに配置する必要があります。

このように、ルームに配置されたオブジェクトを「インスタンス」と呼びます。インスタンスとは聞き慣れない言葉ですが、ここでは「実体」という意味を表します。オブジェクトはインスタンスになることで、ゲーム内で動き出すようになります。

この違いは似ているようでまったく違う概念です。オブジェクトは設計図で、インスタンスは設計図を元に作られたロボットのようなものです。今はまだピンとこないかもしれませんが、このことを知っておくと理解度に大きく影響が出ますので、なんとなくでも「オブジェクトとインスタンスは別のもの」ということを意識しておくとよいでしょう。

なお、このミニゲーム作成では、意図的にオブジェクトとインスタンスを混同して説明しましたが、今後は明確に使い分けて説明することにします。

memo リソース名の付け方

リソース名は重複するものでなければ基本的にどんな名前も付けることができます。ですが、あまりにも適当な名前（デフォルトの「object0」など）にすると後々苦労します。というのもリソースはいろいろなところからアクセスします。その際に、「あのリソースの名前は何だったけ」と目的のリソース名がわからなくなってしまうと、作業効率が下がります。ですので、名前だけでそれが何を意味するのかわかるものにしておいたほうがいいです。

例えば私の場合は、リソース名に以下のような接頭語を付けるようにしています。

リソースの種類	接頭語
スプライト	spr_
サウンド	snd_
背景	bg_
パス	pth_
スクリプト	scr_
フォント	fnt_

タイムライン	tml_
オブジェクト	obj_
ルーム	rm_