Pyxel 4 Kids Book

からあげ　著

2025-05-05 版　　なし 発行

はじめに

この本は、小学校 4 年生のみなさんがプログラミングでゲームを作れるようになるための入門書です。

こんにちは！ この本を手に取ってくれてありがとう。

みなさんは、ゲームで遊ぶのが好きですか？「マリオ」や「マインクラフト」のようなゲームを、自分で作ってみたいと思ったことはありませんか？

この本では、「Pyxel（ピクセル）」というプログラムを使って、自分だけのゲームを作る方法を学びます。Pyxel は、昔ながらのレトロゲーム風の 2D ゲームを簡単に作ることができるツールです。

プログラミングは、最初は難しく感じるかもしれません。でも、一歩一歩進んでいけば、必ず自分の思い通りのゲームを作れるようになります。失敗してもあきらめずに、何度もチャレンジしてみてください。

さあ、いっしょにゲーム作りの冒険に出かけましょう！

この本の使い方

この本は、順番に読み進めていくことをおすすめします。最初の章では、プログラミングや Pyxel についての基本的な知識を学びます。その後、実際にプログラムを書いて、少しずつゲームを作っていきます。

各章の終わりには「チャレンジ問題」があります。学んだことを使って、自分でプログラムを変えてみましょう。うまくいかなくても大丈夫。何度も試してみることが大切です。

プログラミングは、実際に手を動かして書いてみることが一番の上達方法です。この本に書かれているプログラムは、すべて自分で入力してみてください。

準備するもの

この本を使うためには、以下のものが必要です。

- パソコン（Windows か Mac）
- インターネット接続
- やる気とチャレンジ精神！

免責事項

本書に記載された内容は、情報の提供のみを目的としています。したがって、本書を用いた開発、製作、運用は、必ずご自身の責任と判断によって行ってください。これらの情報による開発、製作、運用の結果について、著者はいかなる責任も負いません。

3

目次

第 1 章

Python ってなに？

この章では、プログラミングと Python（パイソン）について学びます。

1.1　プログラミングとは

みなさんは、「プログラミング」という言葉を聞いたことがありますか？

プログラミングとは、コンピュータに「こうしてほしい」という命令を出すことです。コンピュータは、人間の言葉をそのまま理解することができません。だから、コンピュータが理解できる特別な言葉（プログラミング言語）を使って、命令を伝える必要があります。

たとえば、友だちに「3 歩前に進んで、右に曲がって、2 歩進んで」と言うように、コンピュータにも「これをして、次にこれをして」と順番に指示を出します。

--[[path = programming_image (not exist)]]--

▲図 1.1　プログラミングのイメージ

プログラミングをすると、次のようなことができるようになります。

- ゲームを作る
- ウェブサイトを作る
- ロボットを動かす
- 計算を自動でする
- 絵や音楽を作る

1.2 Python とは

Python（パイソン）は、世界中で人気のあるプログラミング言語です。名前は、ヘビの「ニシキヘビ（Python）」からきています。

--[[path = python_logo (not exist)]]--

▲図 1.2　Python のロゴ

Python は、次のような特徴があります。

- 書き方がシンプルで、読みやすい
- 初心者でも始めやすい
- いろいろなことができる万能な言語
- 世界中の多くの人が使っている

Python は、Google やディズニー、NASA など、有名な会社や組織でも使われています。

1.3 Python でできること

Python を使うと、いろいろなことができます。

- ゲーム開発
- ウェブサイト作成
- 人工知能（AI）の開発
- データの分析
- 科学の研究
- ロボットの制御

この本では、Python を使ってゲームを作る方法を学びます。

1.4 プログラムの例

Python のプログラムがどんな感じか、簡単な例を見てみましょう。

▼リスト 1.1　簡単な Python プログラム

```
# これは「こんにちは」と表示するプログラムです
print("こんにちは、世界！ ")
print("私の名前は太郎です")
print("Pythonでプログラミングを学んでいます")
```

このプログラムは、3 行のメッセージを画面に表示します。

- 1 行目は「#」で始まっています。これは「コメント」といって、プログラムの説明を書くための行です。コンピュータはこの行を無視します。
- 2〜4 行目の「print」は、「画面に表示せよ」という命令です。かっこ「()」の中に書いた文字が表示されます。

1.5　まとめ

この章では、プログラミングと Python について学びました。

- プログラミングは、コンピュータに命令を出すこと
- Python は、初心者にもやさしいプログラミング言語
- Python では、ゲームやウェブサイトなど、いろいろなものが作れる

次の章では、ゲーム開発に使う「Pyxel」というツールについて学びます。

1.6　チャレンジ問題

1. 自分の名前や好きなものを表示するプログラムを考えてみよう。どんな「print」の命令を書けばいいかな？
2. プログラミングで作ってみたいものは何ですか？ 友だちや家族に話してみよう。

第 2 章

Pyxel ってなに？

この章では、ゲーム開発ツール「Pyxel（ピクセル）」について学びます。

2.1　ゲームエンジンとは

ゲームを作るときには、画面に絵を表示したり、音を鳴らしたり、キーボードやマウスの操作を受け付けたりする必要があります。これらの機能をゼロから作るのは、とても大変な作業です。

そこで役立つのが「ゲームエンジン」です。ゲームエンジンは、ゲーム開発に必要な基本的な機能をまとめたツールで、これを使うことで簡単にゲームを作ることができます。

▲図 2.1　ゲームエンジンのイメージ

2.2　Pyxel とは

Pyxel（ピクセル）は、Python 用のレトロゲームエンジンです。「レトロ」とは、昔ながらの古いスタイルという意味です。1980 年代のファミコンやゲームボーイのような、ドット絵（小さな四角い点の集まりで作る絵）を使ったゲームを簡単に作ることができます。

▲図 2.2　Pyxel のロゴ

Pyxel は日本人の方が作ったツールで、世界中で使われています。公式サイトは https://github.com/kitao/pyxel です。

2.3 Pyxel の特徴

Pyxel には、次のような特徴があります。

- シンプルなプログラムでゲームが作れる
- レトロな見た目のゲームが作れる
- 画像、音、音楽を簡単に扱える
- Windows、Mac、Linux で動作する
- 日本語のドキュメントがある

2.4 Pyxel でできるゲーム

Pyxel を使うと、次のようなゲームを作ることができます。

- アクションゲーム
- シューティングゲーム
- パズルゲーム
- RPG（ロールプレイングゲーム）

--[[path = pyxel_games (not exist)]]--

▲図 2.3　Pyxel で作れるゲームの例

2.5 Pyxel の画面

Pyxel では、画面の大きさや色の数が制限されています。これは、昔のゲーム機と同じような制約を再現しているからです。

- 画面の大きさ：最大 256 × 256 ドット
- 使える色：16 色まで

この制限があることで、かえって作りやすくなる面もあります。選択肢が少ないので、迷わずに済むからです。

2.6 Pyxel のサンプルコード

Pyxel を使ったプログラムがどんな感じか、簡単な例を見てみましょう。

▼リスト 2.1　簡単な Pyxel プログラム

```
import pyxel

# Pyxelを初期化
pyxel.init(160, 120, title="はじめてのPyxel")

# 画面を更新する関数
def update():
    # ESCキーが押されたら終了
    if pyxel.btnp(pyxel.KEY_ESCAPE):
        pyxel.quit()

# 画面を描画する関数
def draw():
    # 画面を消去
    pyxel.cls(0)
    # 文字を表示
    pyxel.text(55, 55, "Hello, Pyxel!", 7)

# Pyxelの実行
pyxel.run(update, draw)
```

このプログラムは、黒い画面に「Hello, Pyxel!」という文字を表示します。

- ‘import pyxel‘：Pyxel を使うための準備
- ‘pyxel.init(160, 120)‘：160 × 120 ドットの画面を作る
- ‘update‘関数：ゲームの状態を更新する（キー入力など）
- ‘draw‘関数：画面に絵を描く
- ‘pyxel.run‘：ゲームを実行する

2.7 まとめ

この章では、Pyxel について学びました。

- Pyxel は、レトロゲームを作るためのツール
- シンプルな命令でゲームが作れる
- 画面サイズや色数に制限がある
- 様々な種類のゲームが作れる

次の章では、Pyxel と Python をパソコンにインストールする方法を学びます。

2.8 チャレンジ問題

1. 昔のゲーム機（ファミコン、ゲームボーイなど）で遊んだことがありますか？ どんなゲームが好きでしたか？
2. Pyxel で作ってみたいゲームを考えてみよう。どんなゲームを作りたいですか？

第 3 章

開発環境をセットアップしよう

この章では、Python と Pyxel をパソコンにインストールする方法を学びます。

3.1　準備するもの

プログラミングを始める前に、次のものを準備しましょう。

- パソコン（Windows か Mac）
- インターネット接続
- 大人の人の手伝い（必要に応じて）

インストール作業は少し難しいかもしれません。分からないことがあれば、お家の人や先生に手伝ってもらいましょう。

3.2　Python のインストール

まずは、Python をインストールします。Python は無料で使えるプログラミング言語です。

3.2.1　Windows の場合

--[[path = python_windows (not exist)]]--

▲図 3.1　Windows でのインストール画面

1. インターネットブラウザで「Python ダウンロード」と検索するか、https://www.python.org/downloads/ にアクセスします。
2.「Download Python 3.x.x」（x は数字）というボタンをクリックします。
3. ダウンロードしたファイル（python-3.x.x-amd64.exe）をダブルクリックします。
4.「Add Python 3.x to PATH」にチェックを入れます。これは重要です！
5.「Install Now」をクリックします。
6. インストールが完了したら「Close」をクリックします。

3.2.2 Mac の場合

--[[path = python_mac (not exist)]]--

▲図 3.2 Mac でのインストール画面

1. インターネットブラウザで「Python ダウンロード」と検索するか、https://www.python.org/downloads/ にアクセスします。
2.「Download Python 3.x.x」（x は数字）というボタンをクリックします。
3. ダウンロードしたファイル（python-3.x.x-macos11.pkg）をダブルクリックします。
4. 画面の指示に従ってインストールします。
5. インストールが完了したら「閉じる」をクリックします。

3.3 Python のインストール確認

Python がきちんとインストールされたか確認しましょう。

3.3.1 Windows の場合

1. スタートメニューから「コマンドプロンプト」を探して開きます。
2. 開いたウィンドウに「python --version」と入力して、Enter キーを押します。
3.「Python 3.x.x」（x は数字）と表示されれば成功です。

--[[path = python_check_windows (not exist)]]--

▲図 3.3 Windows でのインストール確認

3.3.2 Mac の場合

1. Finder から「アプリケーション」→「ユーティリティ」→「ターミナル」を開きます。
2. 開いたウィンドウに「python3 --version」と入力して、Enter キーを押します。
3.「Python 3.x.x」（x は数字）と表示されれば成功です。

--[[path = python_check_mac (not exist)]]--

▲図 3.4　Mac でのインストール確認

3.4 Pyxel のインストール

次に、Pyxel をインストールします。Pyxel は、Python のコマンドを使ってインストールします。

3.4.1 Windows の場合

1. コマンドプロンプトを開きます（まだ開いていない場合）。
2. 次のコマンドを入力して、Enter キーを押します。

```
pip install -U pyxel
```

3.4.2 Mac の場合

1. ターミナルを開きます（まだ開いていない場合）。
2. 次のコマンドを入力して、Enter キーを押します。

```
pip3 install -U pyxel
```

‘error: externally-managed-environment‘というエラーが出たら以下をためしてください。

```
pip3 install -U pyxel --break-system-packages
```

3.5 Pyxel のインストール確認

Pyxel がきちんとインストールされたか確認しましょう。

3.5.1 Windows の場合

1. コマンドプロンプトで次のコマンドを入力して、Enter キーを押します。

```
python -m pyxel.examples.01_hello_pyxel
```


3.5.2 Mac の場合

1. ターミナルで次のコマンドを入力して、Enter キーを押します。

```
python3 -m pyxel.examples.01_hello_pyxel
```

どちらの場合も、黒い画面に「Hello, Pyxel!」という文字が表示されれば成功です。ESC キーを押すと、画面が閉じます。

--[[path = pyxel_check (not exist)]]--

▲図 3.5　Pyxel のインストール確認

3.6 プログラムを書くためのエディタ

プログラムを書くには、「エディタ」というソフトウェアを使います。メモ帳でも書けますが、専用のエディタを使うと便利です。ここでは、「Visual Studio Code（VSCode）」というエディタをおすすめします。

3.6.1 Visual Studio Code のインストール

1. インターネットブラウザで「Visual Studio Code ダウンロード」と検索するか、https://code.visualstudio.com/ にアクセスします。
2. 「Download for Windows」または「Download for Mac」ボタンをクリックします。
3. ダウンロードしたファイルをダブルクリックして、画面の指示に従ってインストールします。

--[[path = vscode (not exist)]]--

▲図 3.6　Visual Studio Code の画面

3.7 まとめ

この章では、Python と Pyxel をインストールする方法を学びました。

- Python は公式サイトからダウンロードしてインストール
- Pyxel はコマンドでインストール
- プログラムを書くためのエディタとして Visual Studio Code がおすすめ

次の章では、最初の Pyxel プログラムを書いてみましょう。

3.8 チャレンジ問題

1. Pyxel のサンプルプログラムを他にも実行してみよう。次のコマンドを試してみてください。

```
# Windowsの場合
python -m pyxel.examples.02_jump_game
python -m pyxel.examples.03_draw_api

# Macの場合
python3 -m pyxel.examples.02_jump_game
python3 -m pyxel.examples.03_draw_api
```

2. Visual Studio Code を起動して、新しいファイルを作ってみよう。「ファイル」→「新規ファイル」を選んで、何か文字を入力してみましょう。

第 4 章

最初のプログラムを作ろう

この章では、Pyxel を使った最初のプログラムを作成し、画面に文字や図形を表示します。

4.1 プログラムの作成手順

Pyxel でプログラムを作るときは、次の手順で進めます。

1. エディタ（Visual Studio Code）を起動する
2. 新しいファイルを作成する
3. プログラムを書く
4. ファイルを保存する
5. プログラムを実行する

それでは、順番に進めていきましょう。

4.2 エディタの起動とファイル作成

まずは、Visual Studio Code（VSCode）を起動します。

1. スタートメニュー（Windows の場合）またはアプリケーションフォルダ（Mac の場合）から「Visual Studio Code」を探して起動します。
2.「ファイル」→「新規ファイル」を選びます。
3.「ファイル」→「保存」を選び、ファイル名を「hello_pyxel.py」として保存します。ファイル名の最後に「.py」をつけることで、Python のファイルであることを示します。

--[[path = vscode_new_file (not exist)]]--

▲図 4.1　新しいファイルの作成

4.3　最初のプログラム

それでは、最初のプログラムを書いてみましょう。次のコードを入力してください。

▼リスト 4.1　最初の Pyxel プログラム

```
import pyxel

# Pyxelを初期化（画面の大きさを160x120ピクセルに設定）
pyxel.init(160, 120, title="はじめてのPyxel")

# 画面を更新する関数
def update():
    # ESCキーが押されたら終了
    if pyxel.btnp(pyxel.KEY_ESCAPE):
        pyxel.quit()

# 画面を描画する関数
def draw():
    # 画面を黒色（0）でクリア
    pyxel.cls(0)
    # 白色（7）で文字を表示
    pyxel.text(55, 41, "Hello, Pyxel!", 7)
    pyxel.text(31, 61, "はじめてのプログラミング", 10)
    pyxel.text(50, 81, "画面サイズ: 160x120", 11)

# Pyxelの実行（updateとdraw関数を渡す）
pyxel.run(update, draw)
```


4.4　プログラムの保存と実行

プログラムを入力したら、保存して実行しましょう。

1. 「ファイル」→「保存」を選ぶか、Ctrl+S（Windows の場合）または Command+S（Mac の場合）を押して保存します。
2. コマンドプロンプト（Windows の場合）またはターミナル（Mac の場合）を開きます。
3. プログラムを保存したフォルダに移動します。
4. 次のコマンドを入力して実行します。

```
# Windowsの場合
python hello_pyxel.py

# Macの場合
python3 hello_pyxel.py
```

プログラムが正しく実行されると、黒い画面に「Hello, Pyxel!」などの文字が表示されます。ESC キーを押すと、プログラムが終了します。

--[[path = first_program_result (not exist)]]--

▲図 4.2　最初のプログラムの実行結果

4.5　プログラムの解説

最初のプログラムを詳しく見ていきましょう。

4.5.1　Pyxel の読み込みと初期化

▼リスト 4.2　Pyxel の読み込みと初期化

```
import pyxel

# Pyxelを初期化（画面の大きさを160x120ピクセルに設定）
pyxel.init(160, 120, title="はじめてのPyxel")
```

- ‘import pyxel‘：Pyxel を使うための準備をします。
- ‘pyxel.init(160, 120, title="はじめての Pyxel")‘：Pyxel を初期化します。
- 最初の数字（160）は画面の幅（横幅）をピクセル単位で指定します。
- 2 番目の数字（120）は画面の高さをピクセル単位で指定します。
- ‘title="はじめての Pyxel"‘は、ウィンドウのタイトルを設定します。

4.5.2　update 関数

▼リスト 4.3　update 関数

```
# 画面を更新する関数
def update():
    # ESCキーが押されたら終了
    if pyxel.btnp(pyxel.KEY_ESCAPE):
        pyxel.quit()
```

- ‘def update():‘：「update」という名前の関数を定義します。この関数は、ゲームの状態を更新するために使われます。
- ‘if pyxel.btnp(pyxel.KEY_ESCAPE):‘：ESC キーが押されたかどうかをチェックします。
- ‘pyxel.quit()‘：Pyxel を終了します。

