いるかどうかをチェックする関数です。続いてSpriteクラスです。このクラスは、このゲームのすべてのスプライトクラスの親クラスです。その次は、PlatformSpriteクラスと、作成中のStickFigureSpriteクラスです。PlatformSpriteクラスは、スティックマンが飛び乗る床オブジェクトを作るときに使いました。そしてゲームのメインキャラクターを表すのに、StickFigureSpriteクラスのオブジェクトを作りました。

```
def within_x(co1, co2):
    ...
def within_y(co1, co2):
    ...
def collided_left(co1, co2):
    ...
def collided_right(co1, co2):
    ...
def collided_top(co1, co2):
    ...
def collided_bottom(y, co1, co2):
    ...
class Sprite:
    ...
class PlatformSprite(Sprite):
    ...
class StickFigureSprite(Sprite):
    ...
```

プログラムの最後では、これまで作ったクラスのオブジェクトを作ります。Gameクラスのオブジェクトや、PlatformSpriteクラスのオブジェクトです。一番最後の行でmainloop関数を呼んでいます。

```
g = Game()
platform1 = PlatformSprite(g, PhotoImage(file="platform1.gif"), \
    0, 480, 100, 10)
...
g.sprites.append(platform1)
...
g.mainloop()
```

第18章の最後に、このゲーム全体のプログラムがあります。自分のプログラムがちょっと違って見えたり、動かなかったりする場合は参考にしてみてください。

17.5 この章でわかったこと

この章ではスティックマンの画像のクラスを作りはじめました。いまのところ、このクラスのオブジェクトを作っても、スティックマンのアニメーションに必要な画像の読み込みと、あとで使ういくつかのオブジェクトの変数のセットアップ以外は、特に何もしません。このクラスには、キーボードイベント（プレイヤーが左右の矢印キーやスペースキーを押したとき）に基づいて変数の値を変更するいくつかの関数があります。

次の章では、このゲームを完成させます。スティックマンをアニメーションさせてキャンバス上で動かすための関数をStickFigureSpriteクラスに追加します。そして、ミスター・スティックマンが脱出するための出口（ドア）も作ります。

第18章

ミスター・スティックマン脱出ゲームを完成させよう

前の三つの章で、ミスター・スティックマン脱出ゲームを開発してきました。画像を作ったり、背景、床、スティックマンを追加するプログラムを書いたりしました。この章では、スティックマンやドアをアニメーションさせる部分を作ります。

章の最後には、完成したミスター・スティックマン脱出ゲームのプログラムがあります。プログラミング中に迷ったり混乱したりした場合は、自分のプログラムと見比べて間違いを見つけましょう。

18.1 スティックマンをアニメーションさせる

これまでは、利用する画像の読み込みをしたり、関数にキーボードのキーを結びつけたりする、スティックマンの画像のクラスの基本的な部分を作ってきました。しかし、まだプログラムを実行しても特におもしろいことは起きません。

残りの関数（animate、move、coords）を、第17章で作ったStickFigureSpriteクラスに追加していきましょう。animate関数は、さまざまなスティックマン画像を描きます。move関数は、キャラクターが動く場所を決めます。coords関数は、スティックマンの現在位置を返します（床のスプライトと違って、スティックマンは画面上を動き回るので、位置を再計算する必要があります）。

■アニメーション関数を作る

はじめにanimate関数を追加します。この関数は、動きをチェックし、動きに応じて画像を変更します。

● 動きをチェックする

スティックマン画像の入れ替えが速すぎると、アニメーションがおかしくなってしまいます。メモ帳の隅に絵を描くパラパラ漫画を考えてみてください。メモ帳を速くめくりすぎると、描いた絵がきれいなアニメーションにならないのと同じです。

animate関数の前半は、スティックマンが左または右に走っているかをチェックします。また、スティックマンの画像を切り替えるかどうか判断するために、last_time変数を使います。この変数は、アニメーションのスピードを制御するのに役立ちます。animate関数は、第17章でStickFigureSpriteクラスに追加したjump関数のあとに続けて書きます。

```
    def jump(self, evt):
        if self.y == 0:
            self.y = -4
            self.jump_count = 0

    def animate(self):
①       if self.x != 0 and self.y == 0:
②           if time.time() - self.last_time > 0.1:
③               self.last_time = time.time()
④               self.current_image += self.current_image_add
⑤               if self.current_image >= 2:
⑥                   self.current_image_add = -1
⑦               if self.current_image <= 0:
⑧                   self.current_image_add = 1
```

①のif文で、スティックマンが左または右に動いているかどうかを確認するため、xが0ではないことをチェックします。同時に、スティックマンがジャンプしていないことをチェックするため、yが0であることもチェックします。もし、このif文の条件がTrueとなるときには、スティックマンをアニメーションさせる必要があります。if文の条件がFalseの場合、彼は動いていない状態なので、アニメーションさせる必要はありません。

②では、前回animate関数が呼ばれたときから経過した時間を計算しています。これは、現在時刻からlast_time変数の値を引くことで求めます。現在時刻はtime.time()を使えば取得できます。この時間を計算する理由は、スティックマンの画像を次の画像に入れ替えるかどうかを決めるためです。計算結果が0.1秒より大きい場合は、③へ進みます。そして、last_time変数に現在時刻をセットします。次に画像を入れ替えるタイミングを計るためにストップウォッチをリセットするようなものです。

④では、current_image_add変数の値を、current_image変数に加えています。current_image変数は、現在表示されているスティックマン画像のインデックス値です。第17章でStickFigureSpriteクラスの__init__関数にcurrent_image_add変数を作りました。animate関数がはじめて呼ばれたときは、あらかじめcurrent_image_add変数に1がセットされています。

