OOP Introduction

- Why OOP?
  - Duplicate code is a Bad Thing
  - Code will always be changed
  - Similar reason as why we need UVM

- Aside from small "throw-away” programs that are written for some single task and only run once, you'll almost always need to update your code to either **fix bugs** or **add new features**
- A large part of writing good software is writing software that is **readable** and **easy to change**

OOP Basics: Divide into Classes

- OOP is a powerful way to divide a programming problem into manageable parts
- When you create a Class, you are creating a new **data type**
- Objects of this data type will have all of the attributes and abilities that you design into the Class
- Designing good classes that encapsulate the attributes and abilities of an entity can make programming complex problems much simpler

Some OOP Languages

- C++
- Java
- Python
- System Verilog
  - You will use a lot of OOP in verification

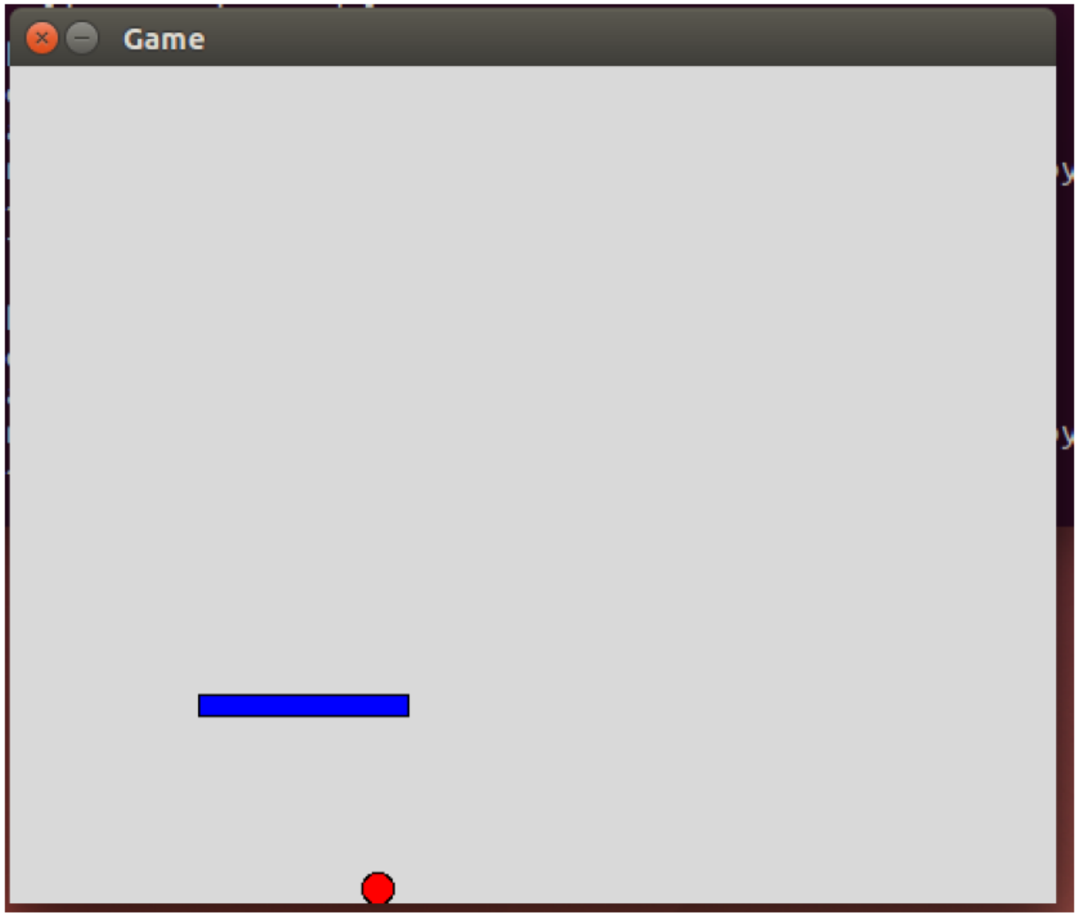
Lab6: Basic OOP Practice

- Learn from a sample python program
  - ball_game.py

- Write a simple OOP program
  - reversi_game.py

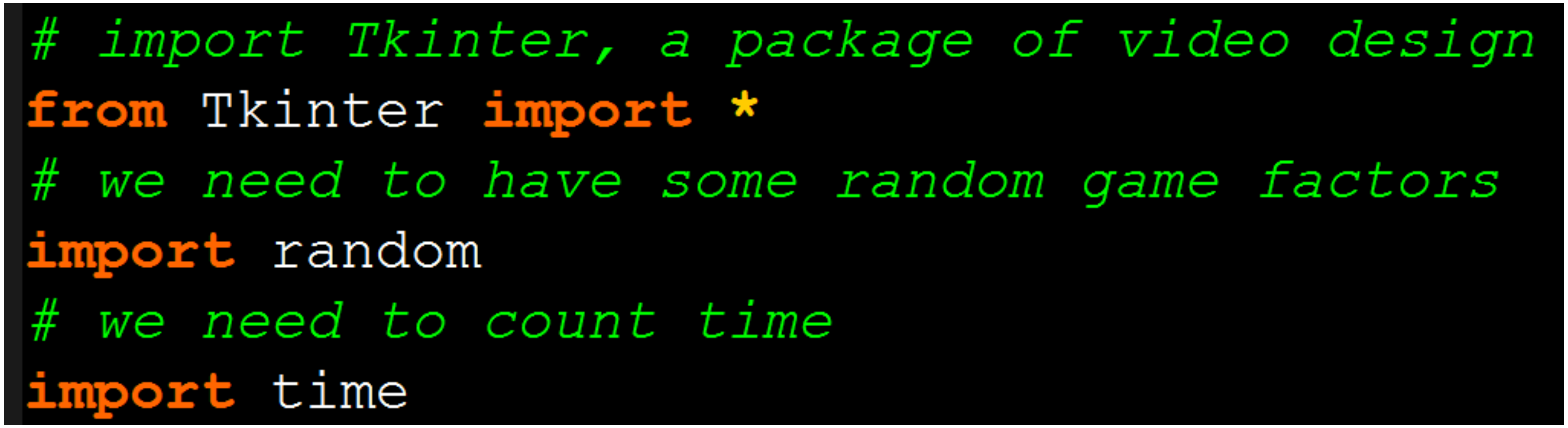
Review: Prelab

- Review what you have demoed



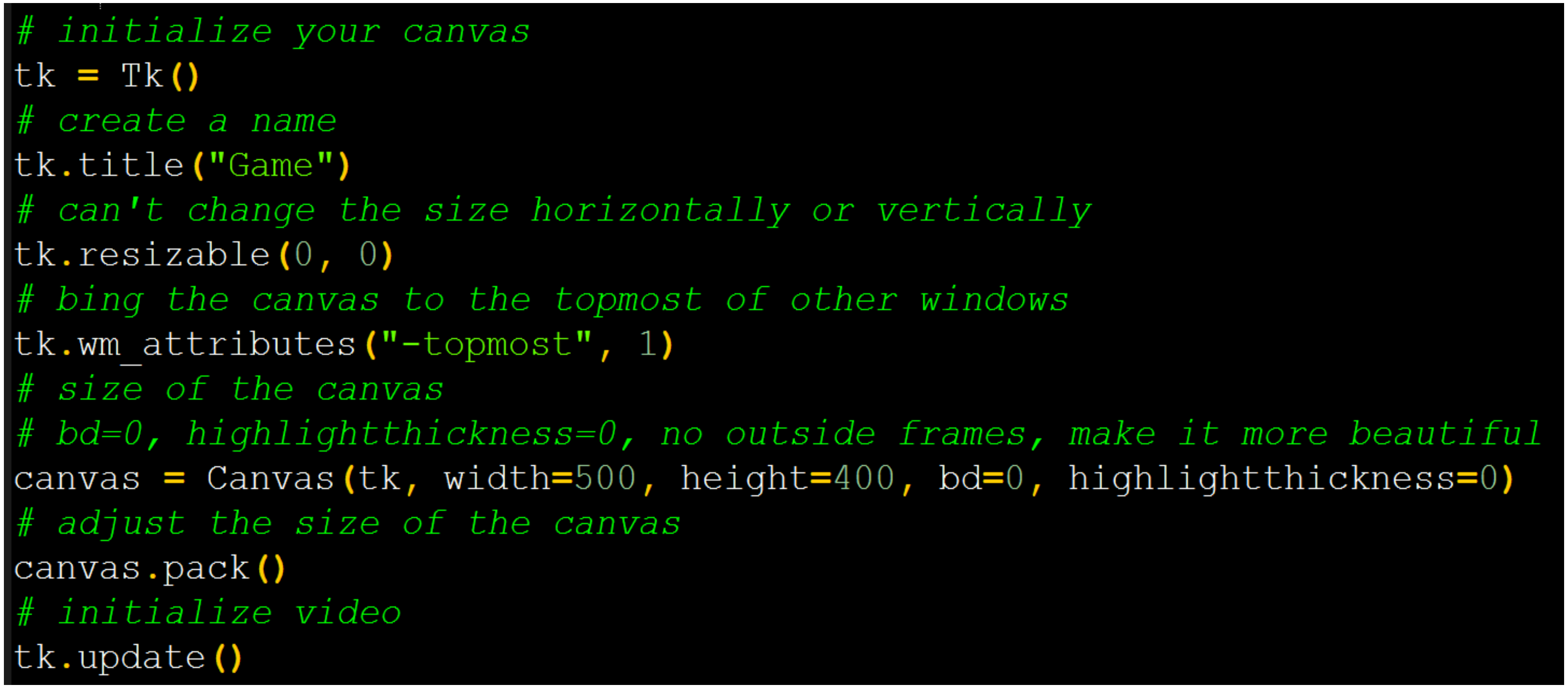
Import Necessary Packages

- Please search online what the difference is between "from Tkinter import *" and "import *"