4.5.3 draw 関数

▼リスト 4.4 draw 関数

```
# 画面を描画する関数
def draw():
    # 画面を黒色（0）でクリア
    pyxel.cls(0)
    # 白色（7）で文字を表示
    pyxel.text(55, 41, "Hello, Pyxel!", 7)
    pyxel.text(31, 61, "はじめてのプログラミング", 10)
    pyxel.text(50, 81, "画面サイズ: 160x120", 11)
```

- ‘def draw():‘：「draw」という名前の関数を定義します。この関数は、画面に絵を描くために使われます。
- ‘pyxel.cls(0)‘：画面を指定した色（ここでは 0=黒）でクリアします。
- ‘pyxel.text(55, 41, "Hello, Pyxel!", 7)‘：文字を表示します。
- 最初の数字（55）は X 座標（左からの位置）を指定します。
- 2 番目の数字（41）は Y 座標（上からの位置）を指定します。
- ‘"Hello, Pyxel!"‘は表示するテキストです。
- 最後の数字（7）は文字の色を指定します（7=白）。

4.5.4 Pyxel の実行

▼リスト 4.5　Pyxel の実行

```
# Pyxelの実行（updateとdraw関数を渡す）
pyxel.run(update, draw)
```

- ‘pyxel.run(update, draw)‘：Pyxel を実行します。update と draw 関数を渡して、ゲームループを開始します。

4.6　Pyxel の色

Pyxel では、0 から 15 までの 16 色を使うことができます。それぞれの色には番号が割り当てられています。

▼表 4.1　Pyxel の色

番号	色
0	黒
1	紺色
2	紫色
3	緑色
4	茶色
5	暗い青色
6	水色
7	白
8	赤色
9	オレンジ
10	黄色
11	黄緑色
12	青色
13	ピンク
14	灰色
15	明るい灰色

色の番号を変えると、文字や図形の色を変えることができます。

4.7　座標系

Pyxel の画面では、左上が原点（0, 0）で、右に行くほど X 座標が大きくなり、下に行くほど Y 座標が大きくなります。

--[[path = coordinate_system (not exist)]]--

▲図 4.3　Pyxel の座標系

たとえば、160 × 120 の画面の場合：* 左上の座標は（0, 0）* 右上の座標は（159, 0）* 左下の座標は（0, 119）* 右下の座標は（159, 119）

4.8　プログラムの変更

プログラムを少し変更して、違う結果を見てみましょう。

4.8.1　画面サイズの変更

画面のサイズを変更するには、‘pyxel.init()‘の引数を変更します。

▼リスト 4.6　画面サイズの変更

```
# 画面サイズを200x150に変更
pyxel.init(200, 150, title="はじめてのPyxel")
```

4.8.2　文字の位置と色の変更

文字の位置や色を変更するには、‘pyxel.text()‘の引数を変更します。

▼リスト 4.7　文字の位置と色の変更

```
# 文字の位置と色を変更
pyxel.text(10, 10, "Hello, Pyxel!", 8)  # 赤色（8）で左上に表示
pyxel.text(50, 50, "こんにちは！ ", 10)  # 黄色（10）で中央に表示
```

4.9 まとめ

この章では、Pyxel を使った最初のプログラムを作成しました。

- Pyxel プログラムの基本的な構造
- 画面の初期化と設定
- 文字の表示方法
- 色の指定方法
- 座標系の仕組み

次の章では、さまざまな図形を描く方法を学びます。

4.10 チャレンジ問題

1. 画面サイズを変更して（例：240x180）、プログラムを実行してみよう。文字の位置も調整してみよう。

2. 自分の名前や好きな言葉を画面に表示するプログラムを作ってみよう。色や位置を工夫してみよう。

3. 背景色を変えてみよう。‘pyxel.cls(0)‘の 0 を別の数字（1〜15）に変えるとどうなるかな？

第 5 章

図形を描いてみよう

この章では、Pyxel を使ってさまざまな図形を描く方法を学びます。

5.1　図形描画の基本

Pyxel では、点、線、四角形、円などのさまざまな図形を簡単に描くことができます。これらの図形を組み合わせることで、キャラクターや背景などを作ることができます。

図形を描くコマンドは、すべて ‘draw‘関数の中に書きます。

5.2　基本的なプログラム構造

まずは、図形を描くための基本的なプログラム構造を確認しましょう。

▼リスト 5.1　基本的なプログラム構造

```
import pyxel

# Pyxelを初期化
pyxel.init(160, 120, title="図形を描こう")

# 画面を更新する関数
def update():
    # ESCキーが押されたら終了
    if pyxel.btnp(pyxel.KEY_ESCAPE):
        pyxel.quit()

# 画面を描画する関数
def draw():
    # 画面を黒色（0）でクリア
    pyxel.cls(0)
```

```
    # ここに図形を描くコードを書きます

# Pyxelの実行
pyxel.run(update, draw)
```

このプログラムをファイル名「drawing.py」として保存し、これから学ぶ図形描画のコードを ‘draw‘関数の中に追加していきます。

5.3 点を描く

最も基本的な図形は「点」です。点を描くには、‘pset‘関数を使います。

▼リスト 5.2 点を描く

```
# 点を描く
pyxel.pset(80, 60, 7)  # 座標(80, 60)に白色(7)の点を描く
pyxel.pset(85, 65, 8)  # 座標(85, 65)に赤色(8)の点を描く
pyxel.pset(75, 65, 12) # 座標(75, 65)に青色(12)の点を描く
```

--[[path = draw_point_image (not exist)]]--

▲図 5.1 点を描いた例

5.4 線を描く

2 点を結ぶ線を描くには、‘line‘関数を使います。

▼リスト 5.3 線を描く

```
# 線を描く
pyxel.line(30, 30, 130, 90, 7)   # 座標(30, 30)から(130, 90)まで白色(7)の線を描く
pyxel.line(30, 90, 130, 30, 8)   # 座標(30, 90)から(130, 30)まで赤色(8)の線を描く
pyxel.line(80, 20, 80, 100, 11)  # 座標(80, 20)から(80, 100)まで黄緑色(11)の線を描
く
```

--[[path = draw_line_image (not exist)]]--

▲図 5.2　線を描いた例

5.5　四角形を描く

四角形を描くには、‘rect‘関数を使います。

▼リスト 5.4　四角形を描く

```
# 四角形を描く
pyxel.rect(10, 10, 40, 30, 7)    # 座標(10, 10)から幅40、高さ30の白色(7)の四角形を描
く
pyxel.rect(60, 40, 50, 40, 8)    # 座標(60, 40)から幅50、高さ40の赤色(8)の四角形を描
く
pyxel.rect(30, 70, 70, 20, 12)   # 座標(30, 70)から幅70、高さ20の青色(12)の四角形を
描く
```


--[[path = draw_rect_image (not exist)]]--

▲図 5.3　四角形を描いた例

‘rect‘関数の引数は次のとおりです。

- 第 1 引数：左上の X 座標
- 第 2 引数：左上の Y 座標
- 第 3 引数：幅（横幅）
- 第 4 引数：高さ
- 第 5 引数：色

5.6　枠だけの四角形を描く

枠だけの四角形（中が塗りつぶされていない四角形）を描くには、‘rectb‘関数を使います。

▼リスト 5.5　枠だけの四角形を描く

```
# 枠だけの四角形を描く
pyxel.rectb(10, 10, 40, 30, 7)   # 座標(10, 10)から幅40、高さ30の白色(7)の枠を描く
pyxel.rectb(60, 40, 50, 40, 8)   # 座標(60, 40)から幅50、高さ40の赤色(8)の枠を描く
pyxel.rectb(30, 70, 70, 20, 12)  # 座標(30, 70)から幅70、高さ20の青色(12)の枠を描く
```

--[[path = draw_rectb_image (not exist)]]--

▲図 5.4　枠だけの四角形を描いた例

5.7　円を描く

円を描くには、‘circ‘関数を使います。

▼リスト 5.6　円を描く

```
# 円を描く
pyxel.circ(40, 30, 10, 7)    # 座標(40, 30)を中心に半径10の白色(7)の円を描く
pyxel.circ(80, 60, 20, 8)    # 座標(80, 60)を中心に半径20の赤色(8)の円を描く
pyxel.circ(120, 90, 15, 12)  # 座標(120, 90)を中心に半径15の青色(12)の円を描く
```


--[[path = draw_circ_image (not exist)]]--

▲図 5.5　円を描いた例

‘circ‘関数の引数は次のとおりです。

- 第 1 引数：中心の X 座標
- 第 2 引数：中心の Y 座標
- 第 3 引数：半径
- 第 4 引数：色

5.8　枠だけの円を描く

枠だけの円（中が塗りつぶされていない円）を描くには、‘circb‘関数を使います。

▼リスト 5.7　枠だけの円を描く

```
# 枠だけの円を描く
pyxel.circb(40, 30, 10, 7)   # 座標(40, 30)を中心に半径10の白色(7)の円の枠を描く
pyxel.circb(80, 60, 20, 8)   # 座標(80, 60)を中心に半径20の赤色(8)の円の枠を描く
pyxel.circb(120, 90, 15, 12) # 座標(120, 90)を中心に半径15の青色(12)の円の枠を描く
```

--[[path = draw_circb_image (not exist)]]--

▲図 5.6　枠だけの円を描いた例

5.9　三角形を描く

三角形を描くには、‘tri‘関数を使います。

▼リスト 5.8　三角形を描く

```
# 三角形を描く
pyxel.tri(80, 20, 50, 80, 110, 80, 7)   # 白色(7)の三角形を描く
pyxel.tri(80, 30, 60, 70, 100, 70, 8)   # 赤色(8)の三角形を描く
```

--[[path = draw_tri_image (not exist)]]--

▲図 5.7　三角形を描いた例

‘tri‘関数の引数は次のとおりです。

- 第 1 引数：1 つ目の頂点の X 座標
- 第 2 引数：1 つ目の頂点の Y 座標
- 第 3 引数：2 つ目の頂点の X 座標
- 第 4 引数：2 つ目の頂点の Y 座標
- 第 5 引数：3 つ目の頂点の X 座標
- 第 6 引数：3 つ目の頂点の Y 座標
- 第 7 引数：色

5.10　枠だけの三角形を描く

枠だけの三角形（中が塗りつぶされていない三角形）を描くには、‘trib‘関数を使います。

▼リスト 5.9　枠だけの三角形を描く

```
# 枠だけの三角形を描く
pyxel.trib(80, 20, 50, 80, 110, 80, 7)  # 白色(7)の三角形の枠を描く
pyxel.trib(80, 30, 60, 70, 100, 70, 8)  # 赤色(8)の三角形の枠を描く
```

--[[path = draw_trib_image (not exist)]]--

▲図 5.8　枠だけの三角形を描いた例

5.11　図形を組み合わせる

これまで学んだ図形を組み合わせて、簡単な絵を描いてみましょう。例えば、家の絵を描いてみます。

▼リスト 5.10　家の絵を描く

```
# 家の絵を描く
# 家の本体（四角形）
pyxel.rect(50, 50, 60, 40, 4)  # 茶色(4)の四角形

# 屋根（三角形）
pyxel.tri(50, 50, 80, 20, 110, 50, 8)  # 赤色(8)の三角形

# ドア
pyxel.rect(70, 70, 20, 20, 5)  # 暗い青色(5)の四角形

# 窓
pyxel.rect(60, 55, 10, 10, 7)  # 白色(7)の四角形
pyxel.rect(90, 55, 10, 10, 7)  # 白色(7)の四角形

# 煙突
pyxel.rect(95, 30, 10, 20, 14)  # 灰色(14)の四角形
```

--[[path = draw_house_image (not exist)]]--

▲図 5.9　家の絵の例

5.12　完成したプログラム

これまで学んだ図形描画の方法を使って、完成したプログラムを作ってみましょう。

▼リスト 5.11　完成したプログラム

```
import pyxel

# Pyxelを初期化
pyxel.init(160, 120, title="図形を描こう")

# 画面を更新する関数
def update():
    # ESCキーが押されたら終了
    if pyxel.btnp(pyxel.KEY_ESCAPE):
        pyxel.quit()

# 画面を描画する関数
def draw():
    # 画面を水色(6)でクリア（空の色）
    pyxel.cls(6)

    # 太陽
    pyxel.circ(120, 30, 15, 10)  # 黄色(10)の円

    # 地面
    pyxel.rect(0, 100, 160, 20, 11)  # 黄緑色(11)の四角形

    # 家の本体
    pyxel.rect(50, 60, 60, 40, 4)  # 茶色(4)の四角形

    # 屋根
    pyxel.tri(50, 60, 80, 30, 110, 60, 8)  # 赤色(8)の三角形

    # ドア
    pyxel.rect(70, 80, 20, 20, 5)  # 暗い青色(5)の四角形
    pyxel.circ(75, 90, 2, 0)  # 黒色(0)の円（ドアノブ）

    # 窓
    pyxel.rect(60, 65, 10, 10, 7)  # 白色(7)の四角形
    pyxel.rect(90, 65, 10, 10, 7)  # 白色(7)の四角形

    # 煙突
    pyxel.rect(95, 40, 10, 20, 14)  # 灰色(14)の四角形
```

```
    # 木
    pyxel.rect(20, 70, 10, 30, 4)  # 茶色(4)の四角形（幹）
    pyxel.circ(25, 60, 15, 3)  # 緑色(3)の円（葉）

    # 雲
    pyxel.circ(30, 25, 10, 7)  # 白色(7)の円
    pyxel.circ(40, 25, 10, 7)  # 白色(7)の円
    pyxel.circ(50, 25, 10, 7)  # 白色(7)の円

    # 文字
    pyxel.text(55, 10, "My First Drawing", 0)  # 黒色(0)の文字

# Pyxelの実行
pyxel.run(update, draw)
```

このプログラムを実行すると、家と木と太陽がある簡単な風景が描かれます。

--[[path = complete_program_image (not exist)]]--

▲図 5.10　完成したプログラムの実行結果

5.13　まとめ

この章では、Pyxel を使ってさまざまな図形を描く方法を学びました。

- 点を描く：‘pset‘関数
- 線を描く：‘line‘関数
- 四角形を描く：‘rect‘関数（塗りつぶし）、‘rectb‘関数（枠のみ）
- 円を描く：‘circ‘関数（塗りつぶし）、‘circb‘関数（枠のみ）
- 三角形を描く：‘tri‘関数（塗りつぶし）、‘trib‘関数（枠のみ）
- 図形を組み合わせて絵を描く方法

これらの図形描画の命令を使いこなせば、さまざまな絵を描くことができます。次の章では、キャラクターを動かす方法を学びます。

5.14　チャレンジ問題

1. 自分の好きな色や形を使って、家の絵を変更してみよう。例えば、家の色を変えたり、窓の形を変えたりしてみよう。
2. 完成したプログラムに、次のものを追加してみよう。* 別の木* 花* 動物（猫や犬な

ど）* 虹* 星

3. 自分だけのキャラクターを図形を組み合わせて描いてみよう。例えば、ロボットやモンスターなど、想像力を働かせて描いてみよう。

第 6 章

キャラクターを動かそう

この章では、キーボードの入力を受け付けて、キャラクターを画面上で動かす方法を学びます。

6.1　キャラクターの動きの基本

ゲームでは、プレイヤーがキーボードやマウスを操作して、キャラクターを動かすことが基本です。Pyxel では、キーボードの入力を簡単に受け付けることができます。

キャラクターを動かすには、次の手順が必要です。

1. キャラクターの位置を記録する変数を用意する
2. キーボードの入力を受け付ける
3. 入力に応じてキャラクターの位置を更新する
4. 新しい位置にキャラクターを描画する

それでは、順番に見ていきましょう。

6.2　基本的なプログラム構造

まずは、キャラクターを動かすための基本的なプログラム構造を確認しましょう。

▼リスト 6.1　基本的なプログラム構造

```
import pyxel

# Pyxelを初期化
pyxel.init(160, 120, title="キャラクターを動かそう")
```

```
# キャラクターの位置
player_x = 80  # 初期X座標
player_y = 60  # 初期Y座標

# 画面を更新する関数
def update():
    global player_x, player_y

    # ESCキーが押されたら終了
    if pyxel.btnp(pyxel.KEY_ESCAPE):
        pyxel.quit()

    # ここにキャラクターを動かすコードを書きます

# 画面を描画する関数
def draw():
    # 画面を黒色（0）でクリア
    pyxel.cls(0)

    # キャラクターを描画
    pyxel.circ(player_x, player_y, 10, 7)  # 白色(7)の円をキャラクターとして描画

# Pyxelの実行
pyxel.run(update, draw)
```

このプログラムをファイル名「character.py」として保存し、これから学ぶキャラクター移動のコードを ‘update‘関数の中に追加していきます。

41

6.3 グローバル変数

プログラムの最初に、キャラクターの位置を記録する変数（‘player_x‘と ‘player_y‘）を定義しています。これらの変数は「グローバル変数」と呼ばれ、プログラム全体で使うことができます。

▼リスト 6.2　グローバル変数

```
# キャラクターの位置
player_x = 80  # 初期X座標
player_y = 60  # 初期Y座標
```

‘update‘関数の中でこれらの変数を変更するには、‘global‘キーワードを使って、これらの変数がグローバル変数であることを宣言する必要があります。

▼リスト 6.3　global キーワード

```
def update():
    global player_x, player_y

    # ここでplayer_xとplayer_yを変更できます
```