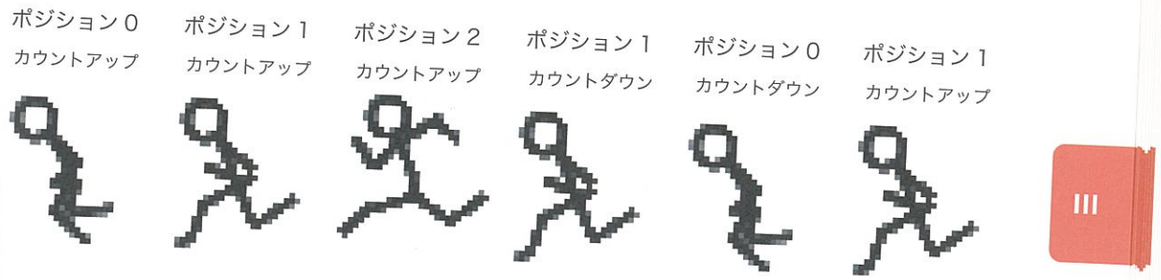
⑤では、current_imageのインデックス値が2以上かどうかをチェックしています。2以上の場合は、current_image_addの値を−1へ変更します（⑥）。⑦の手順もよく似ています。current_imageの値が0以下になった場合は、current_image_addの値を1へ変更します（⑧）。

NOTE　このプログラムのインデントがわからなくなった場合のヒント：①のifの前には8個のスペースがあります。⑧のselfの前には20個のスペースがあります。

ここまでのanimate関数で何が行われているのか考えてみましょう。床の上に一列に並んだ、色つきブロックを想像してください。それぞれのブロックには番号がついています（1、2、3、4、...）。あるブロックから次のブロックへと指を差していきます（current_image変数）。指が一度に移動するブロック数が、current_image_add変数の値です。一列に並んだブロックを、一方向へ指差していく場合、毎回1を足していきます。指がブロック列の端にたどり着いたら、逆方向に戻していきます。この場合は、毎回1を引きます（「毎回−1を足す」と同じ意味）。

animate関数に追加したプログラムは、このような処理を行います。ただし、色つきブロックではなく、リストに保存されたスティックマンの画像（左右方向それぞれ3画像ずつ）を使います。これらの画像のインデックス値は0、1、2です。スティックマンをアニメーションさせるので、最後の画像にたどり着いたらインデックス値をカウントダウンします。またはじめの画像にたどり着いたらカウントアップします。これにより走るスティックマンのアニメーションができあがります。

下の図は、animate関数で計算したインデックス値とスティックマンの画像のリストを使って彼をアニメーションさせる方法を表したものです。

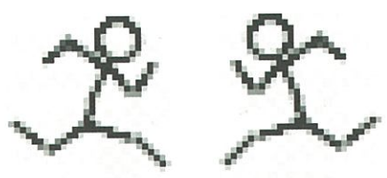


● 画像を変更する

animate関数の後半では、表示する画像を計算したインデックス値を使って変更します。

```
    def animate(self):
        if self.x != 0 and self.y == 0:
            if time.time() - self.last_time > 0.1:
                self.last_time= time.time()
                self.current_image += self.current_image_add
                if self.current_image >= 2:
                    self.current_image_add = -1
                if self.current_image <= 0:
                    self.current_image_add = 1
①       if self.x < 0:
②           if self.y != 0:
③               self.game.canvas.itemconfig(self.image, \
                        image=self.images_left[2])
④           else:
⑤               self.game.canvas.itemconfig(self.image, \
                        image=self.images_left[self.current_image])
⑥       elif self.x > 0:
⑦           if self.y != 0:
⑧               self.game.canvas.itemconfig(self.image, \
                        image=self.images_right[2])
⑨           else:
⑩               self.game.canvas.itemconfig(self.image, \
                        image=self.images_right[self.current_image])
```

xが0より小さい場合、スティックマンは左へ移動しています（①）。左へ移動させるプログラムは②〜⑤です。②では、yが0ではないことをチェックしています。yが0でない場合（スティックマンの画像が上または下に移動中、つまりジャンプ中）は、キャンバスのitemconfig関数を使って、表示されているスティックマンの画像を、左向きスティックマンの画像リストの最後の画像（images_left[2]）に変更します（③）。スティックマンはジャンプ中なので、ちょっとリアルっぽく見せるため、両足をいっぱいに広げている画像を使うわけです。



スティックマンがジャンプ中でない場合（yが0の場合）は、else文以下のプログラムが実行されます（④）。表示するスティックマン画像を、current_image変数をインデックス値としてリストから取り出し、itemconfigを使ってセットします。（⑤）。

⑥は、スティックマンが、右に走っているかどうかをチェックします（xが0より大きい場合）。右に走っている場合のプログラムは⑦〜⑩です。このプログラムは②〜⑤とよく似ています。スティックマンがジャンプ中かどうかをチェックし、それに適した画像へ変更します。ただし、利用する画像リストは右向きのもの（images_right）です。

■ スティックマンの位置を取得する

スティックマンはキャンバス上を動き回るので、どこにいるのかをチェックする必要があります。そのため、スティックマンの現在の位置を示すcoords関数は、他のSpriteクラスの関数とは違ったものになります。キャンバスのcoords関数を使ってスティックマン画像の位置を取得したら、その値をcoordinates変数のx1、y1、x2、y2へセットするようにしましょう。coordinates変数は、第17章のはじめのほうで、StickFigureSpriteクラスの__init__関数内で作りました。以下がプログラムです。animate関数の下に書きましょう。

```
            if self.x < 0:
                if self.y != 0:
                    self.game.canvas.itemconfig(self.image, \
                            image=self.images_left[2])
                else:
                    self.game.canvas.itemconfig(self.image, \
                            image=self.images_left[self.current_image])
            elif self.x > 0:
                if self.y != 0:
                    self.game.canvas.itemconfig(self.image, \
                            image=self.images_right[2])
                else:
                    self.game.canvas.itemconfig(self.image, \
                            image=self.images_right[self.current_image])

    def coords(self):
①       xy = self.game.canvas.coords(self.image)
②       self.coordinates.x1 = xy[0]
③       self.coordinates.y1 = xy[1]
④       self.coordinates.x2 = xy[0] + 27
⑤       self.coordinates.y2 = xy[1] + 30
        return self.coordinates
```