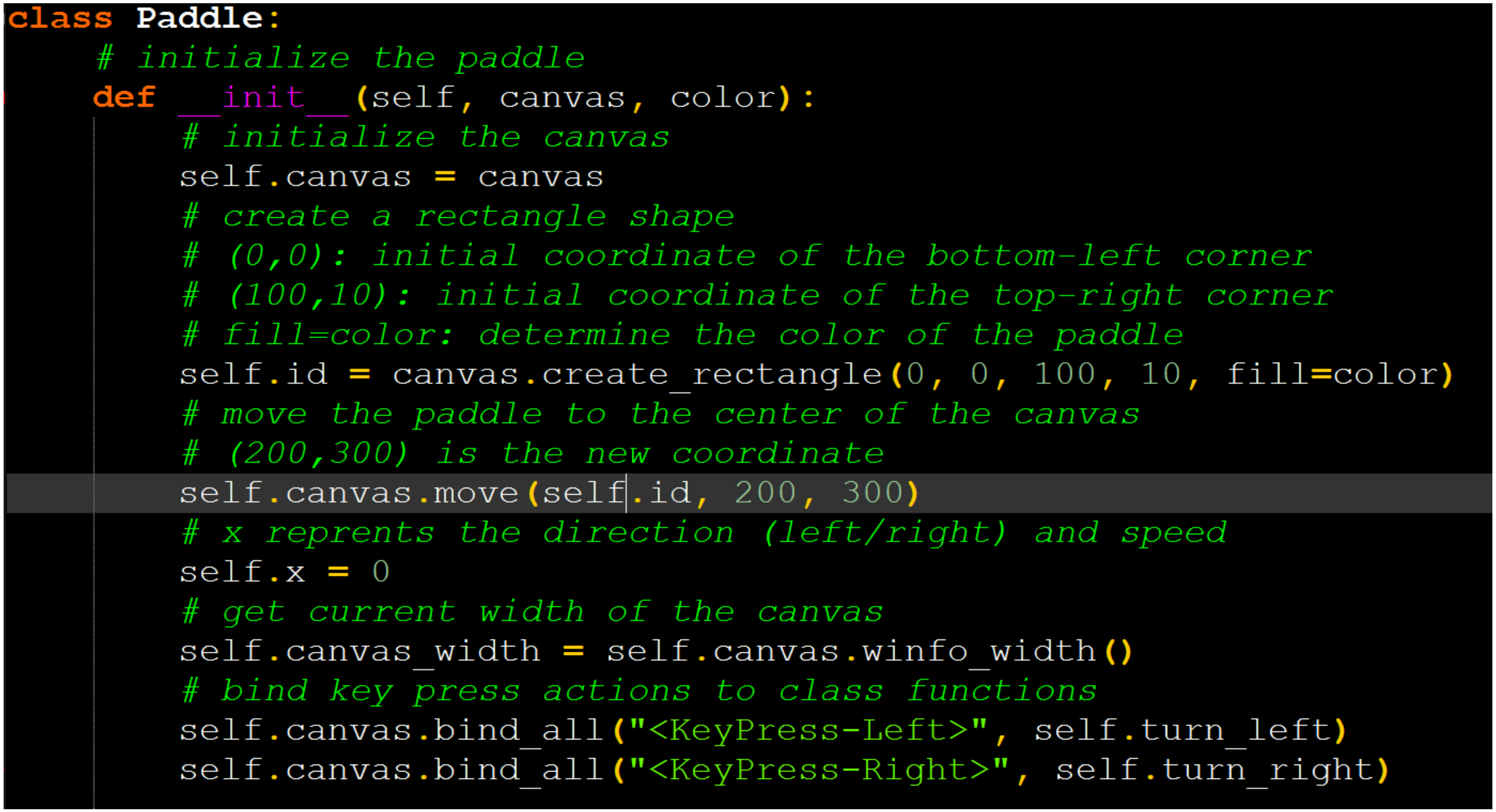
```python
# import Tkinter, a package of video design
from Tkinter import *
# we need to have some random game factors
import random
# we need to count time
import time
```

Create canvas



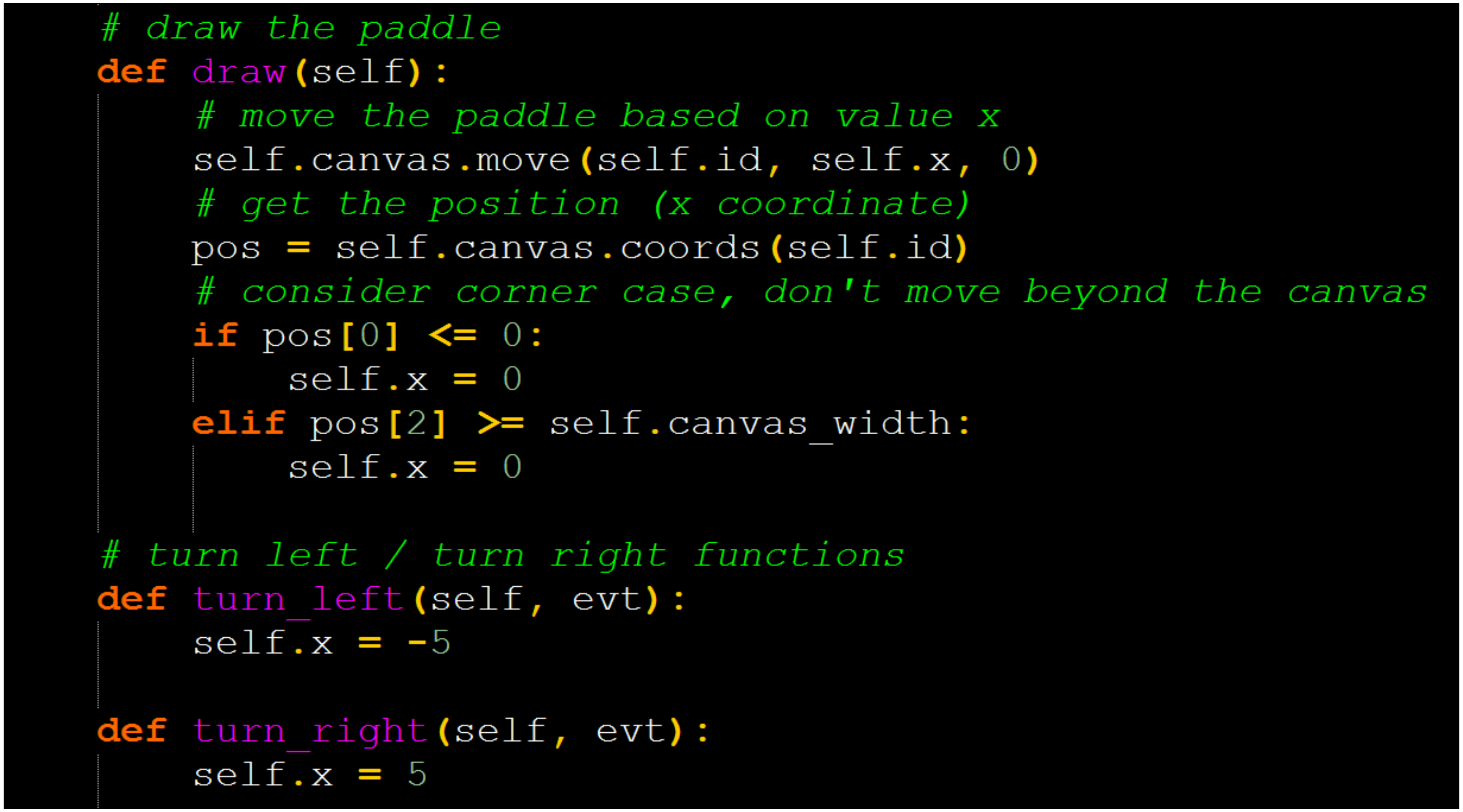
```
# initialize your canvas
tk = Tk()
# create a name
tk.title("Game")
# can't change the size horizontally or vertically
tk.resizable(0, 0)
# bing the canvas to the topmost of other windows
tk.wm_attributes("-topmost", 1)
# size of the canvas
# bd=0, highlightthickness=0, no outside frames, make it more beautiful
canvas = Canvas(tk, width=500, height=400, bd=0, highlightthickness=0)
# adjust the size of the canvas
canvas.pack()
# initialize video
tk.update()
```