6.4 キーボード入力の受け付け

Pyxel では、キーボードの入力を受け付けるために、‘btn‘関数と ‘btnp‘関数を使います。

- ‘btn‘：キーが押されている間、常に ‘True‘を返します。
- ‘btnp‘：キーが押された瞬間だけ ‘True‘を返します。

今回は、キャラクターを連続して動かしたいので、‘btn‘関数を使います。

▼リスト 6.4　キーボード入力の受け付け

```
# キーボード入力の受け付け
if pyxel.btn(pyxel.KEY_LEFT):
    player_x -= 2  # 左キーが押されたら、X座標を減らす（左に移動）

if pyxel.btn(pyxel.KEY_RIGHT):
    player_x += 2  # 右キーが押されたら、X座標を増やす（右に移動）

if pyxel.btn(pyxel.KEY_UP):
    player_y -= 2  # 上キーが押されたら、Y座標を減らす（上に移動）

if pyxel.btn(pyxel.KEY_DOWN):
    player_y += 2  # 下キーが押されたら、Y座標を増やす（下に移動）
```

このコードでは、矢印キー（←→↑↓）の入力を受け付けて、キャラクターの位置（‘player_x‘と ‘player_y‘）を更新しています。

6.5 画面の端での処理

このままだと、キャラクターが画面の外に出てしまう可能性があります。画面の端に来たら、それ以上移動できないようにしましょう。

▼リスト 6.5　画面の端での処理

```
# 画面の端での処理
if player_x < 10:
    player_x = 10  # 左端
if player_x > 150:
    player_x = 150  # 右端
if player_y < 10:
    player_y = 10  # 上端
if player_y > 110:
    player_y = 110  # 下端
```

このコードでは、キャラクターの位置が画面の端（余白を考慮）に来たら、それ以上移動できないようにしています。

6.6 キャラクターの描画

‘draw‘関数では、更新されたキャラクターの位置に円を描画しています。

▼リスト 6.6　キャラクターの描画

```
# キャラクターを描画
pyxel.circ(player_x, player_y, 10, 7)  # 白色(7)の円をキャラクターとして描画
```


ここでは簡単のために円を使っていますが、前の章で学んだ様々な図形を組み合わせて、もっと複雑なキャラクターを描くこともできます。

6.7 完成したプログラム

これまで学んだキャラクター移動の方法を使って、完成したプログラムを作ってみましょう。

▼リスト 6.7　完成したプログラム

```
import pyxel

# Pyxelを初期化
pyxel.init(160, 120, title="キャラクターを動かそう")

# キャラクターの位置
player_x = 80  # 初期X座標
player_y = 60  # 初期Y座標

# 画面を更新する関数
def update():
```

```
    global player_x, player_y

    # ESCキーが押されたら終了
    if pyxel.btnp(pyxel.KEY_ESCAPE):
        pyxel.quit()

    # キーボード入力の受け付け
    if pyxel.btn(pyxel.KEY_LEFT):
        player_x -= 2  # 左キーが押されたら、X座標を減らす（左に移動）

    if pyxel.btn(pyxel.KEY_RIGHT):
        player_x += 2  # 右キーが押されたら、X座標を増やす（右に移動）

    if pyxel.btn(pyxel.KEY_UP):
        player_y -= 2  # 上キーが押されたら、Y座標を減らす（上に移動）

    if pyxel.btn(pyxel.KEY_DOWN):
        player_y += 2  # 下キーが押されたら、Y座標を増やす（下に移動）

    # 画面の端での処理
    if player_x < 10:
        player_x = 10  # 左端
    if player_x > 150:
        player_x = 150  # 右端
    if player_y < 10:
        player_y = 10  # 上端
    if player_y > 110:
        player_y = 110  # 下端

# 画面を描画する関数
def draw():
    # 画面を黒色（0）でクリア
    pyxel.cls(0)

    # 背景に星を描く
    for i in range(16):
        pyxel.pset(i * 10, 20, 7)  # 白色(7)の点
        pyxel.pset(i * 10 + 5, 40, 7)  # 白色(7)の点
        pyxel.pset(i * 10, 60, 7)  # 白色(7)の点
        pyxel.pset(i * 10 + 5, 80, 7)  # 白色(7)の点
        pyxel.pset(i * 10, 100, 7)  # 白色(7)の点

    # キャラクターを描画（宇宙飛行士）
    # 体
    pyxel.circ(player_x, player_y, 10, 7)  # 白色(7)の円

    # 顔
    pyxel.circ(player_x, player_y - 2, 6, 6)  # 水色(6)の円

    # 目
    pyxel.pset(player_x - 2, player_y - 3, 0)  # 黒色(0)の点
    pyxel.pset(player_x + 2, player_y - 3, 0)  # 黒色(0)の点

    # 口
    pyxel.line(player_x - 2, player_y, player_x + 2, player_y, 0)  # 黒色(0)の線

    # 手足
```

```
    pyxel.line(player_x - 10, player_y, player_x - 15, player_y - 5, 7)  # 左手
    pyxel.line(player_x + 10, player_y, player_x + 15, player_y - 5, 7)  # 右手
    pyxel.line(player_x - 5, player_y + 10, player_x - 5, player_y + 15, 7)  # 左
足
    pyxel.line(player_x + 5, player_y + 10, player_x + 5, player_y + 15, 7)  # 右
足

    # 操作方法を表示
    pyxel.text(5, 5, "Arrow keys: Move", 7)

# Pyxelの実行
pyxel.run(update, draw)
```

このプログラムを実行すると、矢印キーで宇宙飛行士のキャラクターを動かすことができます。

--[[path = complete_program_image (not exist)]]--

▲図 6.1　完成したプログラムの実行結果

6.8　応用：スピードの変更

4
5

キャラクターの移動スピードを変更するには、座標の増減値を変更します。

▼リスト 6.8　スピードの変更

```
# スピードを変更（1から5の間で調整）
speed = 3

# キーボード入力の受け付け
if pyxel.btn(pyxel.KEY_LEFT):
    player_x -= speed  # 左に移動

if pyxel.btn(pyxel.KEY_RIGHT):
    player_x += speed  # 右に移動

if pyxel.btn(pyxel.KEY_UP):
    player_y -= speed  # 上に移動

if pyxel.btn(pyxel.KEY_DOWN):
    player_y += speed  # 下に移動
```

6.9　応用：斜め移動

斜め移動をスムーズにするには、同時に押されたキーを考慮する必要があります。

▼リスト 6.9　斜め移動

```
# 移動方向を記録する変数
dx = 0  # X方向の移動量
dy = 0  # Y方向の移動量

# キーボード入力の受け付け
if pyxel.btn(pyxel.KEY_LEFT):
    dx = -2  # 左に移動
elif pyxel.btn(pyxel.KEY_RIGHT):
    dx = 2   # 右に移動
else:
    dx = 0   # X方向には移動しない

if pyxel.btn(pyxel.KEY_UP):
    dy = -2  # 上に移動
elif pyxel.btn(pyxel.KEY_DOWN):
    dy = 2   # 下に移動
else:
    dy = 0   # Y方向には移動しない

# 位置の更新
player_x += dx
player_y += dy
```

6.10　応用：アニメーション

キャラクターにアニメーションをつけるには、フレームカウンターを使います。

▼リスト 6.10　アニメーション

```
# フレームカウンター
frame_count = 0

def update():
    global player_x, player_y, frame_count

    # フレームカウンターを更新
    frame_count = (frame_count + 1) % 30  # 0から29までのループ

    # キャラクターの移動処理...

def draw():
```

```
    # 画面のクリア...

    # アニメーションするキャラクター
    if frame_count < 15:
        # アニメーション1コマ目
        pyxel.circ(player_x, player_y, 10, 7)
        # 足を前に出す
        pyxel.line(player_x - 5, player_y + 10, player_x - 10, player_y + 15, 7)  # 左
足
        pyxel.line(player_x + 5, player_y + 10, player_x + 10, player_y + 15, 7)  # 右
足
    else:
        # アニメーション2コマ目
        pyxel.circ(player_x, player_y, 10, 7)
        # 足を後ろに引く
        pyxel.line(player_x - 5, player_y + 10, player_x - 5, player_y + 15, 7)  # 左
足
        pyxel.line(player_x + 5, player_y + 10, player_x + 5, player_y + 15, 7)  # 右
足
```

6.11　まとめ

この章では、キーボード入力を受け付けて、キャラクターを画面上で動かす方法を学びました。

- キャラクターの位置を記録する変数の使い方
- キーボード入力の受け付け方法（‘btn‘関数）
- 画面の端での処理方法
- キャラクターの描画方法
- 応用テクニック（スピード変更、斜め移動、アニメーション）

これらの技術を使えば、プレイヤーが操作できるキャラクターを作ることができます。次の章では、音を鳴らす方法を学びます。

6.12　チャレンジ問題

1. キャラクターの見た目を変えてみよう。円の代わりに、四角形や三角形を使ったり、複数の図形を組み合わせたりしてみよう。

2. キャラクターの移動スピードを変更してみよう。速すぎず、遅すぎない、ちょうどいいスピードを見つけよう。

3. スペースキー（‘pyxel.KEY_SPACE‘）を押したときに、キャラクターの色が変わるようにしてみよう。

4. 画面上に障害物（例：四角形）を配置して、キャラクターがぶつからないように移動させるゲームを作ってみよう。

第 7 章

音を鳴らしてみよう

この章では、Pyxel を使って効果音や BGM（バックグラウンドミュージック）を再生する方法を学びます。

7.1　ゲームの音の重要性

ゲームには、画面に表示される映像だけでなく、音も重要な要素です。効果音や BGM があることで、ゲームはより楽しく、臨場感のあるものになります。

Pyxel では、レトロゲーム風の 8 ビットサウンドを簡単に作成して再生することができます。

7.2　Pyxel のサウンド機能

Pyxel には、次の 2 種類のサウンド機能があります。

1. 効果音（Sound Effects）：短い音を鳴らす機能
2. BGM（Background Music）：複数の効果音を組み合わせて、曲を作る機能

これらの機能を使って、ゲームに音を追加していきましょう。

7.3　効果音を鳴らす

まずは、効果音を鳴らす方法を学びましょう。効果音を鳴らすには、次の手順が必要です。

1. 効果音を定義する

2. 効果音を再生する

7.3.1 効果音の定義

効果音は、‘sound‘関数を使って定義します。

▼リスト 7.1　効果音の定義

```
# 効果音の定義
pyxel.sound(0).set(
    note="C3 C4 C5",  # 音階
    tone="T",         # 音色（T:三角波、S:矩形波、P:パルス波、N:ノイズ）
    volume="7",       # 音量（0〜7）
    effect="N",       # エフェクト（N:なし、S:スライド、V:ビブラート、F:フェードアウト）
    speed=20          # 再生速度（数値が大きいほど遅い）
)
```

この例では、「C3」「C4」「C5」という 3 つの音階を順番に鳴らす効果音を定義しています。

7.3.2 効果音の再生

定義した効果音は、‘play‘関数を使って再生します。

▼リスト 7.2　効果音の再生

```
# 効果音の再生
pyxel.play(0, 0)  # チャンネル0で、サウンド番号0を再生
```

‘play‘関数の引数は次のとおりです。

- 第 1 引数：チャンネル番号（0〜3）
- 第 2 引数：サウンド番号（‘sound‘関数で定義した番号）

7.4 基本的なプログラム構造

効果音を使った基本的なプログラム構造を確認しましょう。

▼リスト 7.3　基本的なプログラム構造

```
import pyxel

# Pyxelを初期化
pyxel.init(160, 120, title="音を鳴らそう")

# 効果音の定義
pyxel.sound(0).set(
    note="C3 C4 C5",
    tone="T",
    volume="7",
    effect="N",
    speed=20
)

# 画面を更新する関数
def update():
    # ESCキーが押されたら終了
    if pyxel.btnp(pyxel.KEY_ESCAPE):
        pyxel.quit()

    # スペースキーが押されたら効果音を再生
    if pyxel.btnp(pyxel.KEY_SPACE):
        pyxel.play(0, 0)

# 画面を描画する関数
def draw():
    # 画面を黒色（0）でクリア
    pyxel.cls(0)

    # 操作方法を表示
    pyxel.text(30, 60, "Press SPACE to play sound", 7)

# Pyxelの実行
pyxel.run(update, draw)
```

このプログラムをファイル名「sound.py」として保存し、実行してみましょう。スペースキーを押すと、効果音が鳴ります。

7.5 様々な効果音

Pyxel では、音階、音色、音量、エフェクト、再生速度を変えることで、様々な効果音を作ることができます。

7.5.1 音階（note）

音階は、「C（ド）」「D（レ）」「E（ミ）」「F（ファ）」「G（ソ）」「A（ラ）」「B（シ）」の 7 つの音と、その派生音（「C#」「D#」など）を使って指定します。数字は音の高さ（オクターブ）を表し、数字が大きいほど高い音になります。

▼リスト 7.4　音階の例

```
# 上昇音
pyxel.sound(0).set(note="C3 D3 E3 F3 G3 A3 B3 C4", tone="T", volume="7", effect="N", speed=10)

# 下降音
pyxel.sound(1).set(note="C5 B4 A4 G4 F4 E4 D4 C4", tone="T", volume="7", effect="N", speed=10)

# ドレミファソラシド
pyxel.sound(2).set(note="C4 D4 E4 F4 G4 A4 B4 C5", tone="T", volume="7", effect="N", speed=10)
```

7.5.2　音色（tone）

音色は、次の 4 種類から選べます。

- T：三角波（柔らかい音）
- S：矩形波（はっきりした音）
- P：パルス波（鋭い音）
- N：ノイズ（雑音）

▼リスト 7.5　音色の例

```
# 三角波
pyxel.sound(0).set(note="C4 E4 G4", tone="T", volume="7", effect="N", speed=10)

# 矩形波
pyxel.sound(1).set(note="C4 E4 G4", tone="S", volume="7", effect="N", speed=10)

# パルス波
pyxel.sound(2).set(note="C4 E4 G4", tone="P", volume="7", effect="N", speed=10)

# ノイズ
pyxel.sound(3).set(note="C4 E4 G4", tone="N", volume="7", effect="N", speed=10)
```

7.5.3　音量（volume）

音量は、0 から 7 までの数字で指定します。数字が大きいほど音が大きくなります。

▼リスト 7.6　音量の例

```
# 大きい音
pyxel.sound(0).set(note="C4 E4 G4", tone="T", volume="7", effect="N", speed=10)

# 中くらいの音
pyxel.sound(1).set(note="C4 E4 G4", tone="T", volume="4", effect="N", speed=10)

# 小さい音
pyxel.sound(2).set(note="C4 E4 G4", tone="T", volume="1", effect="N", speed=10)
```

7.5.4 エフェクト（effect）

エフェクトは、次の 4 種類から選べます。

- N：なし（通常の再生）
- S：スライド（音が滑らかにつながる）
- V：ビブラート（音が揺れる）
- F：フェードアウト（音が徐々に小さくなる）

▼リスト 7.7　エフェクトの例

```
# エフェクトなし
pyxel.sound(0).set(note="C4 E4 G4", tone="T", volume="7", effect="N", speed=10)

# スライド
pyxel.sound(1).set(note="C4 E4 G4", tone="T", volume="7", effect="S", speed=10)

# ビブラート
pyxel.sound(2).set(note="C4 E4 G4", tone="T", volume="7", effect="V", speed=10)

# フェードアウト
pyxel.sound(3).set(note="C4 E4 G4", tone="T", volume="7", effect="F", speed=10)
```

7.5.5 再生速度（speed）

再生速度は、数値で指定します。数値が小さいほど速く再生され、大きいほど遅く再生されます。

▼リスト 7.8　再生速度の例

```
# 速い
pyxel.sound(0).set(note="C4 E4 G4", tone="T", volume="7", effect="N", speed=5)

# 普通
pyxel.sound(1).set(note="C4 E4 G4", tone="T", volume="7", effect="N", speed=10)

# 遅い
pyxel.sound(2).set(note="C4 E4 G4", tone="T", volume="7", effect="N", speed=20)
```

7.6　ゲームに効果音を追加する

前の章で作ったキャラクター移動のプログラムに、効果音を追加してみましょう。

▼リスト 7.9　ゲームに効果音を追加する

```
import pyxel

# Pyxelを初期化
pyxel.init(160, 120, title="キャラクターと音")

# キャラクターの位置
player_x = 80
player_y = 60

# 効果音の定義
# 移動音
pyxel.sound(0).set(note="C3", tone="P", volume="3", effect="N", speed=5)
# ジャンプ音
pyxel.sound(1).set(note="C5 G4", tone="S", volume="4", effect="N", speed=10)
# アイテム取得音
pyxel.sound(2).set(note="C4 E4 G4 C5", tone="T", volume="5", effect="N", speed=5)

# 画面を更新する関数
def update():
    global player_x, player_y

    # ESCキーが押されたら終了
    if pyxel.btnp(pyxel.KEY_ESCAPE):
        pyxel.quit()

    # キーボード入力の受け付け
    if pyxel.btn(pyxel.KEY_LEFT):
        player_x -= 2
        pyxel.play(0, 0)  # 移動音を再生

    if pyxel.btn(pyxel.KEY_RIGHT):
        player_x += 2
        pyxel.play(0, 0)  # 移動音を再生

    if pyxel.btn(pyxel.KEY_UP):
        player_y -= 2
```

```
        pyxel.play(0, 0)  # 移動音を再生

    if pyxel.btn(pyxel.KEY_DOWN):
        player_y += 2
        pyxel.play(0, 0)  # 移動音を再生

    # スペースキーでジャンプ音
    if pyxel.btnp(pyxel.KEY_SPACE):
        pyxel.play(1, 1)  # ジャンプ音を再生

    # Zキーでアイテム取得音
    if pyxel.btnp(pyxel.KEY_Z):
        pyxel.play(2, 2)  # アイテム取得音を再生

    # 画面の端での処理
    if player_x < 10:
        player_x = 10
    if player_x > 150:
        player_x = 150
    if player_y < 10:
        player_y = 10
    if player_y > 110:
        player_y = 110

# 画面を描画する関数
def draw():
    # 画面を黒色（0）でクリア
    pyxel.cls(0)

    # キャラクターを描画
    pyxel.circ(player_x, player_y, 10, 7)

    # 操作方法を表示
    pyxel.text(5, 5, "Arrow keys: Move", 7)
    pyxel.text(5, 15, "Space: Jump sound", 7)
    pyxel.text(5, 25, "Z: Item sound", 7)

# Pyxelの実行
pyxel.run(update, draw)
```