第16章で作ったGameクラスが持つ変数の一つにcanvasがあります。①では、このcanvas変数のcoords関数（self.game.canvas.coords）を使って、スティックマンの位置を取り出しています。この関数は、image変数の値（キャンバス上の画像の識別番号）を引数として受け取ります。

取り出したスティックマンの位置情報リストは変数xyにセットします。位置情報のリストには、二つの値が含まれています。coordinatesのx1変数に保存される画像左上のx座標と（②で設定）、coordinatesのy1変数に保存される画像左上のy座標です（③で設定）。

作成したスティックマン画像はすべて幅27ピクセル、縦30ピクセルなので、x2変数とy2変数の値は計算できます。x2は、x座標の値にスティックマン画像の幅27ピクセルを足したものです（④）。y2は、y座標の値に高さ30ピクセルを足したものです（⑤）。

そして関数の最後の行で、オブジェクトの変数coordinatesを呼び出し元に返します。

■スティックマンを動かそう

StickFigureSpriteクラスの最後の関数は、moveです。この関数は、ゲームのキャラクターを実際に画面上で動かす部分になります。この関数の中には、キャラクターが何かと衝突したことを調べるためのプログラムも書かれます。

● move関数を書きはじめる

以下はmove関数のはじめの部分です。coords関数のあとに書いていきましょう。

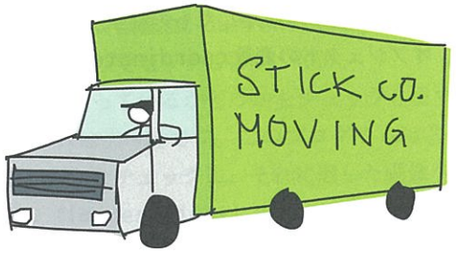
```
    def coords(self):
        xy = self.game.canvas.coords(self.image)
        self.coordinates.x1 = xy[0]
        self.coordinates.y1 = xy[1]
        self.coordinates.x2 = xy[0] + 27
        self.coordinates.y2 = xy[1] + 30
        return self.coordinates

    def move(self):
①       self.animate()
②       if self.y < 0:
③           self.jump_count += 1
④           if self.jump_count > 20:
⑤               self.y = 4
⑥       if self.y > 0:
⑦           self.jump_count -= 1
```

①では、この章の前のほうで作ったanimate関数を呼んでいます。この関数は、必要に応じて表示する画像を変更します。②では、yが0より小さいかどうかをチェックします。0より小さい場合、スティックマンはジャンプ中です。yがマイナスならスティックマンは画面内で上昇するからです（キャンバスの一番上のy座標が0、キャンバスの一番下のy座標が500ピクセルでしたね）。

③では、jump_countに1を足しています。④では、jump_countの値が20を超えてないかどうかチェックしています。超えている場合は、スティックマンを再び落下させるためyを4に変更します（⑤）。

⑥では、yが0より大きいかどうかをチェックします（yが0より大きい場合、スティックマンは落下しています）。yが0より大きい場合は、jump_countから1を引きます。20カウントアップしたなら、20カウントダウンしないといけませんからね（1から20まで数えながら、ゆっくりと手を上にあげてみてください。次に、20から1までカウントダウンしながら手を元に戻してみてください。スティックマンのジャンプカウントの方法や働きが感覚的にわかるはずです）。



move関数の続く行では、coords関数を呼び出しています。この関数はキャラクターがキャンバス上のどこにいるのかを教えてくれます。そのキャラクターの位置情報をco変数にセットします。それから、left、right、top、bottom、falling変数をそれぞれ作ります。

```
        if self.y > 0:
            self.jump_count -= 1
        co = self.coords()
        left = True
        right = True
        top = True
        bottom = True
        falling = True
```

それぞれの変数にTrueをセットしている点に注目してください。これらの変数は、キャラクターがキャンバス上で何かと衝突しているか、また、落下中かどうかを示すものとして使います。

● スティックマンがキャンバスの底辺や上端に衝突しているか？

move関数の続きの部分で、スティックマンがキャンバスの底辺または上端に衝突しているかをチェックします。

```
        bottom = True
        falling = True
①       if self.y > 0 and co.y2 >= self.game.canvas_height:
②           self.y = 0
③           bottom = False
④       elif self.y < 0 and co.y1 <= 0:
⑤           self.y = 0
⑥           top = False
```

キャラクターが画面内を落下しているとき、yは0より大きくなっています。その場合には、①で、まだキャンバスの底辺に衝突してないかどうかチェックしないといけません（さもないと画面の下側へと消え去ってしまいます）。キャラクターのy2座標（スティックマンの下端）がgameオブジェクトのcanvas_height変数の値より大きい、または等しいかどうかを調べて、条件がTrueになる場合は、スティックマンの落下を止めるため、yの値を0にします（②）。次にbottom変数にFalseをセットします（③）。これは、スティックマン下部の衝突をこれ以上チェックする必要がないことを示します。

スティックマンがキャンバスの上端に衝突しているかチェックする方法も、キャンバスの底辺に衝突しているかチェックする方法とよく似ています。まず、スティックマンがジャンプ中（yが0より小さい）かどうかをチェックします。次に、スティックマンのy1の座標が0より小さい、または等しいかどうかをチェックします（④）。小さい、または等しい場合は、彼はキャンバスの上端に衝突しています。どちらの条件もTrueの場合、スティックマンの上昇を止めるためにyの値を0にします（⑤）。また、top変数にFalseをセットします（⑥）。これは、スティックマン上部の衝突をこれ以上チェックする必要がないことを示します。

● スティックマンがキャンバスの左端や右端に衝突しているか？

スティックマンがキャンバスの左端や右端に衝突しているかを調べる方法も、前に出てきた上端や底辺の衝突チェックとほぼ同じです。

```
            elif self.y < 0 and co.y1 <= 0:
                self.y = 0
                top = False
①           if self.x > 0 and co.x2 >= self.game.canvas_width:
②               self.x = 0
③               right = False
④           elif self.x < 0 and co.x1 <= 0:
⑤               self.x = 0
⑥               left = False
```

xが0より大きい場合、スティックマンが右へ向かって走っているとわかります。また、スティックマンのx2座標（co.x2）がcanvas_widthに保存されたキャンバスの幅より大きい、または等しいかどうかを確かめれば、スティックマンがキャンバスの右端に衝突しているかがわかります。これら二つの条件がどちらもTrueの場合は（①）、スティックマンの動きを止めるためにxの値を0にします（②）。また、right変数にFalseをセットします（③）。④〜⑥も同様です。ただし、キャンバスの左端との衝突チェックをしています。