Create paddle



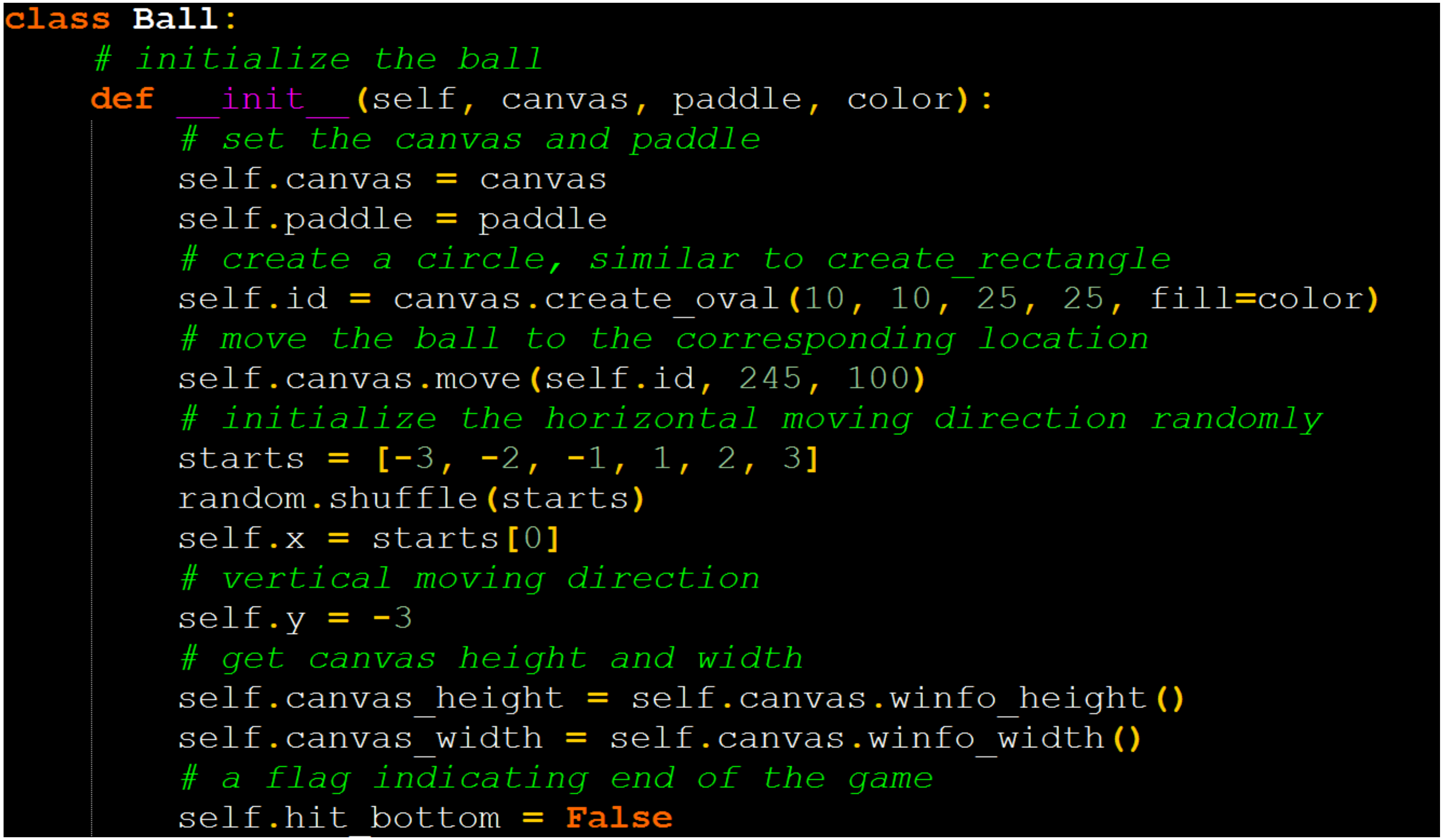
```python
class Paddle:
    # initialize the paddle
    def __init__(self, canvas, color):
        # initialize the canvas
        self.canvas = canvas
        # create a rectangle shape
        # (0,0): initial coordinate of the bottom-left corner
        # (100,10): initial coordinate of the top-right corner
        # fill=color: determine the color of the paddle
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
        # move the paddle to the center of the canvas
        # (200,300) is the new coordinate
        self.canvas.move(self.id, 200, 300)
        # x reprents the direction (left/right) and speed
        self.x = 0
        # get current width of the canvas
        self.canvas_width = self.canvas.winfo_width()
        # bind key press actions to class functions
        self.canvas.bind_all("<KeyPress-Left>", self.turn_left)
        self.canvas.bind_all("<KeyPress-Right>", self.turn_right)
```