このプログラムでは、次の効果音を追加しています。

- 矢印キーを押したときに移動音が鳴る
- スペースキーを押したときにジャンプ音が鳴る
- Z キーを押したときにアイテム取得音が鳴る

7.7 BGM を作る

BGM（バックグラウンドミュージック）は、複数の効果音を組み合わせて作ります。BGM を作るには、次の手順が必要です。

1. 効果音を定義する
2. BGM を定義する
3. BGM を再生する

7.7.1 BGM の定義

BGM は、‘music‘関数を使って定義します。

▼リスト 7.10　BGM の定義

```
# 効果音の定義
pyxel.sound(0).set(note="C3 C4 C5", tone="T", volume="7", effect="N", speed=20)
pyxel.sound(1).set(note="E3 E4 E5", tone="S", volume="6", effect="N", speed=20)
pyxel.sound(2).set(note="G3 G4 G5", tone="P", volume="5", effect="N", speed=20)

# BGMの定義
pyxel.music(0).set(
    [0, 1, 2],  # トラック0, 1, 2を使用
    [           # 各トラックの再生パターン
        [0, 1],  # トラック0: サウンド0, 1を順番に再生
        [2, 0],  # トラック1: サウンド2, 0を順番に再生
        [1, 2]   # トラック2: サウンド1, 2を順番に再生
    ]
)
```

7.7.2 BGM の再生

定義した BGM は、‘playm‘関数を使って再生します。

▼リスト 7.11　BGM の再生

```
# BGMの再生
pyxel.playm(0)  # ミュージック番号0を再生
```

7.8 BGM を使ったプログラム

BGM を使った簡単なプログラムを作ってみましょう。

▼リスト 7.12　BGM を使ったプログラム

```
import pyxel

# Pyxelを初期化
pyxel.init(160, 120, title="BGMを鳴らそう")

# 効果音の定義
# メロディ
pyxel.sound(0).set(note="C4 D4 E4 F4", tone="T", volume="7", effect="N", speed=10)
pyxel.sound(1).set(note="G4 A4 B4 C5", tone="T", volume="7", effect="N", speed=10)
# ベース
pyxel.sound(2).set(note="C3 C3 C3 C3", tone="S", volume="5", effect="N", speed=10)
pyxel.sound(3).set(note="G2 G2 G2 G2", tone="S", volume="5", effect="N", speed=10)
# ドラム
pyxel.sound(4).set(note="F2 F2 F2 F2", tone="N", volume="3", effect="N", speed=10)

# BGMの定義
pyxel.music(0).set(
    [0, 1, 2],  # トラック0, 1, 2を使用
    [
        [0, 1],  # トラック0: メロディ
        [2, 3],  # トラック1: ベース
        [4, 4]   # トラック2: ドラム
    ]
)

# BGM再生フラグ
bgm_playing = False

# 画面を更新する関数
def update():
    global bgm_playing

    # ESCキーが押されたら終了
    if pyxel.btnp(pyxel.KEY_ESCAPE):
        pyxel.quit()

    # スペースキーでBGMの再生/停止を切り替え
    if pyxel.btnp(pyxel.KEY_SPACE):
        if bgm_playing:
            pyxel.stop()  # BGMを停止
            bgm_playing = False
        else:
            pyxel.playm(0)  # BGMを再生
            bgm_playing = True

# 画面を描画する関数
def draw():
    # 画面を黒色 (0) でクリア
    pyxel.cls(0)

    # 操作方法と状態を表示
    pyxel.text(30, 50, "Press SPACE to toggle BGM", 7)

    if bgm_playing:
        pyxel.text(50, 70, "BGM: Playing", 11)
    else:
        pyxel.text(50, 70, "BGM: Stopped", 8)
```

```
# Pyxelの実行
pyxel.run(update, draw)
```

このプログラムでは、スペースキーを押すと BGM の再生/停止を切り替えることができます。

7.9 まとめ

この章では、Pyxel を使って効果音や BGM を再生する方法を学びました。

- 効果音の定義と再生方法
- 音階、音色、音量、エフェクト、再生速度の設定方法
- ゲームに効果音を追加する方法
- BGM の定義と再生方法

音を追加することで、ゲームはより楽しく、臨場感のあるものになります。次の章では、これまで学んだ知識を使って、簡単なゲームを作ります。

7.10 チャレンジ問題

1. 自分だけの効果音を作ってみよう。音階、音色、音量、エフェクト、再生速度を変えて、いろいろな音を試してみよう。

2. 前の章で作ったキャラクター移動のプログラムに、次の効果音を追加してみよう。* 画面の端に到達したときの効果音* キャラクターが特定の位置に来たときの効果音

3. 自分だけの BGM を作ってみよう。複数の効果音を組み合わせて、短い曲を作ってみよう。

第 8 章

簡単なゲームを作ろう

この章では、これまで学んだ知識を使って、ボールキャッチゲームを作ります。

8.1　ゲームの概要

今回作るゲームは、「ボールキャッチゲーム」です。画面の上から落ちてくるボールを、プレイヤーが操作するバスケットでキャッチするゲームです。

ゲームのルールは次のとおりです。

- プレイヤーは左右の矢印キーでバスケットを動かします。
- 上から落ちてくるボールをバスケットでキャッチすると、得点が増えます。
- ボールを取り逃すと、ゲームオーバーになります。
- 得点が高いほど、ボールの落下速度が速くなります。

8.2　ゲームの設計

ゲームを作る前に、どのような要素が必要か考えましょう。

1. プレイヤー（バスケット）* 位置* 移動方法* 描画方法
2. ボール* 位置* 落下速度* 描画方法
3. ゲームの状態* スコア* ゲームオーバーフラグ
4. 当たり判定* バスケットとボールの衝突判定
5. 効果音* ボールをキャッチしたときの音* ゲームオーバーになったときの音

これらの要素を順番に実装していきましょう。

8.3　基本的なプログラム構造

まずは、ゲームの基本的なプログラム構造を作ります。

▼リスト 8.1　基本的なプログラム構造

```
import pyxel
import random

# Pyxelを初期化
pyxel.init(160, 120, title="ボールキャッチゲーム")

# ゲームの初期化
def init_game():
    global player_x, ball_x, ball_y, ball_speed, score, game_over

    # プレイヤー（バスケット）の位置
    player_x = 80

    # ボールの位置と速度
    ball_x = random.randint(10, 150)
    ball_y = 0
    ball_speed = 2

    # ゲームの状態
    score = 0
    game_over = False

# ゲームの初期化を実行
init_game()

# 効果音の定義
# キャッチ音
pyxel.sound(0).set(note="C4 E4 G4", tone="T", volume="7", effect="N", speed=5)
# ゲームオーバー音
pyxel.sound(1).set(note="C3 C2", tone="S", volume="7", effect="F", speed=10)

# 画面を更新する関数
def update():
    global player_x, ball_x, ball_y, ball_speed, score, game_over

    # ESCキーが押されたら終了
    if pyxel.btnp(pyxel.KEY_ESCAPE):
        pyxel.quit()

    # ゲームオーバー時の処理
    if game_over:
        # スペースキーでリスタート
        if pyxel.btnp(pyxel.KEY_SPACE):
            init_game()
        return

    # プレイヤーの移動
    if pyxel.btn(pyxel.KEY_LEFT):
```

```
        player_x = max(player_x - 4, 20)

    if pyxel.btn(pyxel.KEY_RIGHT):
        player_x = min(player_x + 4, 140)

    # ボールの落下
    ball_y += ball_speed

    # ボールがバスケットに入ったかチェック
    if ball_y > 100 and ball_y < 110:
        if abs(ball_x - player_x) < 20:
            # ボールをキャッチした
            score += 1
            ball_x = random.randint(10, 150)
            ball_y = 0
            ball_speed = min(2 + score * 0.1, 6)  # スコアに応じて速度アップ（最大6）

            pyxel.play(0, 0)  # キャッチ音を再生

    # ボールが画面外に出たらゲームオーバー
    if ball_y > 120:
        game_over = True
        pyxel.play(1, 1)  # ゲームオーバー音を再生

# 画面を描画する関数
def draw():
    # 画面を青色（12）でクリア
    pyxel.cls(12)

    # スコアを表示
    pyxel.text(5, 5, f"Score: {score}", 7)

    # プレイヤー（バスケット）を描画
    pyxel.rect(player_x - 20, 100, 40, 10, 14)  # 灰色(14)の四角形
    pyxel.rectb(player_x - 20, 100, 40, 10, 0)  # 黒色(0)の枠

    # ボールを描画
    pyxel.circ(ball_x, ball_y, 5, 8)  # 赤色(8)の円

    # ゲームオーバー時のメッセージ
    if game_over:
        pyxel.text(55, 50, "GAME OVER", 8)
        pyxel.text(35, 70, "Press SPACE to restart", 7)

# Pyxelの実行
pyxel.run(update, draw)
```

このプログラムをファイル名「ball_catch.py」として保存し、実行してみましょう。左右の矢印キーでバスケットを動かし、落ちてくるボールをキャッチします。ボールを取り逃すとゲームオーバーになり、スペースキーでリスタートできます。

--[[path = ball_catch_game (not exist)]]--

▲図 8.1　ボールキャッチゲームの実行画面

8.4 プログラムの解説

ボールキャッチゲームのプログラムを詳しく見ていきましょう。

8.4.1 ライブラリのインポート

▼リスト 8.2 ライブラリのインポート

```
import pyxel
import random
```

Pyxel に加えて、‘random‘ライブラリをインポートしています。これは、ボールの初期位置をランダムに設定するために使います。

8.4.2 ゲームの初期化

▼リスト 8.3 ゲームの初期化

```
# ゲームの初期化
def init_game():
    global player_x, ball_x, ball_y, ball_speed, score, game_over

    # プレイヤー（バスケット）の位置
    player_x = 80

    # ボールの位置と速度
    ball_x = random.randint(10, 150)
    ball_y = 0
    ball_speed = 2

    # ゲームの状態
    score = 0
    game_over = False

# ゲームの初期化を実行
init_game()
```

‘init_game‘関数では、ゲームの状態を初期化しています。

- ‘player_x‘：プレイヤー（バスケット）の X 座標
- ‘ball_x‘、‘ball_y‘：ボールの座標

- ‘ball_speed‘：ボールの落下速度
- ‘score‘：スコア
- ‘game_over‘：ゲームオーバーフラグ

‘random.randint(10, 150)‘は、10 から 150 までのランダムな整数を返します。これにより、ボールの初期 X 座標がランダムに設定されます。

8.4.3 効果音の定義

▼リスト 8.4　効果音の定義

```
# 効果音の定義
# キャッチ音
pyxel.sound(0).set(note="C4 E4 G4", tone="T", volume="7", effect="N", speed=5)
# ゲームオーバー音
pyxel.sound(1).set(note="C3 C2", tone="S", volume="7", effect="F", speed=10)
```

2 つの効果音を定義しています。

- サウンド 0：ボールをキャッチしたときの音（明るい上昇音）
- サウンド 1：ゲームオーバーになったときの音（暗い下降音）

8.4.4 update 関数

‘update‘関数では、ゲームの状態を更新します。

ゲームオーバー時の処理

▼リスト 8.5　ゲームオーバー時の処理

```
# ゲームオーバー時の処理
if game_over:
    # スペースキーでリスタート
    if pyxel.btnp(pyxel.KEY_SPACE):
        init_game()
    return
```

ゲームオーバー状態の場合、スペースキーが押されたらゲームを初期化します。‘return‘文により、以降の処理はスキップされます。

プレイヤーの移動

▼リスト 8.6　プレイヤーの移動

```
# プレイヤーの移動
if pyxel.btn(pyxel.KEY_LEFT):
    player_x = max(player_x - 4, 20)

if pyxel.btn(pyxel.KEY_RIGHT):
    player_x = min(player_x + 4, 140)
```

左右の矢印キーでプレイヤー（バスケット）を移動します。‘max‘と ‘min‘関数を使って、画面の端を超えないようにしています。

ボールの落下と当たり判定

▼リスト 8.7　ボールの落下と当たり判定

```
# ボールの落下
ball_y += ball_speed

# ボールがバスケットに入ったかチェック
if ball_y > 100 and ball_y < 110:
    if abs(ball_x - player_x) < 20:
        # ボールをキャッチした
        score += 1
        ball_x = random.randint(10, 150)
        ball_y = 0
        ball_speed = min(2 + score * 0.1, 6)  # スコアに応じて速度アップ（最大6）
        pyxel.play(0, 0)  # キャッチ音を再生

# ボールが画面外に出たらゲームオーバー
if ball_y > 120:
    game_over = True
    pyxel.play(1, 1)  # ゲームオーバー音を再生
```

ボールは ‘ball_speed‘の速度で下に落ちていきます。ボールがバスケットの高さ（Y 座標が 100〜110）にあり、かつ X 座標がバスケットの範囲内（プレイヤーの X 座標から±20）にある場合、ボールをキャッチしたと判定します。

キャッチした場合は、スコアを増やし、ボールを画面上部にリセットします。また、スコアに応じてボールの速度を上げます（ただし最大 6 まで）。

ボールが画面下端（Y 座標が 120）を超えた場合、ゲームオーバーになります。

8.4.5 draw 関数

‘draw‘関数では、ゲームの状態を画面に描画します。

▼リスト 8.8 draw 関数

```
# 画面を描画する関数
def draw():
    # 画面を青色（12）でクリア
    pyxel.cls(12)

    # スコアを表示
    pyxel.text(5, 5, f"Score: {score}", 7)

    # プレイヤー（バスケット）を描画
    pyxel.rect(player_x - 20, 100, 40, 10, 14)  # 灰色(14)の四角形
    pyxel.rectb(player_x - 20, 100, 40, 10, 0)  # 黒色(0)の枠

    # ボールを描画
    pyxel.circ(ball_x, ball_y, 5, 8)  # 赤色(8)の円

    # ゲームオーバー時のメッセージ
    if game_over:
        pyxel.text(55, 50, "GAME OVER", 8)
        pyxel.text(35, 70, "Press SPACE to restart", 7)
```


画面を青色（12）でクリアした後、スコア、プレイヤー（バスケット）、ボールを描画します。ゲームオーバー時には、「GAME OVER」と「Press SPACE to restart」というメッセージを表示します。

8.5 ゲームの改良

基本的なボールキャッチゲームができたら、さらに改良してみましょう。

8.5.1 複数のボール

1 つのボールだけでなく、複数のボールを同時に落とすようにしてみましょう。

▼リスト 8.9 複数のボール

```
# ゲームの初期化
def init_game():
    global player_x, balls, score, game_over
```

```
    # プレイヤー（バスケット）の位置
    player_x = 80

    # ボールのリスト（各ボールは[x座標, y座標, 速度]のリスト）
    balls = []
    add_ball()

    # ゲームの状態
    score = 0
    game_over = False

# 新しいボールを追加
def add_ball():
    global balls
    balls.append([random.randint(10, 150), 0, random.uniform(1, 3)])

# 画面を更新する関数
def update():
    global player_x, balls, score, game_over

    # ESCキーが押されたら終了
    if pyxel.btnp(pyxel.KEY_ESCAPE):
        pyxel.quit()

    # ゲームオーバー時の処理
    if game_over:
        # スペースキーでリスタート
        if pyxel.btnp(pyxel.KEY_SPACE):
            init_game()
        return

    # プレイヤーの移動
    if pyxel.btn(pyxel.KEY_LEFT):
        player_x = max(player_x - 4, 20)

    if pyxel.btn(pyxel.KEY_RIGHT):
        player_x = min(player_x + 4, 140)

    # ボールの処理
    for i in range(len(balls) - 1, -1, -1):
        ball_x, ball_y, ball_speed = balls[i]

        # ボールの落下
        ball_y += ball_speed
        balls[i][1] = ball_y

        # ボールがバスケットに入ったかチェック
        if ball_y > 100 and ball_y < 110:
            if abs(ball_x - player_x) < 20:
                # ボールをキャッチした
                score += 1
                balls.pop(i)  # ボールを削除
                add_ball()  # 新しいボールを追加

                # スコアが5の倍数になったら、ボールを追加
                if score % 5 == 0:
                    add_ball()
```

```
                pyxel.play(0, 0)  # キャッチ音を再生

        # ボールが画面外に出たらゲームオーバー
        elif ball_y > 120:
            game_over = True
            pyxel.play(1, 1)  # ゲームオーバー音を再生

# 画面を描画する関数
def draw():
    # 画面を青色（12）でクリア
    pyxel.cls(12)

    # スコアを表示
    pyxel.text(5, 5, f"Score: {score}", 7)

    # プレイヤー（バスケット）を描画
    pyxel.rect(player_x - 20, 100, 40, 10, 14)  # 灰色(14)の四角形
    pyxel.rectb(player_x - 20, 100, 40, 10, 0)  # 黒色(0)の枠

    # ボールを描画
    for ball_x, ball_y, _ in balls:
        pyxel.circ(ball_x, ball_y, 5, 8)  # 赤色(8)の円

    # ゲームオーバー時のメッセージ
    if game_over:
        pyxel.text(55, 50, "GAME OVER", 8)
        pyxel.text(35, 70, "Press SPACE to restart", 7)
```