● 他のスプライトとの衝突

キャンバスの両端との衝突チェックが終わったところで、キャンバス内にある他の何かと衝突しているかをチェックします。以下が、そのプログラムです。gameオブジェクトに登録されているスプライトのリストを使って、スティックマンが他のスプライトと衝突しているかを調べます。

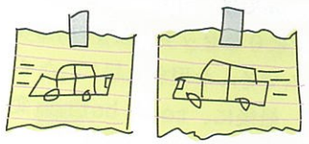
```
            elif self.x < 0 and co.x1 <= 0:
                self.x = 0
                left = False
①           for sprite in self.game.sprites:
②               if sprite == self:
③                   continue
④               sprite_co = sprite.coords()
⑤               if top and self.y < 0 and collided_top(co, sprite_co):
⑥                   self.y = -self.y
⑦                   top = False
```

①では、スプライトのリストからループを使って一つずつスプライトを取り出します。取り出したスプライトは、sprite変数にセットされます。②は、スプライトがselfと同じである場合です。この場合は衝突をチェックする必要はありません。スティックマンが自分自身と衝突していることを検出しても意味がないからです。sprite変数がselfと等しい場合は、リスト内の次のスプライトへ処理を移すため、continueを使います。

次に、スプライトの座標を取り出すために、スプライトのcoords関数を呼び出します（④）。取り出した座標は、変数sprite_coにセットしておきます。

⑤の部分では、下のような条件を調べています。

- スティックマンがキャンバスの上端に衝突していないこと（top変数がTrueのまま）。
- スティックマンがジャンプ中であること（yの値が0より小さい）。
- スティックマンの上部がスプライトリスト内のスプライトと衝突していること（第16章で作ったcollided_top関数を使う）。



これらがすべてTrueの場合、スティックマンを落下させるために、yの値をマイナス（-）を使って反転させます（⑥）。また、top変数にFalseをセットします（⑦）。なぜなら、スティックマンの上部の衝突をこれ以上チェックする必要はないからです。

● スティックマンの下部の衝突

ループの次の部分では、スティックマンの下部が他の何かと衝突しているかを調べます。

```
            if top and self.y < 0 and collided_top(co, sprite_co):
                self.y = -self.y
                top = False
①           if bottom and self.y > 0 and collided_bottom(self.y, \
                    co, sprite_co):
②               self.y = sprite_co.y1 - co.y2
③               if self.y < 0:
④                   self.y = 0
⑤               bottom = False
⑥               top = False
```

前に出てきたのとよく似た三つのチェックが①にあります。bottom変数がTrueのままかどうかのチェックと、スティックマンが下降（yが0より大きい）しているかどうかのチェックと、スティックマンの下部が他のスプライトと衝突しているかどうかのチェックです。三つすべてのチェックがTrueの場合は、スティックマン下部のy座標（y2）の値を、スプライト上部のy座標（y1）の値から引きます（②）。ちょっと変な処理に思えるかもしれません。なぜこのようなことをするのか考えてみましょう。

スティックマンが床から落ちたときを想像してみてください。彼はmainloop関数が実行されるたびに、4ピクセルずつ下降します。彼の足が他の床の3ピクセル上にあったとします。仮に彼の下部の座標（y2）が57ピクセルで、床の上部の座標（y1）が60だったとします。この場合、collided_bottom関数はTrueを返します。なぜなら、この関数はスティックマンのy2座標の値にyの値（ここでは4）を加えるからです。結果的にy2座標の値は61となります。

とはいえ、床やキャンバスの底辺と衝突したとたんにスティックマンの落下をすぐに止めたくありません。すぐに止めると、床から大きく飛び降りて地面の少し上の空中で停止したかのように見えるからです。すごい技になるかもしれませんが、このゲームにはふさわしくありません。その代わり、スティックマン下部のy2の値（57）を床上部のy1の値（60）から引き、3を取得します。

この値は、スティックマンが床の上部に着地するために落下するピクセル値となります。

③では、計算結果が0より小さい値かどうかをチェックしています。0より小さい場合（④）は、yに0をセットします（yが0より小さい値になった場合、スティックマンは上昇します。このゲームでそんな現象は起こしたくありません）。

最後に、topとbottom変数をFalseにセットします（それぞれ⑥と⑤）。スティックマンの上部と下部について、他のスプライトとの衝突をチェックする必要がなくなったためです。

もう一つ、スティックマンの下部の衝突をチェックします。スティックマンが床からはみ出ているかどうかのチェックです。

```
        if self.y < 0:
            self.y = 0
        bottom = False
        top = False
    if bottom and falling and self.y == 0 \
            and co.y2 < self.game.canvas_height \
            and collided_bottom(1, co, sprite_co):
        falling = False
```

以下の五つの条件がすべてTrueだった場合、falling変数にFalseをセットします。

- まだbottomをチェックする必要があること（bottom変数がTrue）。
- スティックマンが落下しているかどうかをチェックする必要があること（falling変数がTrue）。
- スティックマンが落下中ではないこと（yが0）。
- スティックマンの下部が、キャンバスの底辺に衝突していないこと（キャンバスの高さより小さい）。
- スティックマンが、床上部と衝突していること（collided_bottomの戻り値がTrue）。

● 左右をチェックする

スティックマンの上部や下部が他のスプライトと衝突しているかを調べました。次はスティックマンの左右が他のスプライトと衝突しているかを調べます。

```
            if bottom and falling and self.y == 0 \
                    and co.y2 < self.game.canvas_height \
                    and collided_bottom(1, co, sprite_co):
                falling = False
①           if left and self.x < 0 and collided_left(co, sprite_co):
②               self.x = 0
③               left = False
④           if right and self.x > 0 and collided_right(co, sprite_co):
⑤               self.x = 0
⑥               right = False
```