Paddle functions



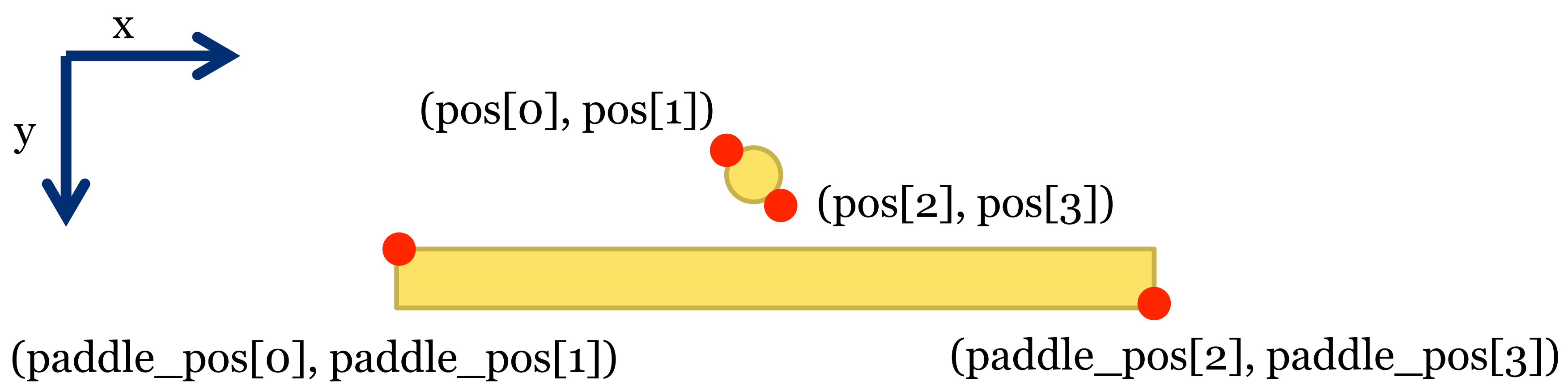
```python
    # draw the paddle
    def draw(self):
        # move the paddle based on value x
        self.canvas.move(self.id, self.x, 0)
        # get the position (x coordinate)
        pos = self.canvas.coords(self.id)
        # consider corner case, don't move beyond the canvas
        if pos[0] <= 0:
            self.x = 0
        elif pos[2] >= self.canvas_width:
            self.x = 0

    # turn left / turn right functions
    def turn_left(self, evt):
        self.x = -5

    def turn_right(self, evt):
        self.x = 5
```

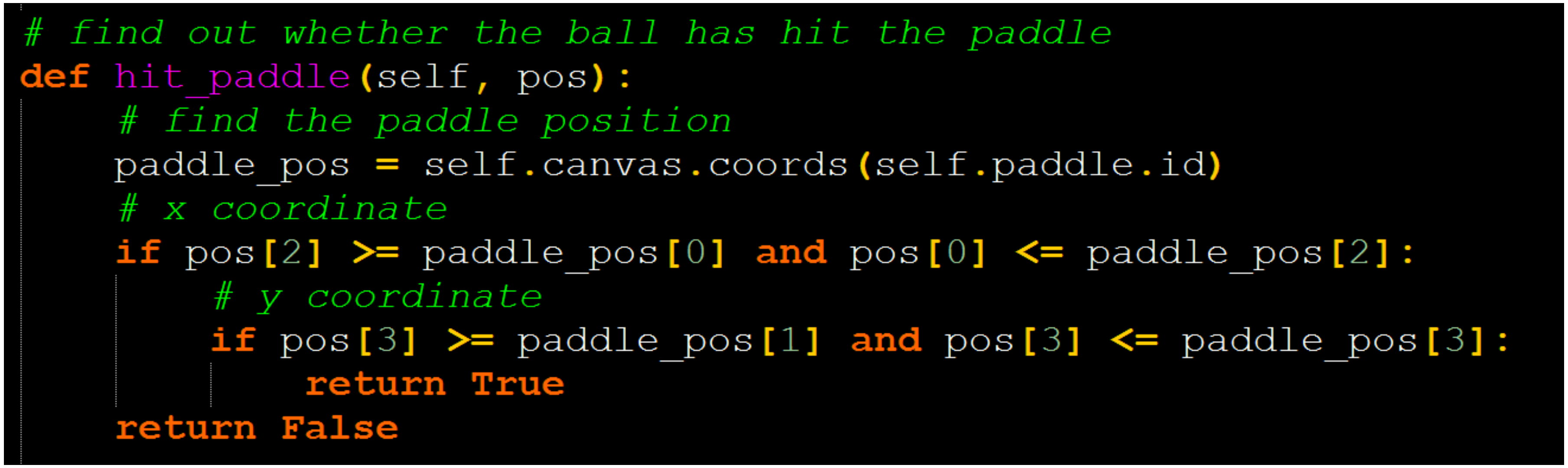
Create ball



```python
class Ball:
    # initialize the ball
    def __init__(self, canvas, paddle, color):
        # set the canvas and paddle
        self.canvas = canvas
        self.paddle = paddle
        # create a circle, similar to create_rectangle
        self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
        # move the ball to the corresponding location
        self.canvas.move(self.id, 245, 100)
        # initialize the horizontal moving direction randomly
        starts = [-3, -2, -1, 1, 2, 3]
        random.shuffle(starts)
        self.x = starts[0]
        # vertical moving direction
        self.y = -3
        # get canvas height and width
        self.canvas_height = self.canvas.winfo_height()
        self.canvas_width = self.canvas.winfo_width()
        # a flag indicating end of the game
        self.hit_bottom = False
```

