「プログラミング演習(Python) レポート１」

C クラス　担当教員:小林先生

20K1026 日比野将己

１．課題：

問題：ピンボールを作成せよ。第３回の課題で作成したプログラムを参考にしても良い。さらに次の項目のうち、出来るものをいくつか選んで実装せよ。

- ターゲットとなる壁を設置し、ターゲットに当たるとスコアが加算される。
- スコアを表示する
- ドロップターゲットを作成し、当たるとターゲットが落ちる（消える）ようになる。
- ドロップターゲットの１セット（一並び）が落ちると、元通りに戻るようにする。
- プランジャー（または同等な役割を果たすもの）を用意して、ボールが発射される仕組みを作る
- 斜めに設置された壁で、跳ね返る。
- フラップをプログラムする。
- フラップでボールが跳ね返るようにする。
- 側面の壁のレイアウトを変更し、左右にアウトレーンを作る。
- 左右にリターンレーン、フリッパーレーンを作成する。
- その他自由な機能

２．課題の目的：

クラスや関数などを活用して、機能のまとまりを意識したオブジェクト指向の考え方を習得する。

また、イベントとイベントハンドラを連結させ、インタラクティブなプログラム処理を理解する。

３．方法：

今回のレポートでは、上記の項目のうち、スコアの加算、スコアの表示、ターゲットの削除、プランジャー、斜めの壁の反射、その他の機能としてスタート画面の作成等の機能を実装した。方針としては、前回の課題で作成したプログラムをもとに、これらの機能を拡張していく。

はじめに、スコアについて説明する。「ボールがブロックに当たったとき、変数スコアをプラス１０にする」とすることで、変数の値が増え、スコアの値を create_text を用いることでキャンバス上に表示させることができた。また、普通は再実行すると変数の初期値から再度読み込まれるため、スコアが０からスタートとなるが、ファイルの読み込み・書き込みを用いてスコアの値を別のファイルに保存しておくことで疑似的なデータベース構造を構築した。そうすることにより、再実行したときに前回のスコアの値の続きからできるように、工夫した。

２つ目に、ターゲットの削除について説明する。「ボールの上部か下部がターゲットの上線か下線を超えたとき、ボールを鉛直方向に跳ね返す」とすることで、ターゲットの上

下で反射させることを可能にした。水平方向の反射については、「ボールの左側と右側の間にブロックの左側か右側が来たとき（ボールと線が完全に被ったとき)、ボールを水平方向に跳ね返す」とすることで、水平方向に反射させることを可能にした。このように、「越えたとき」と「重なったとき」と条件を微妙に変えることで、鉛直方向と水平方向の競合を阻止することを実現できた。

３つ目に、斜めの壁について説明する。ボールを右下に描写し、「スペースのキーボードが押されたら、ボールの初速をマイナス〇倍にする」とすることで、ボールが上に動きプランジャーのレールから飛び出ることが可能になった。また、ｘ軸方向の初速もあるため、左の縦線にも当たり判定を付け反射させなくてはならない。外枠同様、「ボールが線を左に越えたら」とすると、左側にボールがあるときに挙動がおかしくなるため、縦線に当たり判定の幅を持たせ、「その幅にボールが入ったとき」とすることで、左右どちらからでも反射させることが可能になる。

４つ目に、斜めの壁の反射について説明する。まず、斜めの壁の端点の座標を求め、それをもとに直線の方程式を導く。そして、「ボールがｘとｙの範囲内にあって、ボールの上部か下部が方程式より大きい（小さいとき)、ボールを鉛直方向に反射させる」とすることで、斜めの壁の反射が可能になる。

５つ目に、スタート画面の説明をする。基本的にスペースを押すことで画面が切り替わり、スコアを続きからやるか、最初からやるかを選べるようになっている。ここではイベントハンドラと count を用いることで、ターンの制御を行った。