このプログラムでは、ボールを ‘balls‘というリストで管理しています。各ボールは ‘[x 座標, y 座標, 速度]‘というリストで表現されています。スコアが 5 の倍数になるたびに、新しいボールが追加されます。

8.5.2 ライフシステム

1 回ミスしただけでゲームオーバーになるのは厳しいので、ライフシステムを導入してみましょう。

▼リスト 8.10　ライフシステム

```
# ゲームの初期化
def init_game():
    global player_x, ball_x, ball_y, ball_speed, score, lives, game_over

    # プレイヤー（バスケット）の位置
    player_x = 80

    # ボールの位置と速度
    ball_x = random.randint(10, 150)
    ball_y = 0
    ball_speed = 2
```

```
    # ゲームの状態
    score = 0
    lives = 3  # ライフ数
    game_over = False

# 画面を更新する関数
def update():
    global player_x, ball_x, ball_y, ball_speed, score, lives, game_over

    # ESCキーが押されたら終了
    if pyxel.btnp(pyxel.KEY_ESCAPE):
        pyxel.quit()

    # ゲームオーバー時の処理
    if game_over:
        # スペースキーでリスタート
        if pyxel.btnp(pyxel.KEY_SPACE):
            init_game()
        return

    # プレイヤーの移動
    if pyxel.btn(pyxel.KEY_LEFT):
        player_x = max(player_x - 4, 20)

    if pyxel.btn(pyxel.KEY_RIGHT):
        player_x = min(player_x + 4, 140)

    # ボールの落下
    ball_y += ball_speed

    # ボールがバスケットに入ったかチェック
    if ball_y > 100 and ball_y < 110:
        if abs(ball_x - player_x) < 20:
            # ボールをキャッチした
            score += 1
            ball_x = random.randint(10, 150)
            ball_y = 0
            ball_speed = min(2 + score * 0.1, 6)  # スコアに応じて速度アップ（最大6）

            pyxel.play(0, 0)  # キャッチ音を再生

    # ボールが画面外に出たらライフを減らす
    if ball_y > 120:
        lives -= 1
        ball_x = random.randint(10, 150)
        ball_y = 0
        pyxel.play(1, 1)  # ミス音を再生

        # ライフがなくなったらゲームオーバー
        if lives <= 0:
            game_over = True

# 画面を描画する関数
def draw():
    # 画面を青色（12）でクリア
    pyxel.cls(12)

    # スコアとライフを表示
```

```
    pyxel.text(5, 5, f"Score: {score}", 7)
    pyxel.text(5, 15, f"Lives: {lives}", 7)

    # プレイヤー（バスケット）を描画
    pyxel.rect(player_x - 20, 100, 40, 10, 14)  # 灰色(14)の四角形
    pyxel.rectb(player_x - 20, 100, 40, 10, 0)  # 黒色(0)の枠

    # ボールを描画
    pyxel.circ(ball_x, ball_y, 5, 8)  # 赤色(8)の円

    # ゲームオーバー時のメッセージ
    if game_over:
        pyxel.text(55, 50, "GAME OVER", 8)
        pyxel.text(35, 70, "Press SPACE to restart", 7)
```

このプログラムでは、プレイヤーに 3 つのライフが与えられます。ボールを取り逃すとライフが 1 つ減り、ライフがなくなるとゲームオーバーになります。

8.5.3 ハイスコア

プレイヤーのモチベーションを高めるために、ハイスコアを記録してみましょう。

▼リスト 8.11　ハイスコア

```
# ゲームの初期化
def init_game():
    global player_x, ball_x, ball_y, ball_speed, score, high_score, game_over

    # プレイヤー（バスケット）の位置
    player_x = 80

    # ボールの位置と速度
    ball_x = random.randint(10, 150)
    ball_y = 0
    ball_speed = 2

    # ゲームの状態
    score = 0
    game_over = False

# ハイスコアの初期化
high_score = 0

# 画面を更新する関数
def update():
    global player_x, ball_x, ball_y, ball_speed, score, high_score, game_over

    # ESCキーが押されたら終了
    if pyxel.btnp(pyxel.KEY_ESCAPE):
        pyxel.quit()
```

```
    # ゲームオーバー時の処理
    if game_over:
        # ハイスコアの更新
        if score > high_score:
            high_score = score

        # スペースキーでリスタート
        if pyxel.btnp(pyxel.KEY_SPACE):
            init_game()
        return

    # プレイヤーの移動と当たり判定の処理...

# 画面を描画する関数
def draw():
    # 画面を青色（12）でクリア
    pyxel.cls(12)

    # スコアとハイスコアを表示
    pyxel.text(5, 5, f"Score: {score}", 7)
    pyxel.text(5, 15, f"High Score: {high_score}", 7)

    # プレイヤー、ボール、ゲームオーバーメッセージの描画...
```

このプログラムでは、ゲームオーバー時にスコアがハイスコアを超えていれば、ハイスコアを更新します。

8.6 まとめ

この章では、これまで学んだ知識を使って、ボールキャッチゲームを作りました。

- ゲームの基本構造（初期化、更新、描画）
- プレイヤーの移動と入力の処理
- ボールの落下と当たり判定
- スコアとゲームオーバーの処理
- 効果音の追加
- ゲームの改良（複数のボール、ライフシステム、ハイスコア）

これらの要素は、多くのゲームに共通する基本的な要素です。この知識を応用すれば、さまざまなゲームを作ることができます。

次の章では、もう少し複雑なシューティングゲームを作ります。

8.7 チャレンジ問題

1. ボールの色をランダムに変えてみよう。色によって得点が変わるようにしてみよう。

2. ボールの大きさをランダムに変えてみよう。大きいボールは取りやすいが得点が低く、小さいボールは取りにくいが得点が高いようにしてみよう。

3. タイマーを追加して、制限時間内にどれだけ得点を稼げるかを競うゲームにしてみよう。

4. バスケットの大きさをスコアに応じて小さくしていくようにして、難易度を上げてみよう。

5. 自分だけのアイデアを加えて、オリジナルのボールキャッチゲームを作ってみよう。

第 9 章

もっと複雑なゲームを作ろう

この章では、前の章で学んだ知識を発展させて、シューティングゲームを作ります。

9.1 シューティングゲームの概要

今回作るゲームは、横スクロールシューティングゲームです。プレイヤーが操作する宇宙船で、次々と現れる敵を撃ち落としていくゲームです。

ゲームのルールは次のとおりです。

- プレイヤーは上下左右の矢印キーで宇宙船を動かします。
- スペースキーを押すと、弾を発射します。
- 敵に弾が当たると、敵が消えて得点が増えます。
- 敵と宇宙船が衝突すると、ライフが減ります。
- ライフがなくなると、ゲームオーバーになります。

9.2 ゲームの設計

シューティングゲームを作るために、次のような要素が必要です。

1. プレイヤー（宇宙船）* 位置* 移動方法* 描画方法
2. 弾* 位置* 移動方法* 描画方法
3. 敵* 位置* 移動方法* 描画方法
4. ゲームの状態* スコア* ライフ* ゲームオーバーフラグ
5. 当たり判定* 弾と敵の衝突判定* プレイヤーと敵の衝突判定

6. 効果音* 弾の発射音* 敵を倒したときの音* ダメージを受けたときの音* ゲームオーバーになったときの音

これらの要素を順番に実装していきましょう。

9.3 基本的なプログラム構造

まずは、シューティングゲームの基本的なプログラム構造を作ります。

▼リスト 9.1　基本的なプログラム構造

```
import pyxel
import random

# Pyxelを初期化
pyxel.init(160, 120, title="シューティングゲーム")

# ゲームの初期化
def init_game():
    global player_x, player_y, bullets, enemies, score, lives, game_over

    # プレイヤー（宇宙船）の位置
    player_x = 20
    player_y = 60

    # 弾のリスト（各弾は[x座標, y座標]のリスト）
    bullets = []

    # 敵のリスト（各敵は[x座標, y座標, 速度]のリスト）
    enemies = []

    # ゲームの状態
    score = 0
    lives = 3
    game_over = False

# ゲームの初期化を実行
init_game()

# 効果音の定義
# 弾の発射音
pyxel.sound(0).set(note="C5", tone="P", volume="3", effect="N", speed=5)
# 敵を倒した音
pyxel.sound(1).set(note="C4 E4 G4", tone="T", volume="5", effect="N", speed=5)
# ダメージを受けた音
pyxel.sound(2).set(note="C3", tone="S", volume="4", effect="N", speed=10)
# ゲームオーバー音
pyxel.sound(3).set(note="C3 C2", tone="S", volume="6", effect="F", speed=10)

# 画面を更新する関数
def update():
    global player_x, player_y, bullets, enemies, score, lives, game_over
```

```
# ESCキーが押されたら終了
if pyxel.btnp(pyxel.KEY_ESCAPE):
    pyxel.quit()

# ゲームオーバー時の処理
if game_over:
    # スペースキーでリスタート
    if pyxel.btnp(pyxel.KEY_SPACE):
        init_game()
    return

# プレイヤーの移動
if pyxel.btn(pyxel.KEY_LEFT):
    player_x = max(player_x - 2, 0)

if pyxel.btn(pyxel.KEY_RIGHT):
    player_x = min(player_x + 2, 160 - 8)

if pyxel.btn(pyxel.KEY_UP):
    player_y = max(player_y - 2, 0)

if pyxel.btn(pyxel.KEY_DOWN):
    player_y = min(player_y + 2, 120 - 8)

# 弾の発射
if pyxel.btnp(pyxel.KEY_SPACE):
    bullets.append([player_x + 8, player_y + 4])
    pyxel.play(0, 0)  # 弾の発射音を再生

# 弾の移動
for i in range(len(bullets) - 1, -1, -1):
    bullets[i][0] += 4  # 弾は右に移動

    # 画面外に出た弾を削除
    if bullets[i][0] > 160:
        bullets.pop(i)

# 敵の生成（一定確率で）
if random.random() < 0.05:
    enemies.append([160, random.randint(0, 112), random.uniform(1, 3)])

# 敵の移動
for i in range(len(enemies) - 1, -1, -1):
    enemies[i][0] -= enemies[i][2]  # 敵は左に移動

    # 画面外に出た敵を削除
    if enemies[i][0] < -8:
        enemies.pop(i)

# 当たり判定（弾と敵）
for i in range(len(bullets) - 1, -1, -1):
    for j in range(len(enemies) - 1, -1, -1):
        if (bullets[i][0] < enemies[j][0] + 8 and
            bullets[i][0] + 2 > enemies[j][0] and
            bullets[i][1] < enemies[j][1] + 8 and
            bullets[i][1] + 2 > enemies[j][1]):
            # 弾が敵に当たった
            bullets.pop(i)
```

```
                enemies.pop(j)
                score += 1
                pyxel.play(1, 1)  # 敵を倒した音を再生
                break

    # 当たり判定（プレイヤーと敵）
    for i in range(len(enemies) - 1, -1, -1):
        if (player_x < enemies[i][0] + 8 and
            player_x + 8 > enemies[i][0] and
            player_y < enemies[i][1] + 8 and
            player_y + 8 > enemies[i][1]):
            # プレイヤーが敵に当たった
            enemies.pop(i)
            lives -= 1
            pyxel.play(2, 2)  # ダメージを受けた音を再生

            # ライフがなくなったらゲームオーバー
            if lives <= 0:
                game_over = True
                pyxel.play(3, 3)  # ゲームオーバー音を再生
            break

# 画面を描画する関数
def draw():
    # 画面を黒色（0）でクリア
    pyxel.cls(0)

    # 背景に星を描く
    for i in range(16):
        pyxel.pset(i * 10, 20, 7)  # 白色(7)の点
        pyxel.pset(i * 10 + 5, 40, 7)  # 白色(7)の点
        pyxel.pset(i * 10, 60, 7)  # 白色(7)の点
        pyxel.pset(i * 10 + 5, 80, 7)  # 白色(7)の点
        pyxel.pset(i * 10, 100, 7)  # 白色(7)の点

    # ゲームオーバーでなければ、ゲーム要素を描画
    if not game_over:
        # プレイヤー（宇宙船）を描画
        pyxel.rect(player_x, player_y, 8, 8, 11)  # 黄緑色(11)の四角形
        pyxel.tri(player_x, player_y, player_x, player_y + 8, player_x - 4, player_y + 4, 11)  # 先
端

        # 弾を描画
        for bullet_x, bullet_y in bullets:
            pyxel.rect(bullet_x, bullet_y, 2, 2, 10)  # 黄色(10)の四角形

        # 敵を描画
        for enemy_x, enemy_y, _ in enemies:
            pyxel.rect(enemy_x, enemy_y, 8, 8, 8)  # 赤色(8)の四角形

    # スコアとライフを表示
    pyxel.text(5, 5, f"Score: {score}", 7)
    pyxel.text(5, 15, f"Lives: {lives}", 7)

    # ゲームオーバー時のメッセージ
    if game_over:
        pyxel.text(55, 50, "GAME OVER", 8)
```

```
        pyxel.text(35, 70, "Press SPACE to restart", 7)

# Pyxelの実行
pyxel.run(update, draw)
```

このプログラムをファイル名「shooting_game.py」として保存し、実行してみましょう。矢印キーで宇宙船を動かし、スペースキーで弾を発射して敵を倒します。敵と衝突するとライフが減り、ライフがなくなるとゲームオーバーになります。スペースキーでリスタートできます。

9.4 プログラムの解説

シューティングゲームのプログラムを詳しく見ていきましょう。

9.4.1 ゲームの初期化

▼リスト 9.2 ゲームの初期化

```
# ゲームの初期化
def init_game():
    global player_x, player_y, bullets, enemies, score, lives, game_over

    # プレイヤー（宇宙船）の位置
    player_x = 20
    player_y = 60

    # 弾のリスト（各弾は[x座標，y座標]のリスト）
    bullets = []

    # 敵のリスト（各敵は[x座標，y座標，速度]のリスト）
    enemies = []

    # ゲームの状態
    score = 0
    lives = 3
    game_over = False
```

‘init_game‘関数では、ゲームの状態を初期化しています。

- ‘player_x‘、‘player_y‘：プレイヤー（宇宙船）の座標
- ‘bullets‘：弾のリスト（各弾は ‘[x 座標, y 座標]‘のリスト）
- ‘enemies‘：敵のリスト（各敵は ‘[x 座標, y 座標, 速度]‘のリスト）
- ‘score‘：スコア

- ‘lives‘：ライフ数
- ‘game_over‘：ゲームオーバーフラグ

9.4.2 効果音の定義

▼リスト 9.3　効果音の定義

```
# 効果音の定義
# 弾の発射音
pyxel.sound(0).set(note="C5", tone="P", volume="3", effect="N", speed=5)
# 敵を倒した音
pyxel.sound(1).set(note="C4 E4 G4", tone="T", volume="5", effect="N", speed=5)
# ダメージを受けた音
pyxel.sound(2).set(note="C3", tone="S", volume="4", effect="N", speed=10)
# ゲームオーバー音
pyxel.sound(3).set(note="C3 C2", tone="S", volume="6", effect="F", speed=10)
```

4 つの効果音を定義しています。

- サウンド 0：弾の発射音（高い単音）
- サウンド 1：敵を倒した音（明るい上昇音）
- サウンド 2：ダメージを受けた音（低い単音）
- サウンド 3：ゲームオーバー音（暗い下降音）

9.4.3 update 関数

‘update‘関数では、ゲームの状態を更新します。

プレイヤーの移動と弾の発射

▼リスト 9.4　プレイヤーの移動と弾の発射

```
# プレイヤーの移動
if pyxel.btn(pyxel.KEY_LEFT):
    player_x = max(player_x - 2, 0)

if pyxel.btn(pyxel.KEY_RIGHT):
    player_x = min(player_x + 2, 160 - 8)

if pyxel.btn(pyxel.KEY_UP):
    player_y = max(player_y - 2, 0)
```

```
if pyxel.btn(pyxel.KEY_DOWN):
    player_y = min(player_y + 2, 120 - 8)

# 弾の発射
if pyxel.btnp(pyxel.KEY_SPACE):
    bullets.append([player_x + 8, player_y + 4])
    pyxel.play(0, 0)  # 弾の発射音を再生
```

上下左右の矢印キーでプレイヤー（宇宙船）を移動します。‘max‘と ‘min‘関数を使って、画面の端を超えないようにしています。

スペースキーを押すと、弾を発射します。弾の初期位置は、宇宙船の先端（‘player_x + 8, player_y + 4‘）に設定されます。

弾の移動

▼リスト 9.5　弾の移動

```
# 弾の移動
for i in range(len(bullets) - 1, -1, -1):
    bullets[i][0] += 4  # 弾は右に移動

    # 画面外に出た弾を削除
    if bullets[i][0] > 160:
        bullets.pop(i)
```

弾は右に移動します（X 座標が増加）。画面外に出た弾は削除します。

リストの要素を削除する際は、後ろから前に処理することで、インデックスのずれを防いでいます。

敵の生成と移動

▼リスト 9.6　敵の生成と移動

```
# 敵の生成（一定確率で）
if random.random() < 0.05:
    enemies.append([160, random.randint(0, 112), random.uniform(1, 3)])

# 敵の移動
for i in range(len(enemies) - 1, -1, -1):
    enemies[i][0] -= enemies[i][2]  # 敵は左に移動

    # 画面外に出た敵を削除
    if enemies[i][0] < -8:
```

```
        enemies.pop(i)
```