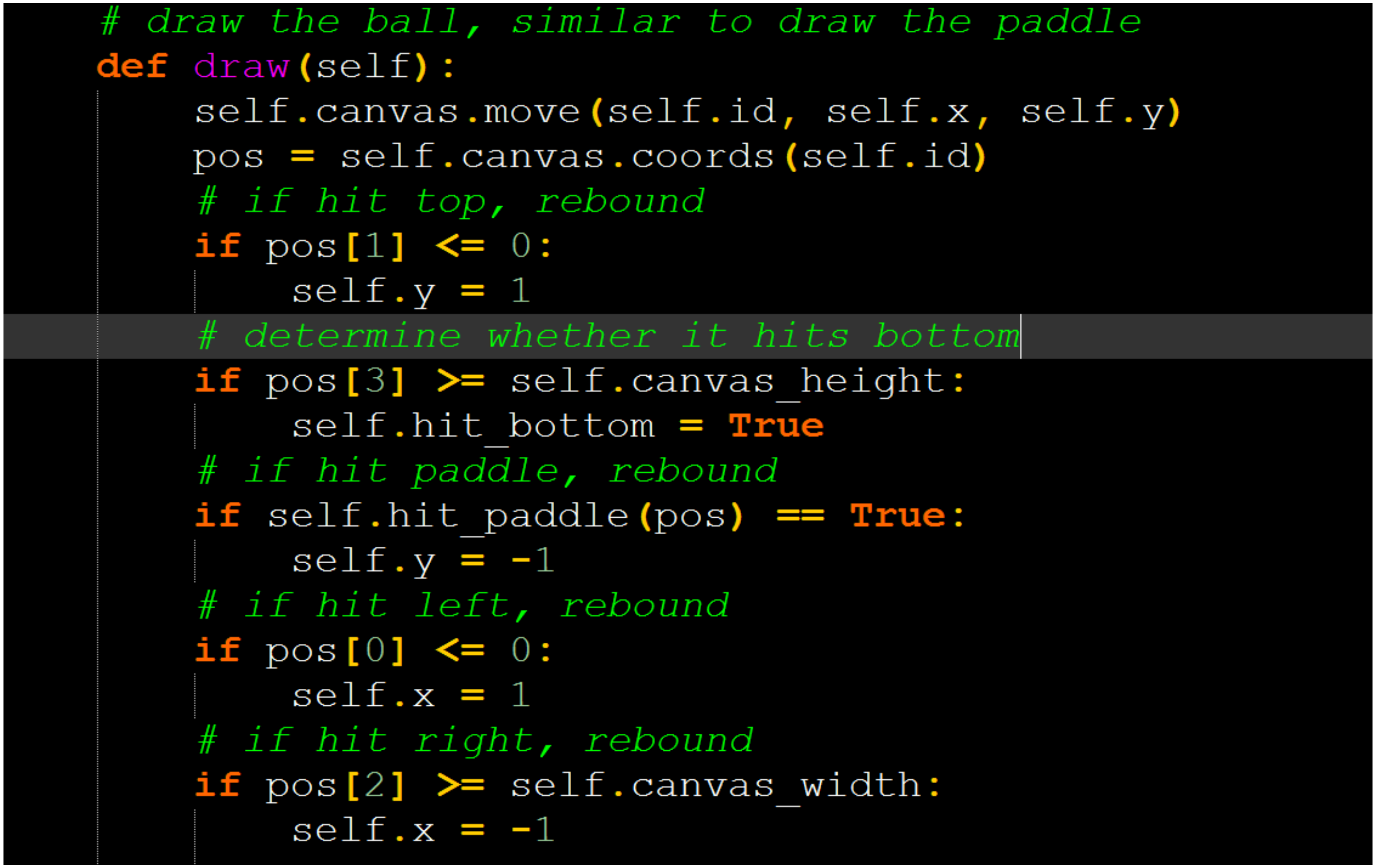
Ball functions


x
y
(pos[0], pos[1])
(pos[2], pos[3])
(paddle_pos[0], paddle_pos[1])
(paddle_pos[2], paddle_pos[3])



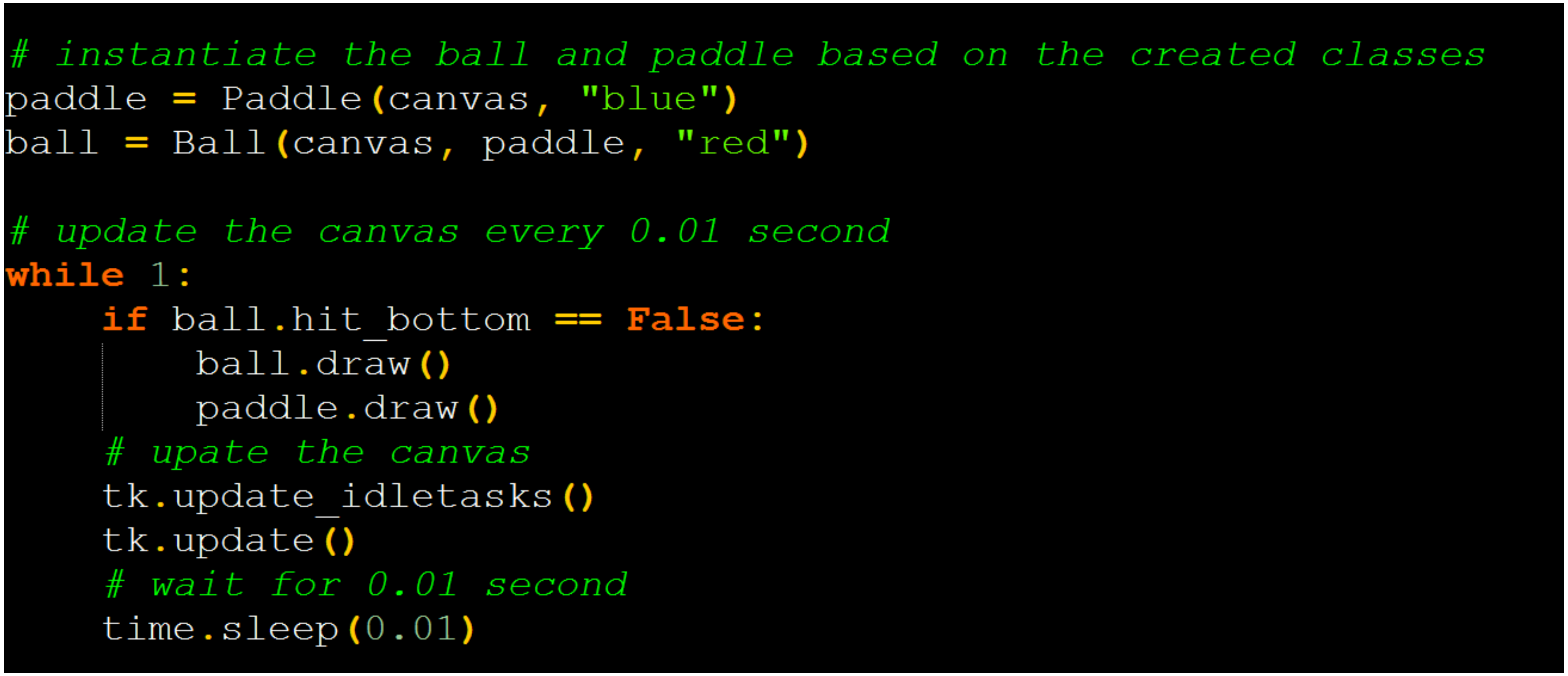
```
# find out whether the ball has hit the paddle
def hit_paddle(self, pos):
    # find the paddle position
    paddle_pos = self.canvas.coords(self.paddle.id)
    # x coordinate
    if pos[2] >= paddle_pos[0] and pos[0] <= paddle_pos[2]:
        # y coordinate
        if pos[3] >= paddle_pos[1] and pos[3] <= paddle_pos[3]:
            return True
    return False
```