最後に、その他細かいところについて、説明する。パドル等でボールを反射させるとき、スピードが速すぎてめり込んでしまうと高速にバウンドしてしまうバグがおきてしまう。しかし、これについては１回目の反射判定の時にボールの外枠を反射する図形のライン上に持っていくことで、２回目にまた反射判定がされることが無くなり、きちんと一回のみ反射するようになる。また、ターゲットに関しては、リストにある座標からランダムに選ばれるようにすることで、配置を動的に決定することができる。

４．結果：

実際のゲームの流れとしては、スペースキーでスコアの値を選択し、再びスペースキーを押すことで、ボールがパドルから発射される。ボールは重力の影響を受けており、ボールは弧を描いて落ちていく。矢印キーの左右でパドルを動かし、ボールが落ちないようにうまくはね返し、見事全部のブロックを消すことができたら、クリアとなる。

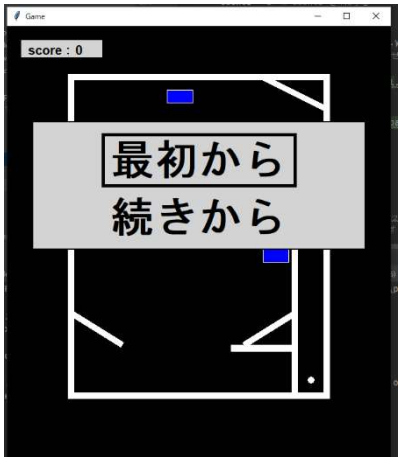


図１．スタート画面

図１の画面で矢印キーの上下を押すことで、score を続きからにするか、最初からにするかを選択できるようになっている。

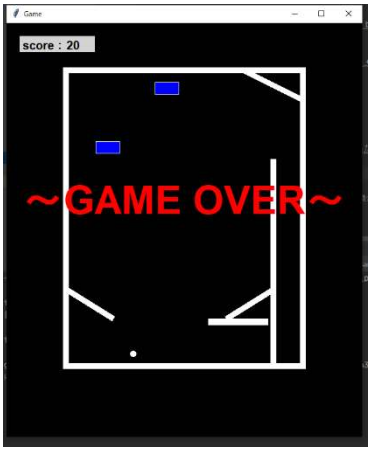


図２．終了画面

図２の画面は、ゲームが終了したときの画面である。ボールが下に落っこちると「～GAME OVE～」、ブロックが全て消えたら「CLEAR!!」と表示されるようになっている。また、難易度を上げるために、パドルに当たる回数制限を８回に設け、越えると「～GAME OVE～」と表示され、終了してしまうようになっている。

また、パドルは当たった回数によって、パドルが白からだんだん赤になっていくため、焦りが煽られるという小要素も取り入れてある。

５．　考察；

今回のプログラミングにおける最大の改善点は、膨大な数の行である。割と PEP8 のコーディング規約を心がけてコーディングを行っていたので、多少は見やすいコードになっ

ていると思う。しかし、実際のところ、これだけのプログラムでも 400 行越えとものすごい量になってしまい、自分で見返すのがとても大変になってしまっている。

春学期の時と比べれば、関数やクラスを多少なりとも活用できるようになっているが、所々で if 文を乱用して処理を書いてしまったり、行き詰まると count を用いて無理やり処理制御したりしてしまっているため、このような見にくいコードになってしまったのではないかと考えられる。

また、自分のスタイルをまだ確立できていないため、コーディングに統一感がないのも原因であると考えられる。

読みやすいコードは可読性だけでなく保守性も高くなるため、今後プログラミングをしていくときは、見やすくて分かりやすい、美しいコーディングを心がけていくことが一番の課題であると考える。

また、プログラミングを上達させるには、たくさん書いて経験値を増やしていく以外にないと考えられる。したがって、今回はこれらの機能しか実装できなかったので、数学の知識を活用するフラップの作成や、ＢＧＭの実装など、色々な要素を複合した幅広いプログラミングにも挑戦していくことで、自分自身のスキルアップにつながるのではないかと考える。