敵は一定確率（5%）で画面右端に生成されます。Y 座標はランダムに設定され、速度も 1〜3 の間でランダムに設定されます。

敵は左に移動します（X 座標が減少）。画面外に出た敵は削除します。

当たり判定

▼リスト 9.7　当たり判定

```
# 当たり判定（弾と敵）
for i in range(len(bullets) - 1, -1, -1):
    for j in range(len(enemies) - 1, -1, -1):
        if (bullets[i][0] < enemies[j][0] + 8 and
            bullets[i][0] + 2 > enemies[j][0] and
            bullets[i][1] < enemies[j][1] + 8 and
            bullets[i][1] + 2 > enemies[j][1]):
            # 弾が敵に当たった
            bullets.pop(i)
            enemies.pop(j)
            score += 1
            pyxel.play(1, 1)  # 敵を倒した音を再生
            break

# 当たり判定（プレイヤーと敵）
for i in range(len(enemies) - 1, -1, -1):
    if (player_x < enemies[i][0] + 8 and
        player_x + 8 > enemies[i][0] and
        player_y < enemies[i][1] + 8 and
        player_y + 8 > enemies[i][1]):
        # プレイヤーが敵に当たった
        enemies.pop(i)
        lives -= 1
        pyxel.play(2, 2)  # ダメージを受けた音を再生

        # ライフがなくなったらゲームオーバー
        if lives <= 0:
            game_over = True
            pyxel.play(3, 3)  # ゲームオーバー音を再生
        break
```

当たり判定は、2 つのオブジェクトの矩形が重なっているかどうかをチェックします。

弾と敵の当たり判定では、弾が敵に当たった場合、弾と敵を削除し、スコアを増やします。

プレイヤーと敵の当たり判定では、プレイヤーが敵に当たった場合、敵を削除し、ライフを減らします。ライフがなくなると、ゲームオーバーになります。

9.5 ゲームの改良

基本的なシューティングゲームができたら、さらに改良してみましょう。ここでは、いくつかの改良案を紹介します。

9.5.1 敵の種類を増やす

敵の種類を増やして、ゲームをより面白くしましょう。

▼リスト 9.8 敵の種類を増やす

```
# 敵の生成（一定確率で）
if random.random() < 0.05:
    # 敵の種類をランダムに決定（0: 通常の敵, 1: 速い敵, 2: 大きい敵）
    enemy_type = random.randint(0, 2)

    if enemy_type == 0:
        # 通常の敵
        enemies.append([160, random.randint(0, 112), random.uniform(1, 2), 8, 8, 8, 1])
    elif enemy_type == 1:
        # 速い敵
        enemies.append([160, random.randint(0, 112), random.uniform(2, 4), 6, 6, 9, 1])
    else:
        # 大きい敵
        enemies.append([160, random.randint(0, 104), random.uniform(0.5, 1), 16, 16, 14, 3])

# 敵の描画
for enemy in enemies:
    enemy_x, enemy_y, _, width, height, color, _ = enemy
    pyxel.rect(enemy_x, enemy_y, width, height, color)
```

敵のリストの各要素を ‘[x 座標, y 座標, 速度, 幅, 高さ, 色, 耐久力]‘のリストに変更しています。敵の種類によって、速度、大きさ、色、耐久力が異なります。

9.5.2 パワーアップアイテム

敵を倒すと、一定確率でパワーアップアイテムが出現するようにしましょう。

▼リスト 9.9 パワーアップアイテム

```
# ゲームの初期化
def init_game():
    global player_x, player_y, bullets, enemies, items, score, lives, power_level, game_over
```

```
    # プレイヤー（宇宙船）の位置
    player_x = 20
    player_y = 60

    # 弾のリスト
    bullets = []

    # 敵のリスト
    enemies = []

    # アイテムのリスト（各アイテムは[x座標，y座標，種類]のリスト）
    items = []

    # ゲームの状態
    score = 0
    lives = 3
    power_level = 0  # パワーレベル（0: 通常，1: 2連射，2: 3連射）
    game_over = False

# 弾の発射
if pyxel.btnp(pyxel.KEY_SPACE):
    if power_level == 0:
        # 通常の弾
        bullets.append([player_x + 8, player_y + 4])
    elif power_level == 1:
        # 2連射
        bullets.append([player_x + 8, player_y + 2])
        bullets.append([player_x + 8, player_y + 6])
    else:
        # 3連射
        bullets.append([player_x + 8, player_y])
        bullets.append([player_x + 8, player_y + 4])
        bullets.append([player_x + 8, player_y + 8])

    pyxel.play(0, 0)  # 弾の発射音を再生
```

パワーアップアイテムを取得すると、弾の発射数が増えます（1 発→ 2 発→ 3 発）。

9.5.3 ボス敵

一定スコアに達すると、ボス敵が出現するようにしましょう。

▼リスト 9.10　ボス敵

```
# ゲームの初期化
def init_game():
    global player_x, player_y, bullets, enemies, items, boss, boss_hp, score, lives, power_level, game_over, b

    # プレイヤー（宇宙船）の位置
    player_x = 20
    player_y = 60
```

```
    # 弾のリスト
    bullets = []

    # 敵のリスト
    enemies = []

    # アイテムのリスト
    items = []

    # ボス敵
    boss = None  # ボスがいない状態
    boss_hp = 0  # ボスの体力

    # ゲームの状態
    score = 0
    lives = 3
    power_level = 0
    game_over = False
    boss_mode = False  # ボスモードフラグ

# update関数内
# ボスモードの切り替え
if not boss_mode and score >= 20:  # スコアが20以上でボスモードに
    boss_mode = True
    enemies = []  # 通常の敵をクリア
    boss = [120, 60, 1]  # ボスの初期位置と移動方向（1: 下, -1: 上）
    boss_hp = 20  # ボスの体力

# ボスの移動と攻撃
if boss_mode and boss:
    # ボスの移動（上下に動く）
    boss[1] += boss[2]
    if boss[1] < 10 or boss[1] > 90:
        boss[2] = -boss[2]  # 方向転換

    # ボスの攻撃（一定確率で弾を発射）
    if random.random() < 0.1:
        enemies.append([boss[0] - 8, boss[1] + 10, 2, 8, 8, 8, 1])  # ボスから弾
が出る

    # プレイヤーの弾とボスの当たり判定
    for i in range(len(bullets) - 1, -1, -1):
        if (bullets[i][0] < boss[0] + 20 and
            bullets[i][0] + 2 > boss[0] - 20 and
            bullets[i][1] < boss[1] + 20 and
            bullets[i][1] + 2 > boss[1] - 20):
            # 弾がボスに当たった
            bullets.pop(i)
            boss_hp -= 1
            pyxel.play(1, 1)  # ヒット音を再生

            # ボスを倒した
            if boss_hp <= 0:
                boss = None
                score += 30  # ボーナス得点
                pyxel.play(1, 1)  # 敵を倒した音を再生
                boss_mode = False  # ボスモード終了
            break
```

```
# ボスの描画
if boss_mode and boss:
    # ボスを描画
    pyxel.rect(boss[0] - 20, boss[1] - 20, 40, 40, 8)  # 赤色(8)の大きな四角形
    pyxel.text(boss[0] - 10, boss[1], f"HP: {boss_hp}", 7)  # 体力表示
```

スコアが一定値（ここでは 20）に達すると、ボスモードに切り替わります。ボスは大きな敵で、体力が多く、弾を発射してきます。ボスを倒すとボーナス得点が入り、通常モードに戻ります。

9.6 まとめ

この章では、シューティングゲームを作りました。

- プレイヤー（宇宙船）の移動と弾の発射
- 敵の生成と移動
- 当たり判定（弾と敵、プレイヤーと敵）
- スコアとライフの管理
- ゲームオーバーの処理
- 効果音の追加
- ゲームの改良（敵の種類、パワーアップアイテム、ボス敵）

これらの要素は、多くのシューティングゲームに共通する基本的な要素です。この知識を応用すれば、さまざまなシューティングゲームを作ることができます。

次の章では、これまで学んだ知識を活かして、自分だけのオリジナルゲームを作る方法を学びます。

9.7 チャレンジ問題

1. 敵の動きをもっと複雑にしてみよう。例えば、ジグザグに動いたり、プレイヤーを追いかけたりする敵を作ってみよう。

2. ボスの攻撃パターンを増やしてみよう。例えば、複数の弾を一度に発射したり、レーザービームを発射したりするボスを作ってみよう。

3. ステージ制を導入してみよう。一定スコアごとに背景や敵の種類が変わるようにしてみよう。

4. 2 人プレイモードを追加してみよう。2 人目のプレイヤーは WASD キーで操作し、F キーで弾を発射するようにしてみよう。

5. 自分だけのアイデアを加えて、オリジナルのシューティングゲームを作ってみよう。

第 10 章

自分だけのゲームを作ろう

この章では、これまで学んだ知識を活かして、自分だけのオリジナルゲームを作る方法を学びます。

10.1　オリジナルゲームを作るための手順

これまでの章で、Pyxel を使ったゲーム開発の基本を学んできました。図形の描画、キャラクターの移動、音の再生、当たり判定など、ゲーム開発に必要な要素を一通り学びました。

これらの知識を活かして、自分だけのオリジナルゲームを作ってみましょう。オリジナルゲームを作るには、次のような手順で進めるとよいでしょう。

1. ゲームのアイデアを考える
2. ゲームの設計をする
3. プログラムを書く
4. テストと改良を繰り返す
5. 完成したゲームを友だちに遊んでもらう

それぞれの手順について、詳しく見ていきましょう。

10.2　ゲームのアイデアを考える

まずは、どんなゲームを作りたいか、アイデアを考えましょう。アイデアを考えるときのポイントは次のとおりです。

10.2.1　好きなゲームを参考にする

自分が好きなゲームを参考にするのは、アイデアを考えるための良い方法です。例えば、次のようなゲームが考えられます。

- アクションゲーム：キャラクターを操作して障害物を避けたり、敵を倒したりするゲーム
- パズルゲーム：ブロックを並べたり、消したりして得点を競うゲーム
- レースゲーム：車や乗り物を操作してコースを走るゲーム
- 収集ゲーム：アイテムを集めて得点を競うゲーム
- 迷路ゲーム：迷路を探索して出口を目指すゲーム

10.2.2　簡単なルールから始める

初めてオリジナルゲームを作るときは、シンプルなルールから始めるのがおすすめです。複雑なルールは、プログラミングも難しくなります。

例えば、「画面上のアイテムをすべて集める」「制限時間内に敵を倒す」「障害物を避けてゴールを目指す」など、シンプルなルールのゲームから始めましょう。

10.2.3　自分の好きな要素を入れる

自分が好きなキャラクターや世界観、音楽などを取り入れると、作るのが楽しくなります。例えば、好きな動物をキャラクターにしたり、好きな場所を背景にしたりしてみましょう。

10.3　ゲームの設計をする

アイデアが決まったら、ゲームの設計をしましょう。設計とは、ゲームの詳細を決めることです。

10.3.1　ゲームの目的とルールを決める

ゲームの目的（何をすればクリアか）とルール（どうやってプレイするか）を明確にしましょう。

例えば、「制限時間内にすべてのコインを集める」というゲームなら、次のように決め

ます。

- 目的：すべてのコインを集める
- ルール：
- 矢印キーでキャラクターを操作する
- コインに触れるとコインが消えて得点が増える
- 敵に触れるとライフが減る
- ライフがなくなるか、制限時間が過ぎるとゲームオーバー

10.3.2 画面のレイアウトを考える

ゲーム画面のレイアウトを考えましょう。スコア、ライフ、時間などの表示位置や、キャラクターや背景の配置を決めます。

紙に絵を描いたり、図を作ったりして、画面のイメージを作るとよいでしょう。

10.3.3 必要な変数とデータ構造を考える

ゲームに必要な変数（データを記録するもの）とデータ構造（データの保存方法）を考えましょう。

例えば、「コイン集めゲーム」なら、次のような変数とデータ構造が必要です。

- プレイヤーの位置（‘player_x‘、‘player_y‘）
- コインのリスト（各コインの位置を記録）
- 敵のリスト（各敵の位置と移動方向を記録）
- スコア（‘score‘）
- ライフ（‘lives‘）
- 制限時間（‘time_left‘）
- ゲームの状態（‘game_over‘）

10.3.4 関数の役割を考える

ゲームの各部分を担当する関数（処理のまとまり）を考えましょう。

基本的には、次のような関数が必要です。

- ‘init_game‘：ゲームの初期化
- ‘update‘：ゲームの状態を更新

- ‘draw‘：画面を描画

さらに、必要に応じて次のような関数を追加します。

- ‘move_player‘：プレイヤーの移動
- ‘move_enemies‘：敵の移動
- ‘check_collisions‘：当たり判定
- ‘update_time‘：時間の更新
- ‘draw_ui‘：UI（ユーザーインターフェース）の描画

10.4　プログラムを書く

設計ができたら、実際にプログラムを書いていきましょう。

10.4.1　基本的な構造から始める

まずは、Pyxel の基本的な構造から始めましょう。

▼リスト 10.1　基本的な構造

```
import pyxel

# Pyxelを初期化
pyxel.init(160, 120, title="オリジナルゲーム")

# ゲームの初期化
def init_game():
    global player_x, player_y, score, lives, game_over

    # プレイヤーの位置
    player_x = 80
    player_y = 60

    # ゲームの状態
    score = 0
    lives = 3
    game_over = False

# ゲームの初期化を実行
init_game()

# 画面を更新する関数
def update():
    global player_x, player_y, score, lives, game_over

    # ESCキーが押されたら終了
    if pyxel.btnp(pyxel.KEY_ESCAPE):
```

```
        pyxel.quit()

    # ゲームオーバー時の処理
    if game_over:
        # スペースキーでリスタート
        if pyxel.btnp(pyxel.KEY_SPACE):
            init_game()
        return

    # プレイヤーの移動
    if pyxel.btn(pyxel.KEY_LEFT):
        player_x = max(player_x - 2, 0)

    if pyxel.btn(pyxel.KEY_RIGHT):
        player_x = min(player_x + 2, 160 - 8)

    if pyxel.btn(pyxel.KEY_UP):
        player_y = max(player_y - 2, 0)

    if pyxel.btn(pyxel.KEY_DOWN):
        player_y = min(player_y + 2, 120 - 8)

# 画面を描画する関数
def draw():
    # 画面を黒色（0）でクリア
    pyxel.cls(0)

    # プレイヤーを描画
    pyxel.rect(player_x, player_y, 8, 8, 11)

    # スコアとライフを表示
    pyxel.text(5, 5, f"Score: {score}", 7)
    pyxel.text(5, 15, f"Lives: {lives}", 7)

    # ゲームオーバー時のメッセージ
    if game_over:
        pyxel.text(55, 50, "GAME OVER", 8)
        pyxel.text(35, 70, "Press SPACE to restart", 7)

# Pyxelの実行
pyxel.run(update, draw)
```

この基本的な構造に、自分のゲームに必要な要素を追加していきましょう。

10.4.2 少しずつ機能を追加する

一度にすべての機能を実装しようとせず、少しずつ機能を追加していきましょう。例えば、次のような順序で実装するとよいでしょう。

1. プレイヤーの移動
2. 背景の描画
3. アイテムや敵の追加

4. 当たり判定
5. スコアや時間の管理
6. ゲームオーバーの処理
7. 効果音の追加

各機能を追加するたびに、プログラムを実行してテストしましょう。

10.5　テストと改良を繰り返す

プログラムを書いたら、実際に動かしてテストしましょう。テストでは、次のようなことをチェックします。

- ゲームが正しく動作するか
- バグ（プログラムの不具合）がないか
- ゲームが面白いか
- 難易度は適切か

テストで見つかった問題点を修正し、改良を加えていきましょう。例えば、次のような改良が考えられます。

- 難易度の調整（敵の速度、アイテムの数など）
- 視覚効果の追加（パーティクル、アニメーションなど）
- 音響効果の追加（BGM、効果音など）
- ゲームモードの追加（難易度選択、2 人プレイなど）