Ball functions



```python
    # draw the ball, similar to draw the paddle
    def draw(self):
        self.canvas.move(self.id, self.x, self.y)
        pos = self.canvas.coords(self.id)
        # if hit top, rebound
        if pos[1] <= 0:
            self.y = 1
        # determine whether it hits bottom
        if pos[3] >= self.canvas_height:
            self.hit_bottom = True
        # if hit paddle, rebound
        if self.hit_paddle(pos) == True:
            self.y = -1
        # if hit left, rebound
        if pos[0] <= 0:
            self.x = 1
        # if hit right, rebound
        if pos[2] >= self.canvas_width:
            self.x = -1
```

Main function



```python
# instantiate the ball and paddle based on the created classes
paddle = Paddle(canvas, "blue")
ball = Ball(canvas, paddle, "red")

# update the canvas every 0.01 second
while 1:
    if ball.hit_bottom == False:
        ball.draw()
        paddle.draw()
    # upate the canvas
    tk.update_idletasks()
    tk.update()
    # wait for 0.01 second
    time.sleep(0.01)
```

Fun?

You can create your own game in lab 6

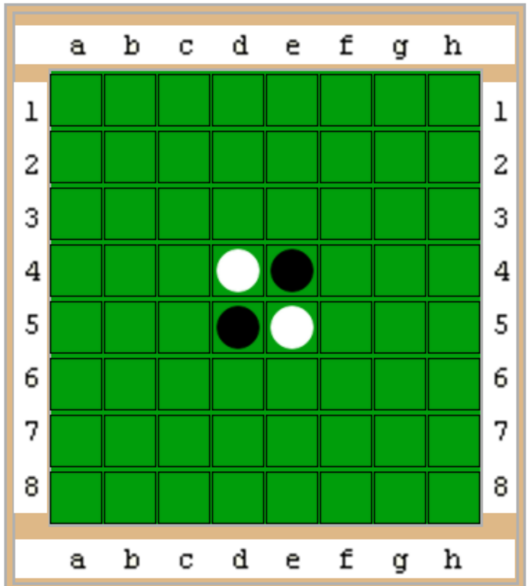
Reversi game

- Write a python program named “reversi_game.py” to build the basic structure of the reversi game.
- Please refer to the following website for the rules of reverse game:

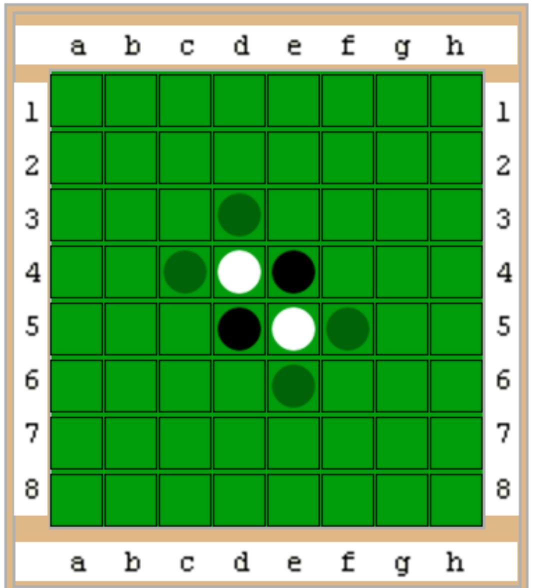
  https://en.wikipedia.org/wiki/Reversi

Rule for Reversi Game

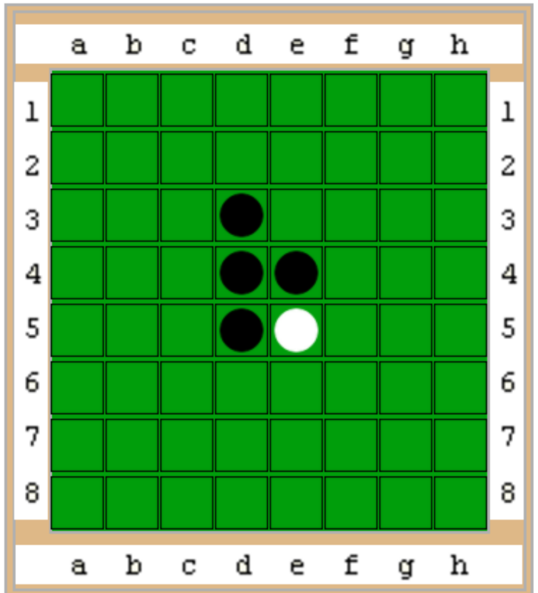
- **Reversi** is a strategy board game for two players.
- 8×8 uncheckered board
- 64 identical game pieces called *disks* with light and dark two sides. One side for each player