６．　参考文献：

- 講義資料
- たのしいプログラミング　PYTHON ではしめよう！
- Python によるプログラミング

７．　付録：

以下の図 3 ～図 13 は、今回作成したプログラムのソースコード、図 14 はスコアを保存するテキストファイルの添付である。

```
# 20K1026 日比野将己
# 第１回　レポート課題プログラム
# ------------------------
# プログラム名: 20K1026-日比野将己-R01.py

from tkinter import *
from dataclasses import dataclass
import random
import time

# 初期状態の設定
SPEEDS = [-2, -1, 1, 2]  # ボールのx方向初速選択肢
BLOCKS_X = [150, 250, 350, 400]  # ブロックのｘ座標のリスト
random.shuffle(BLOCKS_X)
BLOCKS_Y = [100, 200, 300, 350]  # ブロックのｙ座標のリスト
random.shuffle(BLOCKS_Y)
BLOCKS_W = [40, 40, 40, 40]  # ブロックの幅のリスト
BLOCKS_H = [20, 20, 20, 20]  # ブロックの高さのリスト
DURATION = 0.01  # 描画間隔(秒)
BALL_X0 = 470  # ボールの初期位置(x)
BALL_Y0 = 550  # ボールの初期位置(y)
PADDLE_X0 = 350  # パドルの初期位置(x)
PADDLE_Y0 = 500  # パドルの初期位置(y)
PADDLE_VX = 5  # パドルの速度
WALL_X0 = 100  # 外枠のｘ座標
WALL_Y0 = 80  # 外枠のｙ座標
WALL_W = 400  # 外枠の幅
WALL_H = 500  # 外枠の高さ

BALL_VX = random.choice(SPEEDS)  # ボールのx方向初速
BALL_VY = -10  # ボールのy方向初速

GRAVITY = 0.12  # 重力加速度
REACTION = 1  # 反発係数

count1 = 0  # ブロックのx座標を判定する count
count2 = 0  # ブロックのｙ座標を判定する count
count3 = 0  # スタート画面制御
count4 = 0  # db からダウンロード

play = 0  # ゲーム前か後か
```

図３．ソースコード 01

```
play = 0  # ゲーム前か後か
score = 0  # スコアの初期化
select_x = 150  # 以下４つ、初めから続きからの選択の四角
select_y = 170
select_w = 300
select_h = 80
c = 0  # パドルの色

# 変える色を用意する。
COLORS = ["#1f0000", "#3f0000", "#5f0000", "#7f0000", "#9f0000", "#bf0000", "#df0000", "#ff0000"] # だんだん赤


# ------------------------
@dataclass
class Ball:
    id: int
    x: int
    y: int
    vx: int
    vy: int
    d: int
    c: str


@dataclass
class Paddle:
    id: int
    x: int
    y: int
    w: int
    h: int
    vx: int
    c: str


@dataclass
class Wall:
    x: int
    y: int
    w: int
    h: int
```

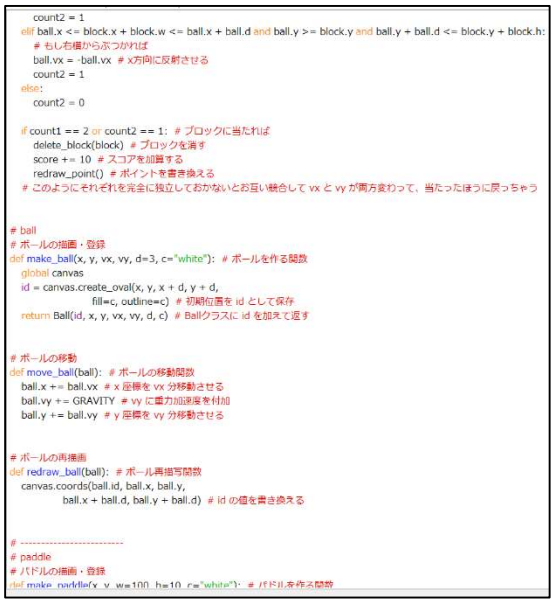
図４．ソースコード 02

```
@dataclass
class Wall:
    x: int
    y: int
    w: int
    h: int


@dataclass
class Block:
    id: int
    x: int
    y: int
    w: int
    h: int
    c: str


@dataclass
class Point:
    id: int
    x: int
    y: int
    score: int


@dataclass
class Select:
    id: int
    y: int


# ------------------------

def make_wall(wall):  # 外枠を作る関数
    global canvas
    canvas.create_rectangle(wall.x, wall.y, wall.x + wall.w, wall.y + wall.h, width=10, outline="white")  # 外枠
    canvas.create_line(wall.x + wall.w - 50, wall.y + 150, wall.x + wall.w - 50, wall.y + wall.h, width=10,
                       fill="white")  # 縦線
    canvas.create_line(wall.x + wall.w - 100, wall.y, wall.x + wall.w, wall.y + 50, width=10, fill="white")  # 右上斜め
    canvas.create_line(wall.x, paddle.y - 50, wall.x + 80, paddle.y, width=10, fill="white")  # 左下斜め
```