10.6　オリジナルゲームの例

ここでは、オリジナルゲームの例として、「宝石集めゲーム」を紹介します。このゲームは、制限時間内にできるだけ多くの宝石を集めるゲームです。

▼リスト 10.2　宝石集めゲーム

```
import pyxel
import random

# Pyxelを初期化
pyxel.init(160, 120, title="宝石集めゲーム")

# ゲームの初期化
```

```
def init_game():
    global player_x, player_y, jewels, enemies, score, lives, time_left, game_over

    # プレイヤーの位置
    player_x = 80
    player_y = 60

    # 宝石のリスト（各宝石は[x座標, y座標, 色]のリスト）
    jewels = []
    for _ in range(10):
        jewels.append([random.randint(10, 150), random.randint(10, 110), random.randint(8, 15)])

    # 敵のリスト（各敵は[x座標, y座標, 速度X, 速度Y]のリスト）
    enemies = []
    for _ in range(5):
        enemies.append([
            random.randint(10, 150),
            random.randint(10, 110),
            random.choice([-1, 1]) * random.uniform(0.5, 1.5),
            random.choice([-1, 1]) * random.uniform(0.5, 1.5)
        ])

    # ゲームの状態
    score = 0
    lives = 3
    time_left = 30 * 30  # 30秒（30フレーム/秒）
    game_over = False

# ゲームの初期化を実行
init_game()

# 効果音の定義
# 宝石を取得した音
pyxel.sound(0).set(note="C4 E4 G4", tone="T", volume="7", effect="N", speed=5)
# ダメージを受けた音
pyxel.sound(1).set(note="C3", tone="S", volume="4", effect="N", speed=10)
# ゲームオーバー音
pyxel.sound(2).set(note="C3 C2", tone="S", volume="6", effect="F", speed=10)

# 画面を更新する関数
def update():
    global player_x, player_y, jewels, enemies, score, lives, time_left, game_over

    # ESCキーが押されたら終了
    if pyxel.btnp(pyxel.KEY_ESCAPE):
        pyxel.quit()

    # ゲームオーバー時の処理
    if game_over:
        # スペースキーでリスタート
        if pyxel.btnp(pyxel.KEY_SPACE):
            init_game()
        return

    # 時間の更新
    time_left -= 1
    if time_left <= 0:
        game_over = True
```

```
            pyxel.play(2, 2)  # ゲームオーバー音を再生
            return

        # プレイヤーの移動
        if pyxel.btn(pyxel.KEY_LEFT):
            player_x = max(player_x - 2, 0)

        if pyxel.btn(pyxel.KEY_RIGHT):
            player_x = min(player_x + 2, 160 - 8)

        if pyxel.btn(pyxel.KEY_UP):
            player_y = max(player_y - 2, 0)

        if pyxel.btn(pyxel.KEY_DOWN):
            player_y = min(player_y + 2, 120 - 8)

        # 敵の移動
        for i in range(len(enemies)):
            enemies[i][0] += enemies[i][2]
            enemies[i][1] += enemies[i][3]

            # 画面の端での反射
            if enemies[i][0] < 0 or enemies[i][0] > 160 - 8:
                enemies[i][2] = -enemies[i][2]

            if enemies[i][1] < 0 or enemies[i][1] > 120 - 8:
                enemies[i][3] = -enemies[i][3]

        # 宝石との当たり判定
        for i in range(len(jewels) - 1, -1, -1):
            if (player_x < jewels[i][0] + 6 and
                player_x + 8 > jewels[i][0] and
                player_y < jewels[i][1] + 6 and
                player_y + 8 > jewels[i][1]):
                # 宝石を取得
                jewels.pop(i)
                score += 1
                pyxel.play(0, 0)  # 宝石取得音を再生

                # すべての宝石を集めたら、新しい宝石を追加
                if len(jewels) == 0:
                    for _ in range(10):
                        jewels.append([random.randint(10, 150), random.randint(10, 110), random.randint(8, 15)])

        # 敵との当たり判定
        for enemy in enemies:
            if (player_x < enemy[0] + 8 and
                player_x + 8 > enemy[0] and
                player_y < enemy[1] + 8 and
                player_y + 8 > enemy[1]):
                # ダメージを受ける
                lives -= 1
                pyxel.play(1, 1)  # ダメージ音を再生

                # プレイヤーを初期位置に戻す
                player_x = 80
                player_y = 60
```

```
            # ライフがなくなったらゲームオーバー
            if lives <= 0:
                game_over = True
                pyxel.play(2, 2)  # ゲームオーバー音を再生

            break

# 画面を描画する関数
def draw():
    # 画面を暗い青色 (1) でクリア
    pyxel.cls(1)

    # 背景に星を描く
    for i in range(16):
        pyxel.pset(i * 10, 20, 7)
        pyxel.pset(i * 10 + 5, 40, 7)
        pyxel.pset(i * 10, 60, 7)
        pyxel.pset(i * 10 + 5, 80, 7)
        pyxel.pset(i * 10, 100, 7)

    # 宝石を描画
    for jewel_x, jewel_y, jewel_color in jewels:
        pyxel.rect(jewel_x, jewel_y, 6, 6, jewel_color)
        pyxel.pset(jewel_x + 2, jewel_y + 2, 7)  # 光る部分

    # 敵を描画
    for enemy_x, enemy_y, _, _ in enemies:
        pyxel.rect(enemy_x, enemy_y, 8, 8, 8)  # 赤色(8)の四角形

    # プレイヤーを描画
    pyxel.rect(player_x, player_y, 8, 8, 11)  # 黄緑色(11)の四角形

    # スコア、ライフ、時間を表示
    pyxel.text(5, 5, f"Score: {score}", 7)
    pyxel.text(5, 15, f"Lives: {lives}", 7)
    pyxel.text(5, 25, f"Time: {time_left // 30}", 7)

    # ゲームオーバー時のメッセージ
    if game_over:
        pyxel.text(55, 50, "GAME OVER", 8)
        pyxel.text(45, 70, f"Final Score: {score}", 7)
        pyxel.text(35, 90, "Press SPACE to restart", 7)

# Pyxelの実行
pyxel.run(update, draw)
```

このゲームでは、プレイヤーが宝石を集めながら、敵を避けるというシンプルなルールになっています。制限時間内にできるだけ多くの宝石を集めることが目標です。

10.7　完成したゲームを友だちに遊んでもらう

オリジナルゲームが完成したら、友だちや家族に遊んでもらいましょう。他の人に遊んでもらうことで、次のようなことがわかります。

- ゲームのルールはわかりやすいか
- 操作は簡単か
- 難易度は適切か
- バグはないか
- 面白いと感じてもらえるか

友だちからのフィードバック（感想や意見）を参考に、さらにゲームを改良していきましょう。

10.8 まとめ

この章では、オリジナルゲームを作るための手順を学びました。

- ゲームのアイデアを考える
- ゲームの設計をする
- プログラムを書く
- テストと改良を繰り返す
- 完成したゲームを友だちに遊んでもらう

これまでの章で学んだ知識を活かして、自分だけのオリジナルゲームを作ってみましょう。プログラミングの楽しさは、自分のアイデアを形にできることです。

次の章では、よくあるエラーと解決方法について学びます。

10.9 チャレンジ問題

1. 「宝石集めゲーム」に新しい要素を追加してみよう。例えば、特別な宝石（得点が高い）、パワーアップアイテム（一時的に無敵になる）、ワープゾーン（別の場所に移動する）などを追加してみよう。

2. 自分の好きなキャラクターや世界観を取り入れたオリジナルゲームを考えてみよう。どんなゲームにしたいか、紙に絵を描いたり、文章にまとめたりしてみよう。

3. 友だちと一緒にゲームのアイデアを出し合って、協力してゲームを作ってみよう。一人が描画を担当し、もう一人がプログラミングを担当するなど、役割分担をしてみよう。

4. これまでの章で作ったゲーム（ボールキャッチゲーム、シューティングゲーム）をベースに、新しいルールやキャラクターを追加して、オリジナルゲームにアレンジしてみよう。

5. 自分だけのオリジナルゲームを作って、友だちや家族に遊んでもらおう。感想や意

見をもらって、さらに改良してみよう。

付録 A

よくあるエラーと解決方法

この章では、Python プログラミングでよく発生するエラーとその解決方法を学びます。

A.1　プログラミングとエラー

プログラミングをしていると、エラー（間違い）が発生することがあります。エラーが発生すると、プログラムが正しく動かなかったり、途中で止まったりします。

エラーは、プログラミングの学習過程では避けられないものです。大切なのは、エラーが発生したときに、その原因を理解して解決する方法を知っておくことです。

この章では、Python プログラミングでよく発生するエラーと、その解決方法を紹介します。

A.2　エラーメッセージを読む

Python でエラーが発生すると、「エラーメッセージ」が表示されます。エラーメッセージには、エラーの種類や発生した場所などの情報が含まれています。

例えば、次のようなプログラムを実行すると、エラーが発生します。

▼リスト A.1　エラーの例

```
# 変数xに10を代入
x = 10

# 変数yに20を代入（ただし、スペルミスがある）
y = 20
```

```
# 変数xと変数zを足し算（変数zは存在しない）
result = x + z

# 結果を表示
print(result)
```

このプログラムを実行すると、次のようなエラーメッセージが表示されます。

```
Traceback (most recent call last):
  File "error_example.py", line 8, in <module>
    result = x + z
NameError: name 'z' is not defined
```

このエラーメッセージを読み解くと、次のことがわかります。

- エラーが発生したファイル：‘error_example.py‘
- エラーが発生した行番号：8 行目
- エラーが発生したコード：‘result = x + z‘
- エラーの種類：‘NameError‘（名前エラー）
- エラーの詳細：‘name ’z’ is not defined‘（変数 ‘z‘が定義されていない）

エラーメッセージを読むことで、エラーの原因を特定し、解決することができます。

A.3 よくあるエラーの種類と解決方法

Python プログラミングでよく発生するエラーの種類と、その解決方法を見ていきましょう。

A.3.1 SyntaxError（構文エラー）

SyntaxError は、Python の文法（構文）に違反しているときに発生します。例えば、かっこの閉じ忘れ、コロン（:）の忘れ、インデント（字下げ）の間違いなどが原因です。

▼リスト A.2　SyntaxError の例

```
# かっこの閉じ忘れ
print("Hello, World!"

# コロンの忘れ
if x > 10
```

```
    print("xは10より大きい")

# インデントの間違い
if x > 10:
print("xは10より大きい")
```

解決方法

- エラーメッセージで指摘された行を確認する
- かっこ、コロン、インデントなどが正しいか確認する
- コードを整形して、見やすくする

A.3.2 NameError（名前エラー）

NameError は、存在しない変数や関数を使おうとしたときに発生します。変数名のスペルミスや、変数を使う前に定義していないことが原因です。

▼リスト A.3　NameError の例

```
# 存在しない変数を使う
result = x + 10  # 変数xが定義されていない

# 変数名のスペルミス
count = 5
print(Count)  # 大文字と小文字は区別される
```

解決方法

- 変数が正しく定義されているか確認する
- 変数名のスペルが正しいか確認する（大文字と小文字も区別される）
- 変数を使う前に、必ず定義する

A.3.3 TypeError（型エラー）

TypeError は、異なる型（文字列、数値など）の値を不適切に組み合わせたときに発生します。例えば、文字列と数値を足し算しようとしたときなどです。

▼リスト A.4　TypeError の例

```
# 文字列と数値の足し算
result = "Hello" + 10  # 文字列と数値は足せない

# 数値でない値に対する計算
text = "Hello"
length = text / 2  # 文字列は割り算できない
```

解決方法

- 変数の型を確認する（‘print(type(変数名))‘で型を表示できる）
- 必要に応じて型変換を行う（‘str()‘で文字列に、‘int()‘で整数に変換）
- 演算子が適切か確認する

A.3.4 IndexError（インデックスエラー）

IndexError は、リスト（配列）の範囲外のインデックス（添え字）にアクセスしようとしたときに発生します。

▼リスト A.5　IndexError の例


```
# リストの範囲外のインデックスにアクセス
fruits = ["りんご", "バナナ", "オレンジ"]
print(fruits[3])  # インデックスは0から始まるので、3は範囲外
```

解決方法

- リストのサイズを確認する（‘len(リスト名)‘でサイズを取得できる）
- インデックスは 0 から始まることを意識する
- リストの範囲内のインデックスを使う

A.3.5 IndentationError（インデントエラー）

IndentationError は、インデント（字下げ）が不適切なときに発生します。Python では、インデントがコードのブロックを示す重要な要素です。

▼リスト A.6　IndentationError の例

```
# インデントが不足している
if x > 10:
print("xは10より大きい")  # インデントが必要

# インデントが一貫していない
if x > 10:
    print("xは10より大きい")
  print("これはエラー")  # インデントが一貫していない
```

解決方法

- インデントを適切に設定する（通常は 4 つのスペース）
- 同じブロック内のコードは、同じレベルのインデントにする
- タブとスペースを混在させない

A.3.6 FileNotFoundError（ファイル未検出エラー）

FileNotFoundError は、存在しないファイルを開こうとしたときに発生します。

▼リスト A.7　FileNotFoundError の例

```
# 存在しないファイルを開く
file = open("存在しないファイル.txt", "r")
```

解決方法

- ファイル名が正しいか確認する
- ファイルのパス（場所）が正しいか確認する
- ファイルが実際に存在するか確認する

A.3.7 ZeroDivisionError（ゼロ除算エラー）

ZeroDivisionError は、0 で割り算をしようとしたときに発生します。

▼リスト A.8　ZeroDivisionError の例

```
# 0で割り算
result = 10 / 0  # 0で割ることはできない
```

解決方法

- 除数（割る数）が 0 でないことを確認する
- 除数が 0 になる可能性がある場合は、条件分岐で対処する

A.4 Pyxel でよくあるエラーと解決方法

Pyxel を使ったゲーム開発でも、さまざまなエラーが発生することがあります。ここでは、Pyxel でよく発生するエラーと解決方法を紹介します。

A.4.1 ModuleNotFoundError（モジュール未検出エラー）

ModuleNotFoundError は、必要なモジュール（ライブラリ）がインストールされていないときに発生します。

▼リスト A.9　ModuleNotFoundError の例

```
# Pyxelモジュールのインポート
import pyxel  # Pyxelがインストールされていないとエラー
```


解決方法

- 必要なモジュールをインストールする（Pyxel の場合は ‘pip install pyxel‘）
- モジュール名のスペルが正しいか確認する
- Python の環境が正しく設定されているか確認する

A.4.2 AttributeError（属性エラー）

AttributeError は、オブジェクトにない属性（プロパティやメソッド）にアクセスしようとしたときに発生します。

▼リスト A.10　AttributeError の例

```
# 存在しないメソッドの呼び出し
pyxel.draw_circle(50, 50, 10, 7)  # 正しくはpyxel.circ()
```

解決方法

- Pyxel の公式ドキュメントで正しいメソッド名を確認する
- メソッド名のスペルが正しいか確認する
- 必要なオブジェクトが正しく初期化されているか確認する

A.4.3 画面が表示されない

Pyxel のプログラムを実行しても、画面が表示されないことがあります。これは、‘pyxel.run()‘が呼び出されていないか、‘update‘関数や ‘draw‘関数が正しく定義されていないことが原因です。

▼リスト A.11　画面が表示されない例

```
import pyxel

pyxel.init(160, 120)

# update関数とdraw関数を定義していない

# pyxel.run()を呼び出していない
```

解決方法

- ‘update‘関数と ‘draw‘関数を正しく定義する
- ‘pyxel.run(update, draw)‘を呼び出す
- ‘pyxel.init()‘が正しく呼び出されているか確認する

A.4.4 キー入力が反応しない

キー入力が反応しないことがあります。これは、キー入力の検出方法が間違っているか、‘update‘関数内でキー入力の処理が行われていないことが原因です。

▼リスト A.12　キー入力が反応しない例

```
def update():
    # キー入力の検出が間違っている
    if pyxel.key(pyxel.KEY_SPACE):  # 正しくはpyxel.btn()またはpyxel.btnp()
        # 処理
```

解決方法

- ‘pyxel.btn()‘（キーが押されている間）または ‘pyxel.btnp()‘（キーが押された瞬間）を使う
- キーコード（‘pyxel.KEY_SPACE‘など）が正しいか確認する
- ‘update‘関数内でキー入力の処理が行われているか確認する

A.4.5　当たり判定が機能しない

当たり判定が正しく機能しないことがあります。これは、当たり判定の条件が間違っているか、オブジェクトの位置や大きさが正しく設定されていないことが原因です。

▼リスト A.13　当たり判定が機能しない例

```
# 当たり判定の条件が間違っている
if player_x == enemy_x and player_y == enemy_y:  # 完全に一致する必要はない
    # 当たった時の処理
```

解決方法

- 当たり判定の条件を正しく設定する（範囲の重なりをチェック）
- オブジェクトの位置と大きさを確認する
- デバッグ用の表示を追加して、位置や当たり判定の状態を確認する

A.5　デバッグの方法

エラーを解決するためには、「デバッグ」という作業が重要です。デバッグとは、プログラムの問題を見つけて修正することです。

A.5.1　print 文を使ったデバッグ

最も簡単なデバッグ方法は、‘print‘文を使って変数の値や処理の流れを確認することです。

▼リスト A.14　print 文を使ったデバッグ

```
# 変数の値を確認
x = 10
y = 20
print(f"x = {x}, y = {y}")

# 処理の流れを確認
print("処理1を実行")
# 処理1のコード
print("処理2を実行")
# 処理2のコード
```

A.5.2　コメントアウトを使ったデバッグ

問題のある部分を一時的に無効にするために、コメントアウト（‘#‘をつける）を使うこともできます。

▼リスト A.15　コメントアウトを使ったデバッグ

```
# 問題のある部分をコメントアウト
# result = x / y  # ここでエラーが発生する可能性がある

# 代わりに別の処理を試す
result = 0
if y != 0:
    result = x / y
```

A.5.3　段階的なデバッグ

複雑なプログラムの場合は、段階的にデバッグすることが効果的です。まず、プログラムを小さな部分に分けて、それぞれの部分が正しく動作するか確認します。

▼リスト A.16　段階的なデバッグ

```
# 段階1: 変数の初期化が正しいか確認
x = 10
y = 20
print(f"段階1: x = {x}, y = {y}")

# 段階2: 計算が正しいか確認
result = x + y
print(f"段階2: result = {result}")

# 段階3: 条件分岐が正しいか確認
if result > 30:
    print("段階3: resultは30より大きい")
else:
    print("段階3: resultは30以下")
```

A.6　エラーを防ぐためのヒント

エラーを解決するだけでなく、エラーを防ぐための工夫も大切です。ここでは、エラーを防ぐためのヒントを紹介します。

A.6.1　コードを整理する

コードを整理して、読みやすくすることで、エラーを見つけやすくなります。

- 適切なインデント（字下げ）を使う
- 空行を入れて、コードのブロックを分ける
- 長い行は適切に分割する

A.6.2　コメントを書く

コードにコメント（‘#‘で始まる行）を書くことで、コードの意図を明確にし、後で見直したときに理解しやすくなります。

▼リスト A.17　コメントの例

```
# プレイヤーの初期位置を設定
player_x = 80  # 画面の中央（X座標）
player_y = 60  # 画面の中央（Y座標）

# プレイヤーの移動処理
if pyxel.btn(pyxel.KEY_LEFT):
    player_x -= 2  # 左に2ピクセル移動
```

A.6.3　変数名を工夫する

変数名は、その変数の役割がわかるような名前にしましょう。短すぎる名前や、意味のわかりにくい名前は避けましょう。

▼リスト A.18　変数名の例

```
# 良い例
player_x = 80
player_y = 60
score = 0
lives = 3

# 悪い例
x = 80
y = 60
s = 0
l = 3
```