①では、まだスティックマンの左側の衝突を検出する必要があるか（left変数がTrue）と、スティックマンが左へ移動しているかどうかをチェックします（xが0より小さい場合は左へ移動しています）。また、collided_left関数を使って、他のスプライトと衝突しているかをチェックします。これら三つの条件がTrueの場合は、②でxを0にします（スティックマンの移動を止めるためです）。左側の衝突はこれ以上チェックする必要がないので、left変数をFalseにします（③）。

右側の衝突を検出するプログラムも左側の衝突を検出するプログラムとよく似ています（④）。三つの条件がTrueの場合はxを0にセットします（⑤）。右側の衝突はこれ以上チェックする必要がないので、rightをFalseにします（⑥）。

四つの方向について衝突をチェックするforループを使ったプログラムは以下のようになります。

```
elif self.x < 0 and co.x1 <= 0:
    self.x = 0
    left = False
for sprite in self.game.sprites:
    if sprite == self:
        continue
    sprite_co = sprite.coords()
    if top and self.y < 0 and collided_top(co, sprite_co):
        self.y = -self.y
        top = False
    if bottom and self.y > 0 and collided_bottom(self.y, \
            co, sprite_co):
        self.y = sprite_co.y1 - co.y2
        if self.y < 0:
            self.y = 0
        bottom = False
        top = False
    if bottom and falling and self.y == 0 \
            and co.y2 < self.game.canvas_height \
            and collided_bottom(1, co, sprite_co):
        falling = False
    if left and self.x < 0 and collided_left(co, sprite_co):
        self.x = 0
        left = False
    if right and self.x > 0 and collided_right(co, sprite_co):
        self.x = 0
        right = False
```

move関数に、ちょっとだけプログラムを追加します。

```
            if right and self.x > 0 and collided_right(co, sprite_co):
                self.x = 0
                right = False
①       if falling and bottom and self.y == 0 \
                and co.y2 < self.game.canvas_height:
②           self.y = 4
③       self.game.canvas.move(self.image, self.x, self.y)
```

①では、fallingとbottom変数がTrueかどうかをチェックしています。どちらもTrueの場合は、スティックマンの下部と他のスプライトとの衝突が検出されなかったということです。

最後のチェック項目は、スティックマンの下端の座標がキャンバスの高さより小さいかどうかです。つまり、彼が地面（キャンバスの底辺）より上にいるかどうかです。スティックマンと衝突す

るものが何もなく、彼が地面より上にいるとしたら、彼は空中に浮いていることになります。その場合、スティックマンは床からはみ出ていることになるので、落下すべきです。床の端から落下させるためにyを4にします（②）。

③では、xとyにセットされている値に基づいて、画面内で画像を動かします。スティックマンの画像は左側と下部が衝突しているので、他のスプライトをループさせて衝突をチェックしたということは、xもyも0になっている可能性があります。その場合には、キャンバスのmove関数を呼んでも何も起きません。

あるいは、スティックマンが床の端から落ちている可能性があるかもしれません。その場合、yに4をセットしてスティックマンを落下させます。

ふぅ、なんて長い関数だったんでしょう！

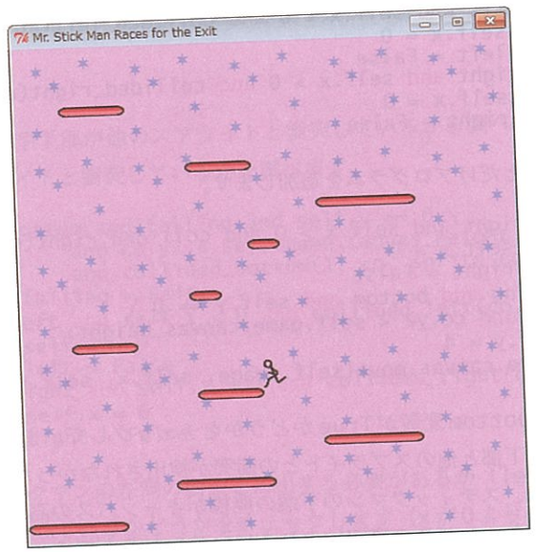
18.2 スティックマンを動かしてみる

mainloop関数を呼び出す前に、2行ほど追加してスティックマンを動かしてみましょう。

```
①sf = StickFigureSprite(g)
②g.sprites.append(sf)
  g.mainloop()
```

①では、StickFigureSpriteのオブジェクトを作って、変数sfにセットしています。作ったオブジェクトは、床のときと同じように、gameオブジェクトのスプライトのリストに追加します（②）。

さあ、プログラムを実行してみましょう。キーボードを操作すれば、スティックマンは、走ったり、床から床へジャンプしたり、床から落ちたりと動きはじめます。



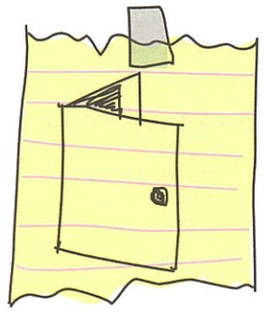
18.3 ドア！

これでゲームに足りないのは出口のドアのみになりました。ドアのスプライトを作り、ドアを見つけるプログラムを作り、最後にdoorオブジェクトをいままで作ってきたプログラムに追加すればゲームの完成です。

■ドアクラスを作る

ご想像のとおり、もう一つクラスを作る必要があります。DoorSpriteクラスです。以下がそのプログラムです。

```
  class DoorSprite(Sprite):
①     def __init__(self, game, photo_image, x, y, width, height):
②         Sprite.__init__(self, game)
③         self.photo_image = photo_image
④         self.image = game.canvas.create_image(x, y, \
                  image=self.photo_image, anchor='nw')
⑤         self.coordinates = Coords(x, y, x + (width / 2), y + height)
⑥         self.endgame = True
```



DoorSpriteクラスの__init__関数には、引数としてself、gameオブジェクト、photo_imageオブジェクト、xとy座標、ドア画像のwidth（幅）とheight（高さ）があります（①）。②では、親クラスであるSpriteクラスの__init__関数を呼んでいます。