図５．ソースコード 03

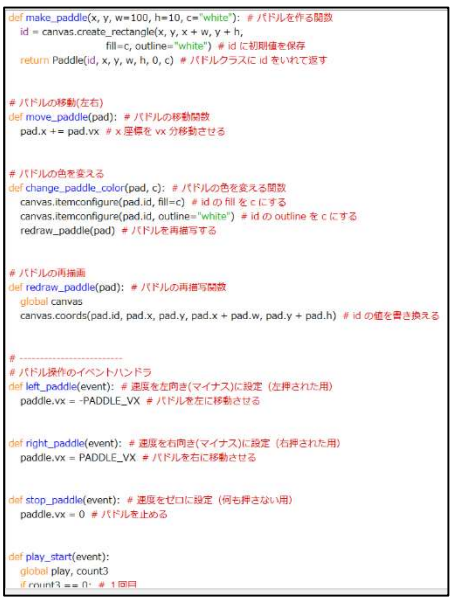
```
    canvas.create_line(wall.x, paddle.y - 50, wall.x + 80, paddle.y, width=10, fill="white")  # 左下斜め
    canvas.create_line(wall.x + wall.w - 130, paddle.y, wall.x + wall.w - 50, paddle.y - 50, width=10,
                       fill="white")  # 右下斜め


# ブロックの描画
def make_block(x, y, w, h, c="blue"):  # ブロックを作成する関数
    global canvas
    id = canvas.create_rectangle(x, y, x + w, y + h, fill=c, outline="white")  # id に保存
    return Block(id, x, y, w, h, c)  # 生成クラスを返す


# ブロックの消去
def delete_block(block):  # ブロックを消す関数
    global blocks
    canvas.delete(block.id)  # ブロックのidを消す
    blocks.remove(block)  # ブロック自体を消す


# ブロックをまとめて描画
def make_blocks(x, y, w, h):
    blocks = []  # ブロックのリスト
    for i in range(len(x)):  # 4回繰り返す
        blocks.append(make_block(x[i], y[i], w[i], h[i]))  # ブロックを生成してリストに追加
    return blocks  # リストを返す


def block_judge(block):  # ブロックに当たったかを判定する関数
    global canvas, count1, count2, score
    if ball.x + ball.d > block.x and ball.x < block.x + block.w:  # もしブロックの上か下にボールがあれば
        count1 = 1  # count1 を１にする
    else:
        count1 = 0  # その他は 0
    if ball.y + ball.d > block.y and ball.y < block.y + block.h and count1 == 1:  # count1 が１で、ボールがブロックの上か下に当たれば
        ball.vy = -ball.vy  # ｙ方向に反射させる
        count1 = 2

    if ball.x <= block.x <= ball.x + ball.d and ball.y >= block.y and ball.y + ball.d <= block.y + block.h:
        # もし左横からぶつかれば
        ball.vx = -ball.vx  # x方向に反射させる
        count2 = 1
```