A.6.4　エラーチェックを入れる

エラーが発生しそうな部分には、事前にチェックを入れておくと安心です。

▼リスト A.19　エラーチェックの例

```
# 0で割り算する前にチェック
if divisor != 0:
    result = dividend / divisor
else:
    result = 0  # 0で割る場合は、別の値を設定

# リストの範囲外アクセスを防ぐ
if index < len(my_list):
    value = my_list[index]
else:
    value = None  # 範囲外の場合は、別の値を設定
```

A.7　まとめ

この章では、Python プログラミングでよく発生するエラーとその解決方法を学びました。

- エラーメッセージを読んで、エラーの原因を特定する
- よくあるエラーの種類と解決方法を知る
- Pyxel でよく発生するエラーと解決方法を知る
- デバッグの方法を学ぶ
- エラーを防ぐためのヒントを知る

プログラミングでは、エラーは避けられないものです。大切なのは、エラーが発生したときに、あきらめずに原因を探り、解決する力を身につけることです。エラーを解決するたびに、プログラミングの知識と経験が増えていきます。

次の章では、Pyxel のリファレンス（命令の一覧）を紹介します。

A.8　チャレンジ問題

1. 次のプログラムにはエラーがあります。どんなエラーが発生するか予想して、修正してみよう。

▼リスト A.20　エラーのあるプログラム 1

```
import pyxel

pyxel.init(160, 120)

def update()
    if pyxel.btnp(pyxel.KEY_Q):
        pyxel.quit()

def draw():
    pyxel.cls(0)
    pyxel.text(10, 10, "Hello, Pyxel!", 7)

pyxel.run(update, draw)
```

2. 次のプログラムにはエラーがあります。どんなエラーが発生するか予想して、修正してみよう。

▼リスト A.21　エラーのあるプログラム 2

```
import pyxel

pyxel.init(160, 120)

player_x = 80
player_y = 60
```

```
score = "0"

def update():
    global player_x, player_y

    if pyxel.btn(pyxel.KEY_LEFT):
        player_x = player_x - 2

    if pyxel.btn(pyxel.KEY_RIGHT):
        player_x = player_x + 2

    score = score + 1

def draw():
    pyxel.cls(0)
    pyxel.rect(player_x, player_y, 8, 8, 11)
    pyxel.text(10, 10, "Score: " + score, 7)

pyxel.run(update, draw)
```

3. 自分が作ったゲームでエラーが発生したら、この章で学んだ方法を使って解決してみよう。エラーの内容と解決方法をノートに書き留めておくと、次回同じエラーが発生したときに役立ちます。

付録 B

Pyxel リファレンス

この章では、Pyxel で使える命令の一覧と使い方を紹介します。

B.1　リファレンスとは

リファレンス（reference）とは、「参照」や「参考資料」という意味です。プログラミングにおけるリファレンスは、使える命令（関数やメソッド）の一覧と、その使い方をまとめたものです。

この章では、Pyxel で使える主な命令を紹介します。新しいゲームを作るときや、わからない命令があるときに、この章を参考にしてください。

より詳しい情報は、Pyxel の公式ドキュメント（https://github.com/kitao/pyxel）を参照してください。

B.2　基本的な命令

Pyxel の基本的な命令を紹介します。

B.2.1　初期化と実行

▼リスト B.1　初期化と実行

```
# Pyxelの初期化
pyxel.init(width, height, title="タイトル", fps=30, quit_key=pyxel.KEY_ESCAPE)
# width: 画面の幅（ピクセル単位）
# height: 画面の高さ（ピクセル単位）
```

```
# title: ウィンドウのタイトル（省略可）
# fps: フレームレート（1秒あたりのフレーム数、省略可）
# quit_key: 終了キー（省略可）

# Pyxelの実行
pyxel.run(update, draw)
# update: 画面を更新する関数
# draw: 画面を描画する関数
```

B.2.2 入力

▼リスト B.2 入力

```
# キーが押されているかチェック
pyxel.btn(key)
# key: キーコード（例: pyxel.KEY_SPACE, pyxel.KEY_LEFT）
# 戻り値: キーが押されていればTrue、そうでなければFalse

# キーが押された瞬間をチェック
pyxel.btnp(key, hold=0, period=0)
# key: キーコード
# hold: キーを押し続けた時に、最初の入力を受け付けるまでの時間（省略可）
# period: キーを押し続けた時に、2回目以降の入力を受け付ける間隔（省略可）
# 戻り値: キーが押された瞬間であればTrue、そうでなければFalse

# マウスの位置を取得
pyxel.mouse_x  # マウスのX座標
pyxel.mouse_y  # マウスのY座標

# マウスボタンが押されているかチェック
pyxel.btn(button)
# button: ボタンコード（pyxel.MOUSE_LEFT, pyxel.MOUSE_RIGHT, pyxel.MOUSE_MIDDLE）
# 戻り値: ボタンが押されていればTrue、そうでなければFalse

# マウスボタンが押された瞬間をチェック
pyxel.btnp(button)
# button: ボタンコード
# 戻り値: ボタンが押された瞬間であればTrue、そうでなければFalse
```

B.2.3 描画

▼リスト B.3 描画

```
# 画面をクリア
pyxel.cls(col)
# col: 背景色（0-15）

# 点を描画
pyxel.pset(x, y, col)
# x, y: 点の座標
# col: 点の色（0-15）

# 線を描画
pyxel.line(x1, y1, x2, y2, col)
# x1, y1: 始点の座標
# x2, y2: 終点の座標
# col: 線の色（0-15）

# 四角形を描画（塗りつぶし）
pyxel.rect(x, y, w, h, col)
# x, y: 左上の座標
# w, h: 幅と高さ
# col: 四角形の色（0-15）

# 四角形を描画（枠のみ）
pyxel.rectb(x, y, w, h, col)
# x, y: 左上の座標
# w, h: 幅と高さ
# col: 枠の色（0-15）

# 円を描画（塗りつぶし）
pyxel.circ(x, y, r, col)
# x, y: 中心の座標
# r: 半径
# col: 円の色（0-15）

# 円を描画（枠のみ）
pyxel.circb(x, y, r, col)
# x, y: 中心の座標
# r: 半径
# col: 枠の色（0-15）

# 三角形を描画（塗りつぶし）
pyxel.tri(x1, y1, x2, y2, x3, y3, col)
# x1, y1: 1つ目の頂点の座標
# x2, y2: 2つ目の頂点の座標
# x3, y3: 3つ目の頂点の座標
# col: 三角形の色（0-15）

# 三角形を描画（枠のみ）
pyxel.trib(x1, y1, x2, y2, x3, y3, col)
# x1, y1: 1つ目の頂点の座標
# x2, y2: 2つ目の頂点の座標
# x3, y3: 3つ目の頂点の座標
# col: 枠の色（0-15）

# テキストを描画
pyxel.text(x, y, text, col)
# x, y: テキストの左上の座標
# text: 表示するテキスト
```

```
# col: テキストの色（0-15）
```

B.2.4　音と音楽

▼リスト B.4　音と音楽

```
# 効果音の定義
pyxel.sound(snd).set(note, tone, volume, effect, speed)
# snd: サウンド番号（0-63）
# note: 音階（"C0", "D0", ... "B0", "C1", ... など）
# tone: 音色（"T": 三角波, "S": 矩形波, "P": パルス波, "N": ノイズ）
# volume: 音量（"0", "1", ... "7"）
# effect: エフェクト（"N": なし, "S": スライド, "V": ビブラート, "F": フェードアウト）
# speed: 再生速度（数値が大きいほど遅い）

# 効果音の再生
pyxel.play(ch, snd)
# ch: チャンネル番号（0-3）
# snd: サウンド番号（0-63）

# 音楽の定義
pyxel.music(msc).set(tracks, patterns)
# msc: ミュージック番号（0-7）
# tracks: トラックのリスト（例: [0, 1, 2, 3]）
# patterns: 各トラックのパターンのリスト（例: [[0, 1], [2, 3], [4, 5], [6, 7]]）

# 音楽の再生
pyxel.playm(msc)
# msc: ミュージック番号（0-7）

# 音の停止
pyxel.stop()
```


B.2.5　その他

▼リスト B.5　その他

```
# フレームカウンターの取得
pyxel.frame_count
# 戻り値: プログラム開始からのフレーム数

# 乱数の生成
pyxel.rndi(a, b)
# a, b: 範囲（a以上b以下の整数）
# 戻り値: a以上b以下のランダムな整数
```

```
# プログラムの終了
pyxel.quit()
```

B.3　キーコード一覧

Pyxel では、キーボードの各キーに対して、キーコードが定義されています。キーコードは、‘pyxel.KEY_XXX‘の形式で表されます。

B.3.1　文字キー

▼表 B.1　文字キー

キーコード	説明
pyxel.KEY_A	A キー
pyxel.KEY_B	B キー
pyxel.KEY_C	C キー
..	..
pyxel.KEY_Z	Z キー
pyxel.KEY_0	0 キー
pyxel.KEY_1	1 キー
..	..
pyxel.KEY_9	9 キー

B.3.2 特殊キー

▼表 B.2 特殊キー

キーコード	説明
pyxel.KEY_SPACE	スペースキー
pyxel.KEY_RETURN	Enter キー
pyxel.KEY_ESCAPE	Esc キー
pyxel.KEY_BACKSPACE	Backspace キー
pyxel.KEY_TAB	Tab キー
pyxel.KEY_LEFT	左矢印キー
pyxel.KEY_RIGHT	右矢印キー
pyxel.KEY_UP	上矢印キー
pyxel.KEY_DOWN	下矢印キー
pyxel.KEY_SHIFT	Shift キー
pyxel.KEY_CTRL	Ctrl キー
pyxel.KEY_ALT	Alt キー

B.4 色コード一覧

Pyxel では、16 色のパレットが用意されています。色は 0 から 15 までの数値で指定します。

▼表 B.3　色コード

色コード	色
0	黒
1	紺色
2	紫色
3	緑色
4	茶色
5	暗い青色
6	水色
7	白
8	赤色
9	オレンジ
10	黄色
11	黄緑色
12	青色
13	ピンク
14	灰色
15	明るい灰色

B.5　使用例

ここでは、Pyxel の命令を使った例を紹介します。

B.5.1　基本的な使い方

▼リスト B.6　基本的な使い方

```
import pyxel

# Pyxelを初期化
pyxel.init(160, 120, title="Pyxelの基本")

# 画面を更新する関数
def update():
    # ESCキーが押されたら終了
    if pyxel.btnp(pyxel.KEY_ESCAPE):
        pyxel.quit()

# 画面を描画する関数
def draw():
    # 画面を黒色（0）でクリア
```

```
    pyxel.cls(0)

    # テキストを表示
    pyxel.text(55, 41, "Hello, Pyxel!", 7)

    # 四角形を描画
    pyxel.rect(50, 50, 30, 20, 11)

    # 円を描画
    pyxel.circ(80, 80, 15, 8)

# Pyxelの実行
pyxel.run(update, draw)
```

B.5.2　キー入力の使い方

▼リスト B.7　キー入力の使い方

```
import pyxel

# Pyxelを初期化
pyxel.init(160, 120, title="キー入力")

# プレイヤーの位置
player_x = 80
player_y = 60

# 画面を更新する関数
def update():
    global player_x, player_y

    # ESCキーが押されたら終了
    if pyxel.btnp(pyxel.KEY_ESCAPE):
        pyxel.quit()

    # 矢印キーでプレイヤーを移動
    if pyxel.btn(pyxel.KEY_LEFT):
        player_x = max(player_x - 2, 0)

    if pyxel.btn(pyxel.KEY_RIGHT):
        player_x = min(player_x + 2, 160 - 8)

    if pyxel.btn(pyxel.KEY_UP):
        player_y = max(player_y - 2, 0)

    if pyxel.btn(pyxel.KEY_DOWN):
        player_y = min(player_y + 2, 120 - 8)

    # スペースキーが押された瞬間に効果音を再生
    if pyxel.btnp(pyxel.KEY_SPACE):
        pyxel.play(0, 0)
```

```
# 画面を描画する関数
def draw():
    # 画面を黒色（0）でクリア
    pyxel.cls(0)

    # プレイヤーを描画
    pyxel.rect(player_x, player_y, 8, 8, 11)

    # 操作方法を表示
    pyxel.text(5, 5, "Arrow keys: Move", 7)
    pyxel.text(5, 15, "Space: Sound", 7)

# 効果音の定義
pyxel.sound(0).set(note="C4 E4 G4", tone="T", volume="7", effect="N", speed=5)

# Pyxelの実行
pyxel.run(update, draw)
```

B.5.3　当たり判定の使い方

▼リスト B.8　当たり判定の使い方

```
import pyxel
import random

# Pyxelを初期化
pyxel.init(160, 120, title="当たり判定")

# プレイヤーの位置
player_x = 80
player_y = 60

# アイテムの位置
item_x = random.randint(10, 150)
item_y = random.randint(10, 110)

# スコア
score = 0

# 画面を更新する関数
def update():
    global player_x, player_y, item_x, item_y, score

    # ESCキーが押されたら終了
    if pyxel.btnp(pyxel.KEY_ESCAPE):
        pyxel.quit()

    # 矢印キーでプレイヤーを移動
    if pyxel.btn(pyxel.KEY_LEFT):
        player_x = max(player_x - 2, 0)

    if pyxel.btn(pyxel.KEY_RIGHT):
```

```
            player_x = min(player_x + 2, 160 - 8)

        if pyxel.btn(pyxel.KEY_UP):
            player_y = max(player_y - 2, 0)

        if pyxel.btn(pyxel.KEY_DOWN):
            player_y = min(player_y + 2, 120 - 8)

        # プレイヤーとアイテムの当たり判定
        if (player_x < item_x + 8 and
            player_x + 8 > item_x and
            player_y < item_y + 8 and
            player_y + 8 > item_y):
            # アイテムを取得
            item_x = random.randint(10, 150)
            item_y = random.randint(10, 110)
            score += 1
            pyxel.play(0, 0)

    # 画面を描画する関数
    def draw():
        # 画面を黒色（0）でクリア
        pyxel.cls(0)

        # プレイヤーを描画
        pyxel.rect(player_x, player_y, 8, 8, 11)

        # アイテムを描画
        pyxel.rect(item_x, item_y, 8, 8, 10)

        # スコアを表示
        pyxel.text(5, 5, f"Score: {score}", 7)

    # 効果音の定義
    pyxel.sound(0).set(note="C4 E4 G4", tone="T", volume="7", effect="N", speed=5)

    # Pyxelの実行
    pyxel.run(update, draw)
```

B.6 まとめ

この章では、Pyxel で使える主な命令を紹介しました。

- 基本的な命令（初期化、実行、入力、描画、音と音楽、その他）
- キーコード一覧
- 色コード一覧
- 使用例（基本的な使い方、キー入力、当たり判定）

これらの命令を組み合わせることで、さまざまなゲームを作ることができます。わからない命令があるときは、この章を参考にしてください。

また、Pyxel の公式ドキュメント（https://github.com/kitao/pyxel）には、より詳しい情報が載っています。さらに学びたい場合は、公式ドキュメントも参考にしてみてください。

B.7 チャレンジ問題

1. 次のプログラムを実行して、どのような図形が描かれるか予想してみよう。その後、実際に実行して確認しよう。

▼リスト B.9　図形描画のプログラム

```
import pyxel

pyxel.init(160, 120)

def update():
    if pyxel.btnp(pyxel.KEY_ESCAPE):
        pyxel.quit()

def draw():
    pyxel.cls(0)
    pyxel.rect(30, 30, 30, 30, 11)
    pyxel.rectb(90, 30, 30, 30, 8)
    pyxel.circ(45, 85, 15, 10)
    pyxel.circb(105, 85, 15, 14)
    pyxel.line(30, 30, 90, 90, 7)

pyxel.run(update, draw)
```

2. 次のプログラムを実行して、キーボードの各キーを押したときの動作を確認しよう。

▼リスト B.10　キー入力のプログラム

```
import pyxel

pyxel.init(160, 120)

def update():
    if pyxel.btnp(pyxel.KEY_ESCAPE):
        pyxel.quit()

def draw():
    pyxel.cls(0)

    # 各キーの状態を表示
    pyxel.text(10, 10, "Arrow keys:", 7)
    pyxel.text(30, 20, "LEFT: " + ("ON" if pyxel.btn(pyxel.KEY_LEFT) else "OFF"), 7)
    pyxel.text(30, 30, "RIGHT: " + ("ON" if pyxel.btn(pyxel.KEY_RIGHT) else "OFF"), 7)
```

```
    pyxel.text(30, 40, "UP: " + ("ON" if pyxel.btn(pyxel.KEY_UP) else "OFF"), 7)
    pyxel.text(30, 50, "DOWN: " + ("ON" if pyxel.btn(pyxel.KEY_DOWN) else "OFF"), 7)

    pyxel.text(10, 70, "Other keys:", 7)
    pyxel.text(30, 80, "SPACE: " + ("ON" if pyxel.btn(pyxel.KEY_SPACE) else "OFF"), 7)
    pyxel.text(30, 90, "Z: " + ("ON" if pyxel.btn(pyxel.KEY_Z) else "OFF"), 7)
    pyxel.text(30, 100, "X: " + ("ON" if pyxel.btn(pyxel.KEY_X) else "OFF"), 7)

pyxel.run(update, draw)
```

3. この章で紹介した命令を使って、自分だけのミニゲームを作ってみよう。例えば、マウスでキャラクターを操作するゲームや、キーボードで音楽を演奏するプログラムなど、自由に作ってみよう。

著者紹介

からあげ / @karaage0703

Pyxel 4 Kids Book

2025 年 5 月 5 日　仮発行 v0.1.0

著　者　からあげ
編　集　からあげ
発行所　なし