③では、PlatformSpriteのときと同じように、photo_image引数を、同じ名前のオブジェクトの変数でセットしています。④では、キャンバスのcreate_image関数を使って、ドア画像をキャンバスに追加しています。そして、create_image関数の戻り値である画像の識別番号を、オブジェクトの変数imageにセットしています。

⑤では、DoorSpriteクラスの座標にxとyをセットしています（これらがドアのx1とy1座標となります）。次にx2とy2を計算します。x2座標は、ドア画像の幅の半分（width変数の値を2で割った値）をxに加えて求めます。たとえば、xを10（x1座標も10です）とし、幅を40とします。この場合、x2座標は30（10に40の半分を足した値）となります。

なぜこんなちょっと複雑なことをするのでしょうか？ それは、床の場合はスティックマンが床の端に衝突したらすぐに止まってほしいのですが、ドアの場合は彼をドアの正面で止まらせたいからです（ドアの隣で停止したら不自然です！）。ゲームをプレイしてスティックマンをドアへたどり着かせたら、この動きを見ることができます。

x2座標と違い、y2座標の計算は単純です。yにheight変数の値を足すだけです。本当にそれだけです。

最後に⑥で、endgameオブジェクトの変数にTrueをセットします。これは、スティックマンがドアにたどり着いたときにゲームを終了させるということです。

■ドアを見つける

ここで、StickFigureSpriteクラスにあるmove関数のプログラムを変更します。他のスプライトとスティックマンの左右との衝突を検出する部分です。

```
if left and self.x < 0 and collided_left(co, sprite_co):
    self.x = 0
    left = False
    if sprite.endgame:
        self.game.running = False
```

スティックマンが衝突したスプライトのendgame変数がTrueかどうかをチェックします。Trueの場合は、running変数にFalseをセットし、すべてを停止させます。つまりゲームをクリアしたということです。

他のスプライトとスティックマンの右側との衝突を検出する部分にも、同じようなプログラムを追加します。

```
if right and self.x > 0 and collided_right(co, sprite_co):
    self.x = 0
    right = False
    if sprite.endgame:
        self.game.running = False
```

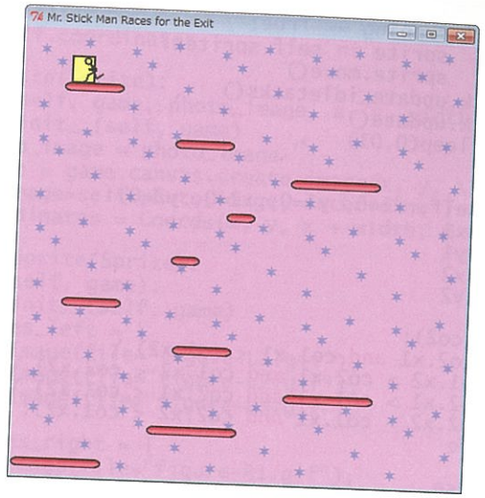
■ドアオブジェクトを追加する

追加する最後のプログラムは、ドアオブジェクトです。このプログラムはメインループの前に追加します。

```
g.sprites.append(platform7)
g.sprites.append(platform8)
g.sprites.append(platform9)
g.sprites.append(platform10)
door = DoorSprite(g, PhotoImage(file="door1.gif"), 45, 30, 40, 35)
g.sprites.append(door)
sf = StickFigureSprite(g)
g.sprites.append(sf)
g.mainloop()
```

gameオブジェクトの変数gと、PhotoImage（第15章で作ったドア画像）を使って、doorオブジェクトを作ります。キャンバス内で一番上にある床の上にドアを配置させるため、xとy引数に、それぞれ45と30をセットします。widthとheightには、それぞれ40と35をセットします。doorオブジェクトは、このゲームの他のすべてのスプライトと同じく、スプライトのリストに追加しておきます。