図６．　ソースコード 04



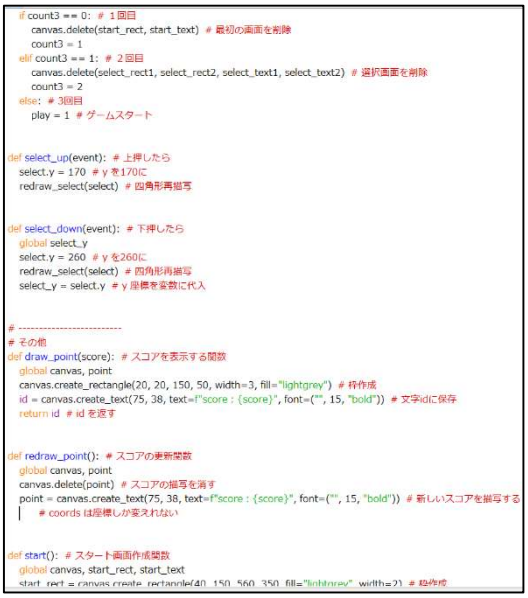
```
        count2 = 1
    elif ball.x <= block.x + block.w <= ball.x + ball.d and ball.y >= block.y and ball.y + ball.d <= block.y + block.h:
        # もし右横からぶつかれば
        ball.vx = -ball.vx  # x方向に反射させる
        count2 = 1
    else:
        count2 = 0

    if count1 == 2 or count2 == 1:  # ブロックに当たれば
        delete_block(block)  # ブロックを消す
        score += 10  # スコアを加算する
        redraw_point()  # ポイントを書き換える
    # このようにそれぞれを完全に独立しておかないとお互い競合して vx と vy が両方変わって、当たったほうに戻っちゃう


# ball
# ボールの描画・登録
def make_ball(x, y, vx, vy, d=3, c="white"):  # ボールを作る関数
    global canvas
    id = canvas.create_oval(x, y, x + d, y + d,
                            fill=c, outline=c)  # 初期位置を id として保存
    return Ball(id, x, y, vx, vy, d, c)  # Ballクラスに id を加えて返す


# ボールの移動
def move_ball(ball):  # ボールの移動関数
    ball.x += ball.vx  # x 座標を vx 分移動させる
    ball.vy += GRAVITY  # vy に重力加速度を付加
    ball.y += ball.vy  # y 座標を vy 分移動させる


# ボールの再描画
def redraw_ball(ball):  # ボール再描写関数
    canvas.coords(ball.id, ball.x, ball.y,
                  ball.x + ball.d, ball.y + ball.d)  # id の値を書き換える


# ------------------------
# paddle
# パドルの描画・登録
def make_paddle(x, y, w=100, h=10, c="white"):  # パドルを作る関数
```

図 7．ソースコード 05



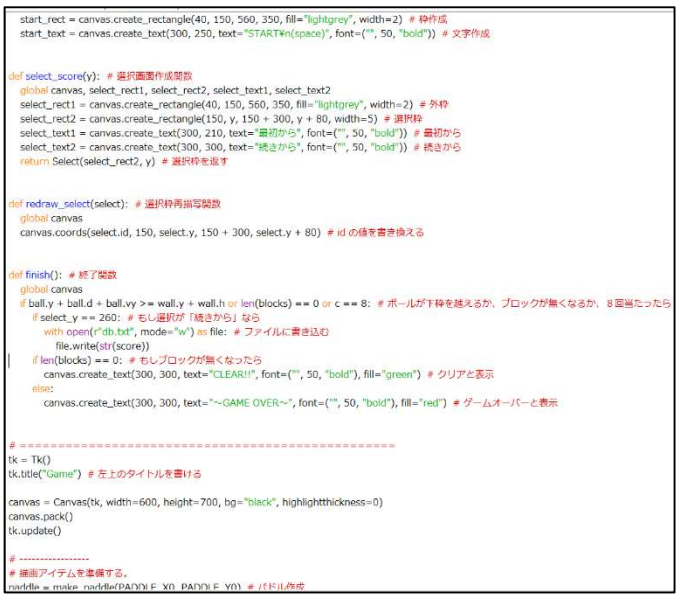
```
def make_paddle(x, y, w=100, h=10, c="white"):  # パドルを作る関数
    id = canvas.create_rectangle(x, y, x + w, y + h,
                                 fill=c, outline="white")  # id に初期値を保存
    return Paddle(id, x, y, w, h, 0, c)  # パドルクラスに id をいれて返す


# パドルの移動(左右)
def move_paddle(pad):  # パドルの移動関数
    pad.x += pad.vx  # x 座標を vx 分移動させる


# パドルの色を変える
def change_paddle_color(pad, c):  # パドルの色を変える関数
    canvas.itemconfigure(pad.id, fill=c)  # id の fill を c にする
    canvas.itemconfigure(pad.id, outline="white")  # id の outline を c にする
    redraw_paddle(pad)  # パドルを再描写する


# パドルの再描画
def redraw_paddle(pad):  # パドルの再描写関数
    global canvas
    canvas.coords(pad.id, pad.x, pad.y, pad.x + pad.w, pad.y + pad.h)  # id の値を書き換える


# ------------------------
# パドル操作のイベントハンドラ
def left_paddle(event):  # 速度を左向き(マイナス)に設定（左押された用）
    paddle.vx = -PADDLE_VX  # パドルを左に移動させる


def right_paddle(event):  # 速度を右向き(マイナス)に設定（右押された用）
    paddle.vx = PADDLE_VX  # パドルを右に移動させる


def stop_paddle(event):  # 速度をゼロに設定（何も押さない用）
    paddle.vx = 0  # パドルを止める


def play_start(event):
    global play, count3
    if count3 == 0:  # 1回目
```

図 8．ソースコード 06



```
    if count3 == 0:  # 1回目
        canvas.delete(start_rect, start_text)  # 最初の画面を削除
        count3 = 1
    elif count3 == 1:  # 2回目
        canvas.delete(select_rect1, select_rect2, select_text1, select_text2)  # 選択画面を削除
        count3 = 2
    else:  # 3回目
        play = 1  # ゲームスタート


def select_up(event):  # 上押したら
    select.y = 170  # y を170に
    redraw_select(select)  # 四角形再描写


def select_down(event):  # 下押したら
    global select_y
    select.y = 260  # y を260に
    redraw_select(select)  # 四角形再描写
    select_y = select.y  # y 座標を変数に代入


# ------------------------
# その他
def draw_point(score):  # スコアを表示する関数
    global canvas, point
    canvas.create_rectangle(20, 20, 150, 50, width=3, fill="lightgrey")  # 枠作成
    id = canvas.create_text(75, 38, text=f"score : {score}", font=("", 15, "bold"))  # 文字idに保存
    return id  # id を返す


def redraw_point():  # スコアの更新関数
    global canvas, point
    canvas.delete(point)  # スコアの描写を消す
    point = canvas.create_text(75, 38, text=f"score : {score}", font=("", 15, "bold"))  # 新しいスコアを描写する
        # coords は座標しか変えれない


def start():  # スタート画面作成関数
    global canvas, start_rect, start_text
    start_rect = canvas.create_rectangle(40, 150, 560, 350, fill="lightgrey", width=2)  # 枠作成
```