ここで追加したプログラムの動きは、スティックマンがドアにたどり着いたときに見ることができます。スティックマンはドアの隣ではなく、ドアの正面で停止するはずです。



18.4 ゲーム完成

このゲームのプログラムは全体で200行ちょっとあります。そのすべてを以下に掲載してあります。もし自分のプログラムが動かなかったら、関数（またはクラス）ごとに以下のプログラムと見比べて、どこが間違っているのかを調べてみましょう。

```
from tkinter import *
import random
import time

class Game:
    def __init__(self):
        self.tk = Tk()
        self.tk.title("Mr. Stick Man Races for the Exit")
        self.tk.resizable(0, 0)
        self.tk.wm_attributes("-topmost", 1)
        self.canvas = Canvas(self.tk, width=500, height=500, \
            highlightthickness=0)
        self.canvas.pack()
        self.tk.update()
        self.canvas_height = 500
        self.canvas_width = 500
        self.bg = PhotoImage(file="background.gif")
        w = self.bg.width()
        h = self.bg.height()
        for x in range(0, 5):
            for y in range(0, 5):
                self.canvas.create_image(x * w, y * h, \
                    image=self.bg, anchor='nw')
        self.sprites = []
        self.running = True

    def mainloop(self):
        while 1:
```

```
            if self.running == True:
                for sprite in self.sprites:
                    sprite.move()
            self.tk.update_idletasks()
            self.tk.update()
            time.sleep(0.01)

class Coords:
    def __init__(self, x1=0, y1=0, x2=0, y2=0):
        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2

def within_x(co1, co2):
    if (co1.x1 > co2.x1 and co1.x1 < co2.x2) \
            or (co1.x2 > co2.x1 and co1.x2 < co2.x2) \
            or (co2.x1 > co1.x1 and co2.x1 < co1.x2) \
            or (co2.x2 > co1.x1 and co2.x2 < co1.x2):
        return True
    else:
        return False

def within_y(co1, co2):
    if (co1.y1 > co2.y1 and co1.y1 < co2.y2) \
            or (co1.y2 > co2.y1 and co1.y2 < co2.y2) \
            or (co2.y1 > co1.y1 and co2.y1 < co1.y2) \
            or (co2.y2 > co1.y1 and co2.y2 < co1.y2):
        return True
    else:
        return False

def collided_left(co1, co2):
    if within_y(co1, co2):
        if co1.x1 <= co2.x2 and co1.x1 >= co2.x1:
            return True
    return False

def collided_right(co1, co2):
    if within_y(co1, co2):
        if co1.x2 >= co2.x1 and co1.x2 <= co2.x2:
            return True
    return False

def collided_top(co1, co2):
    if within_x(co1, co2):
        if co1.y1 <= co2.y2 and co1.y1 >= co2.y1:
            return True
    return False

def collided_bottom(y, co1, co2):
    if within_x(co1, co2):
        y_calc = co1.y2 + y
        if y_calc >= co2.y1 and y_calc <= co2.y2:
            return True
    return False

class Sprite:
    def __init__(self, game):
        self.game = game
        self.endgame = False
        self.coordinates = None
    def move(self):
        pass
```



```
    def coords(self):
        return self.coordinates

class PlatformSprite(Sprite):
    def __init__(self, game, photo_image, x, y, width, height):
        Sprite.__init__(self, game)
        self.photo_image = photo_image
        self.image = game.canvas.create_image(x, y, \
                image=self.photo_image, anchor='nw')
        self.coordinates = Coords(x, y, x + width, y + height)

class StickFigureSprite(Sprite):
    def __init__(self, game):
        Sprite.__init__(self, game)
        self.images_left = [
            PhotoImage(file="figure-L1.gif"),
            PhotoImage(file="figure-L2.gif"),
            PhotoImage(file="figure-L3.gif")
        ]
        self.images_right = [
            PhotoImage(file="figure-R1.gif"),
            PhotoImage(file="figure-R2.gif"),
            PhotoImage(file="figure-R3.gif")
        ]
        self.image = game.canvas.create_image(200, 470, \
                image=self.images_left[0], anchor='nw')
        self.x = -2
        self.y = 0
        self.current_image = 0
        self.current_image_add = 1
        self.jump_count = 0
        self.last_time = time.time()
        self.coordinates = Coords()
        game.canvas.bind_all('<KeyPress-Left>', self.turn_left)
        game.canvas.bind_all('<KeyPress-Right>', self.turn_right)
        game.canvas.bind_all('<space>', self.jump)

    def turn_left(self, evt):
        if self.y == 0:
            self.x = -2

    def turn_right(self, evt):
        if self.y == 0:
            self.x = 2

    def jump(self, evt):
        if self.y == 0:
            self.y = -4
            self.jump_count = 0

    def animate(self):
        if self.x != 0 and self.y == 0:
            if time.time() - self.last_time > 0.1:
                self.last_time= time.time()
                self.current_image += self.current_image_add
                if self.current_image >= 2:
                    self.current_image_add = -1
                if self.current_image <= 0:
                    self.current_image_add = 1
        if self.x < 0:
            if self.y != 0:
                self.game.canvas.itemconfig(self.image, \
```

III

```
                        image=self.images_left[2])
            else:
                self.game.canvas.itemconfig(self.image, \
                        image=self.images_left[self.current_image])
        elif self.x > 0:
            if self.y != 0:
                self.game.canvas.itemconfig(self.image, \
                        image=self.images_right[2])
            else:
                self.game.canvas.itemconfig(self.image, \
                        image=self.images_right[self.current_image])

    def coords(self):
        xy = self.game.canvas.coords(self.image)
        self.coordinates.x1 = xy[0]
        self.coordinates.y1 = xy[1]
        self.coordinates.x2 = xy[0] + 27
        self.coordinates.y2 = xy[1] + 30
        return self.coordinates

    def move(self):
        self.animate()
        if self.y < 0:
            self.jump_count += 1
            if self.jump_count > 20:
                self.y = 4
        if self.y > 0:
            self.jump_count -= 1
        co = self.coords()
        left = True
        right = True
        top = True
        bottom = True
        falling = True
        if self.y > 0 and co.y2 >= self.game.canvas_height:
            self.y = 0
            bottom = False
        elif self.y < 0 and co.y1 <= 0:
            self.y = 0
            top = False
        if self.x > 0 and co.x2 >= self.game.canvas_width:
            self.x = 0
            right = False
        elif self.x < 0 and co.x1 <= 0:
            self.x = 0
            left = False
        for sprite in self.game.sprites:
            if sprite == self:
                continue
            sprite_co = sprite.coords()
            if top and self.y < 0 and collided_top(co, sprite_co):
                self.y = -self.y
                top = False
            if bottom and self.y > 0 and collided_bottom(self.y, \
                    co, sprite_co):
                self.y = sprite_co.y1 - co.y2
                if self.y < 0:
                    self.y = 0
                bottom = False
                top = False
            if bottom and falling and self.y == 0 \
                    and co.y2 < self.game.canvas_height \
                    and collided_bottom(1, co, sprite_co):
```