図 9．ソースコード 07



```
    start_rect = canvas.create_rectangle(40, 150, 560, 350, fill="lightgrey", width=2)  # 枠作成
    start_text = canvas.create_text(300, 250, text="START¥n(space)", font=("", 50, "bold"))  # 文字作成


def select_score(y):  # 選択画面作成関数
    global canvas, select_rect1, select_rect2, select_text1, select_text2
    select_rect1 = canvas.create_rectangle(40, 150, 560, 350, fill="lightgrey", width=2)  # 外枠
    select_rect2 = canvas.create_rectangle(150, y, 150 + 300, y + 80, width=5)  # 選択枠
    select_text1 = canvas.create_text(300, 210, text="最初から", font=("", 50, "bold"))  # 最初から
    select_text2 = canvas.create_text(300, 300, text="続きから", font=("", 50, "bold"))  # 続きから
    return Select(select_rect2, y)  # 選択枠を返す


def redraw_select(select):  # 選択枠再描写関数
    global canvas
    canvas.coords(select.id, 150, select.y, 150 + 300, select.y + 80)  # id の値を書き換える


def finish():  # 終了関数
    global canvas
    if ball.y + ball.d + ball.vy >= wall.y + wall.h or len(blocks) == 0 or c == 8:  # ボールが下枠を越えるか、ブロックが無くなるか、8回当たったら
        if select_y == 260:  # もし選択が「続きから」なら
            with open(r"db.txt", mode="w") as file:  # ファイルに書き込む
                file.write(str(score))
        if len(blocks) == 0:  # もしブロックが無くなったら
            canvas.create_text(300, 300, text="CLEAR!!", font=("", 50, "bold"), fill="green")  # クリアと表示
        else:
            canvas.create_text(300, 300, text="~GAME OVER~", font=("", 50, "bold"), fill="red")  # ゲームオーバーと表示


# ================================================
tk = Tk()
tk.title("Game")  # 左上のタイトルを書ける

canvas = Canvas(tk, width=600, height=700, bg="black", highlightthickness=0)
canvas.pack()
tk.update()

# ----------------
# 描画アイテムを準備する。
paddle = make_paddle(PADDLE_X0, PADDLE_Y0)  # パドル作成
```

図 10．　ソースコード 08

```
paddle = make_paddle(PADDLE_X0, PADDLE_Y0)  # パドル作成
ball = make_ball(BALL_X0, BALL_Y0, BALL_VX, BALL_VY, 10)  # ボール作成
wall = Wall(WALL_X0, WALL_Y0, WALL_W, WALL_H)  # 外枠作成
make_wall(wall)  # 実際に外枠作成
blocks = make_blocks(BLOCKS_X, BLOCKS_Y, BLOCKS_W, BLOCKS_H)  # ブロック作成
point = draw_point(score)  # スコアの描写
select = select_score(select_y)  # 選択枠の描写
start()  # スタート画面の制御

# イベントと、イベントハンドラを連結する。
canvas.bind_all('<KeyPress-Left>', left_paddle)  # Left 押したら left_paddle 実行
canvas.bind_all('<KeyPress-Right>', right_paddle)  # Right 押したら right_paddle 実行
canvas.bind_all('<KeyRelease-Left>', stop_paddle)  # Left 離したら stop_paddle 実行
canvas.bind_all('<KeyRelease-Right>', stop_paddle)  # Right 離したら stop_paddle 実行
canvas.bind_all('<KeyPress-space>', play_start)  # Space 押したら play_start 実行
canvas.bind_all('<KeyPress-Up>', select_up)  # Up 押したら select_up 実行
canvas.bind_all('<KeyPress-Down>', select_down)  # Down 押したら select_down 実行

# -------------------------
# プログラムのメインループ
while True:
    move_paddle(paddle)  # パドルの移動
    if paddle.x <= wall.x:  # パドルが左横の棒に当たったら
        paddle.x = wall.x  # パドルをその場に止める
    if paddle.x + paddle.w >= wall.x + wall.w - 50:  # パドルが右横の棒に当たったら
        paddle.x = wall.x + wall.w - 50 - paddle.w  # パドルをその場に止める

    if play == 0:
        redraw_paddle(paddle)  # パドルの再描画
        tk.update()  # 描画が画面に反映される。
        time.sleep(DURATION)  # 次に描画するまで、sleep する。
        continue

    if select_y == 260 and count4 == 0:  # 下選んで一回目なら
        with open(r"db.txt") as file:  # ファイルを読み込む
            score = int(file.readline())  # score に代入
            redraw_point()  # スコア表示
        count4 = 1  # 1にする

    move_ball(ball)  # ボールの移動
    if ball.x + ball.vx <= wall.x:  # ボールが左枠を越えたら
```

図 11． ソースコード 09

```
    if ball.x + ball.vx <= wall.x:  # ボールが左枠を越えたら
        ball.vx = -ball.vx  # ボールを反射させる
    if ball.x + ball.d + ball.vx >= wall.x + wall.w:  # ボールが右枠を越えたら
        ball.vx = -ball.vx  # ボールを反射させる
    if ball.y + ball.vy <= wall.y:  # ボールが上枠を越えたら
        ball.vy = -ball.vy  # ボールを反射させる
    if wall.x + wall.w - 50 >= ball.x >= wall.x + wall.w - 52 and ball.y >= wall.y + 150:  # 縦線で跳ね返る
        ball.vx = -ball.vx
    if wall.x + wall.w - 100 <= ball.x <= wall.x + wall.w and wall.y <= ball.y <= wall.y + 50 and ball.y <= (
            1 / 2) * ball.x - 120:  # 右上斜めで跳ね返る
        ball.vy = -ball.vy  # y方向に跳ね返る
        ball.vx = -abs(ball.vx)  # 絶対マイナス方向
    if wall.x <= ball.x <= wall.x + 80 and paddle.y - 50 <= ball.y + ball.d <= paddle.y and (
            5 / 8) * ball.x + 387.5 <= ball.y + ball.d:  # 左下斜めで跳ね返る
        ball.vy = -ball.vy  # y 方向に跳ね返る
        ball.y = (5 / 8) * ball.x + 387.5 - ball.d  # 線上にする

    if not (not (wall.x + wall.w - 130 <= ball.x + ball.d <= wall.x + wall.w - 50) or not (
            paddle.y - 50 <= ball.y + ball.d <= paddle.y) or not (
            ball.y + ball.d >= -(5 / 8) * ball.x + ball.d + 731.25)):  # 右下斜めで跳ね返る
        ball.vy = -ball.vy  # y 方向に跳ね返る
        ball.y = -(5 / 8) * ball.x + ball.d + 731.25 - ball.d  # 線上にする

    if ball.y + ball.d + ball.vy >= wall.y + wall.h or len(blocks) == 0 or c == 8:  # ボールが下枠を越えたら
        if select_y == 260:  # もし「続きから」なら
            with open(r"db.txt", mode="w") as file:  # ファイルに保存する
                file.write(str(score))
        finish()  # 文字表示
        break  # while を抜ける（終了）

    # ボールの下側がパドルの上面に届き、横位置がパドルと重なる
    if (paddle.y <= ball.y + ball.d <= paddle.y + paddle.h ¥
            and paddle.x <= ball.x + ball.d / 2 <= paddle.x + paddle.w):  # パドルにボールが当たったら
        change_paddle_color(paddle, COLORS[c])  # だんだん赤くなる
        c += 1  # インデックスを＋！
        ball.vy = -ball.vy * REACTION  # ボールの移動方向が変わる（反射が大きくなる）
        ball.y = paddle.y - ball.d  # 線上にする

    if ball.x <= wall.x + 50 and ball.y + ball.d >= paddle.y ¥
            or ball.x >= wall.x + wall.w - 100 and ball.y + ball.d >= paddle.y:  # ボールが左の棒か右の棒に当たったら
        ball.vy = -ball.vy  # ボールを反射させる
        ball.y = paddle.y - ball.d  # 線上にする

    for block in blocks:  # 4 回繰り返す（今回は 4 つ作るから）
        block_judge(block)  # ボールがブロックに当たったら跳ね返す

    redraw_paddle(paddle)  # パドルの再描画
    redraw_ball(ball)  # ボールの再描画
    tk.update()  # 描画が画面に反映される。
    time.sleep(DURATION)  # 次に描画するまで、sleep する。

tk.mainloop()
```

図 12． ソースコード 10

図 13． ソースコード 11

```
File  Edit  Format  Run  Options  Window  Help
730
```

図 14． スコア保存ファイル