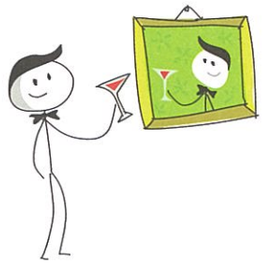


```
                falling = False
            if left and self.x < 0 and collided_left(co, sprite_co):
                self.x = 0
                left = False
                if sprite.endgame:
                    self.game.running = False
            if right and self.x > 0 and collided_right(co, sprite_co):
                self.x = 0
                right = False
                if sprite.endgame:
                    self.game.running = False
        if falling and bottom and self.y == 0 \
                and co.y2 < self.game.canvas_height:
            self.y = 4
        self.game.canvas.move(self.image, self.x, self.y)

class DoorSprite(Sprite):
    def __init__(self, game, photo_image, x, y, width, height):
        Sprite.__init__(self, game)
        self.photo_image = photo_image
        self.image = game.canvas.create_image(x, y, \
                image=self.photo_image, anchor='nw')
        self.coordinates = Coords(x, y, x + (width / 2), y + height)
        self.endgame = True

g = Game()
platform1 = PlatformSprite(g, PhotoImage(file="platform1.gif"), \
    0, 480, 100, 10)
platform2 = PlatformSprite(g, PhotoImage(file="platform1.gif"), \
    150, 440, 100, 10)
platform3 = PlatformSprite(g, PhotoImage(file="platform1.gif"), \
    300, 400, 100, 10)
platform4 = PlatformSprite(g, PhotoImage(file="platform1.gif"), \
    300, 160, 100, 10)
platform5 = PlatformSprite(g, PhotoImage(file="platform2.gif"), \
    175, 350, 66, 10)
platform6 = PlatformSprite(g, PhotoImage(file="platform2.gif"), \
    50, 300, 66, 10)
platform7 = PlatformSprite(g, PhotoImage(file="platform2.gif"), \
    170, 120, 66, 10)
platform8 = PlatformSprite(g, PhotoImage(file="platform2.gif"), \
    45, 60, 66, 10)
platform9 = PlatformSprite(g, PhotoImage(file="platform3.gif"), \
    170, 250, 32, 10)
platform10 = PlatformSprite(g, PhotoImage(file="platform3.gif"), \
    230, 200, 32, 10)
g.sprites.append(platform1)
g.sprites.append(platform2)
g.sprites.append(platform3)
g.sprites.append(platform4)
g.sprites.append(platform5)
g.sprites.append(platform6)
g.sprites.append(platform7)
g.sprites.append(platform8)
g.sprites.append(platform9)
g.sprites.append(platform10)
door = DoorSprite(g, PhotoImage(file="door1.gif"), 45, 30, 40, 35)
g.sprites.append(door)
sf = StickFigureSprite(g)
g.sprites.append(sf)
g.mainloop()
```

18.5 この章でわかったこと



この章で、ミスター・スティックマン脱出ゲームが完成しました。スティックマンのアニメーション用クラスを作り、彼をキャンバス上で動かすための関数を書き、彼が走っているようなアニメーションが表示されるようにしました（スティックマン画像を次々に変化させ、彼が走っているように見せました）。基本的な衝突チェックで、キャンバスの左端や右端との衝突を検出したり、床やドアなど他のスプライトとの衝突を検出しました。また、キャンバスの上端や底辺との衝突を検出するプログラムも追加しました。彼が床の端からはみ出た場合は、床から落下するようにしました。ゲームを終了させるため、ミスター・スティックマンがドアにたどり着いたことを検出するプログラムを追加しました。

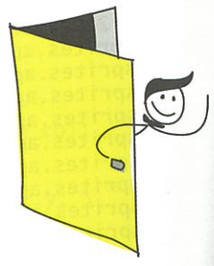
18.6 自分でやってみよう

このゲームをより良くするためにできることはまだまだたくさんあります。いまのところ、このゲームはとてもシンプルです。もっとプロっぽく見せたり、もっと楽しいゲームにするプログラムを追加できます。以下の三つの機能を追加してみましょう。

■①“You Win!”

第14章で作ったBounce!ゲームの“Game Over”テキストのように、スティックマンがドアにたどり着いたとき“You Win!”というテキストを表示してみましょう。ゲームをクリアしたことがわかりやすくなります。

■②ドアをアニメーションさせよう



第15章でドアの画像を二つ作りました。一つはドアが開いている画像で、もう一つは閉じている画像でした。ミスター・スティックマンがドアにたどり着いたとき、ドアの画像を開いたドアの画像に変更してスティックマンを見えなくし、ドアの画像を元の閉じたドアに戻してみましょう。スティックマンが出口から脱出し、ドアが閉じたように見えるはずです。`DoorSprite`クラスと`StickFigureSprite`クラスを変更すると、このアニメーションが作れます。

■③プラットフォームを動かそう

新しい床クラス`MovingPlatformSprite`を追加して、この床をキャンバスの端から端へと動くようにします。ゲームをもっと難しくさせましょう。

これからどうしよう？

ここまで、基本的なプログラミングのやり方を、Pythonを使いながら学んできました。いったんプログラミングのコツみたいなものを身につけてしまえば、Python以外のプログラミング言語も、比較的簡単に使えるようになるはずです。Pythonは非常に便利なプログラミング言語ですが、あらゆる場合において、一番いい選択肢というわけではないのです。プログラミング言語には、それぞれ得意分野があるので、怖がらないでいろいろなプログラミング言語を試してみるのも、いい経験になるでしょう。ここでは、ゲームやグラフィックスのプログラムを作るための他の方法や、世の中でよく使われているプログラミング言語の一部を、簡単にではありますが紹介してみましょう。

■ゲームとグラフィックスのプログラミング

ゲームやグラフィックスのプログラムをもっと作りたいなら、いろいろなやり方がありますよ。ここでは、そのほんの一部を紹介します。

- BlitzBasic（`http://www.blitzbasic.com/`）：ゲームのために特別に設計されたBASICプログラミング言語を使っています。
- Adobe Flash：ブラウザで動作するように設計されているアニメーションソフトの一種で、ActionScriptと呼ばれる独自のプログラミング言語を使っています（`http://www.adobe.com/devnet/actionscript.html`）。
- Alice（`http://www.alice.org/`）：3Dのプログラミング環境（Microsoft WindowsおよびMac OS X用のみ）。
- Scratch（`http://scratch.mit.edu/`）：ゲームを作るためのツール。
- Unity3D（`http://unity3d.com/`）：ゲームを作るためのツール。

どれを使ってプログラミングをはじめるにしても、インターネットで検索すれば、インストールの方法や使い方を説明しているページが見つかるでしょう（もしかすると英語のページしかないかもしれませんが）。Pythonがとても気に入って、Pythonで開発し続けたいと思ってくれたのなら、PyGameというゲームプログラミングのために設計されたPythonのモジュールがあります。

■PyGame

Python 3で動くPyGameのバージョンは、pygame2です（pgreloadedともいいます）。pygame2は、高度なゲームやマルチメディアのアプリケーションをPythonで開発するときに使える、素晴らしいモジュールの集まりで、いろいろなオペレーティングシステムをサポートしています。