Starting position



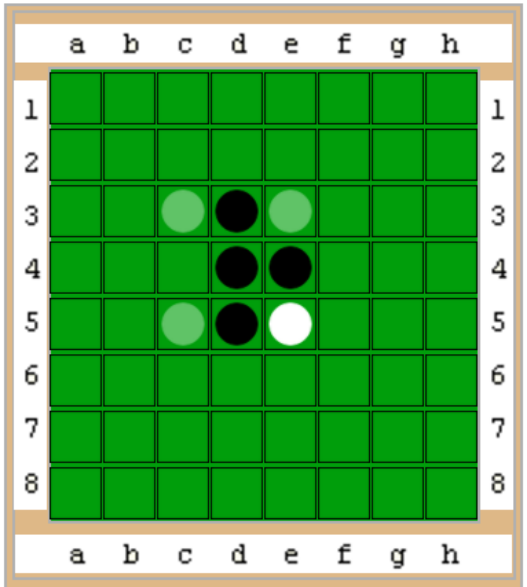
Where dark may play



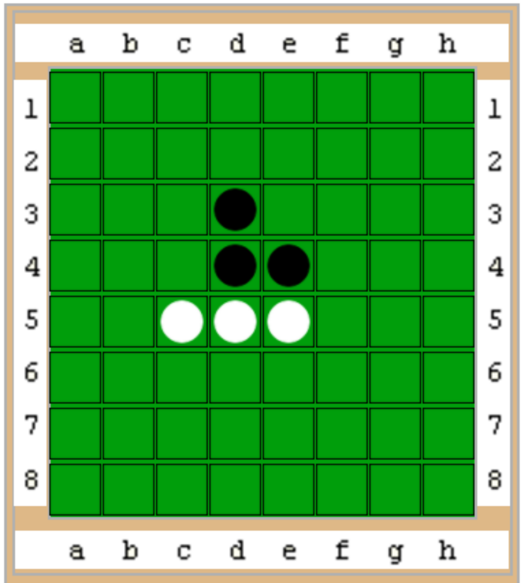
After dark play

Rule for Reversi Game

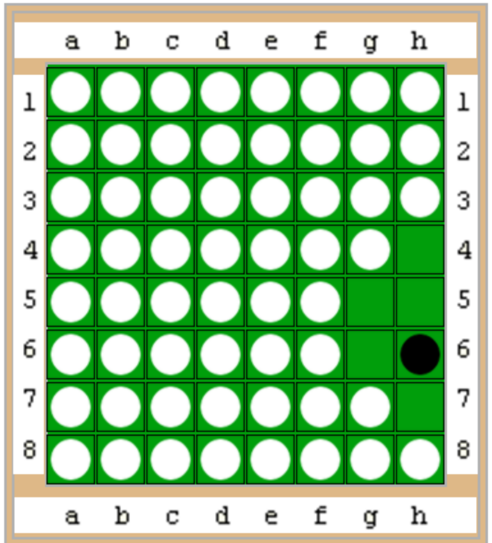
- Players take alternate turns.
- If one player can not make a valid move, play passes back to the other player.
- When neither player can move, the game ends.
  - the grid has filled up
  - neither player can legally place a piece in any of the remaining squares.



Where light may play



After light play

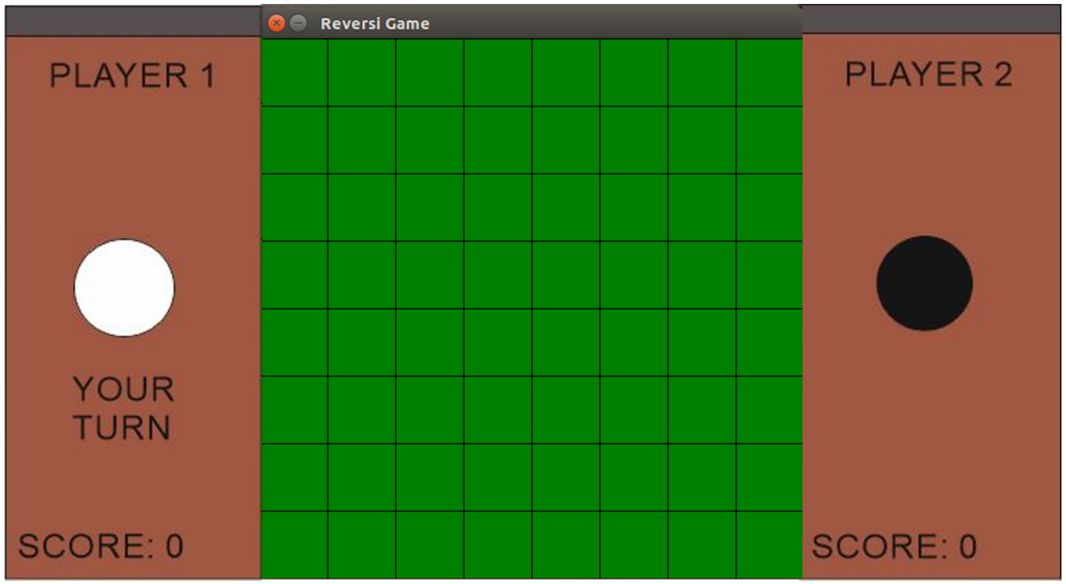


Vlasáková 1 – 63 Schotte (European Grand Prix Prague 2011)

Reversi game- Step 1

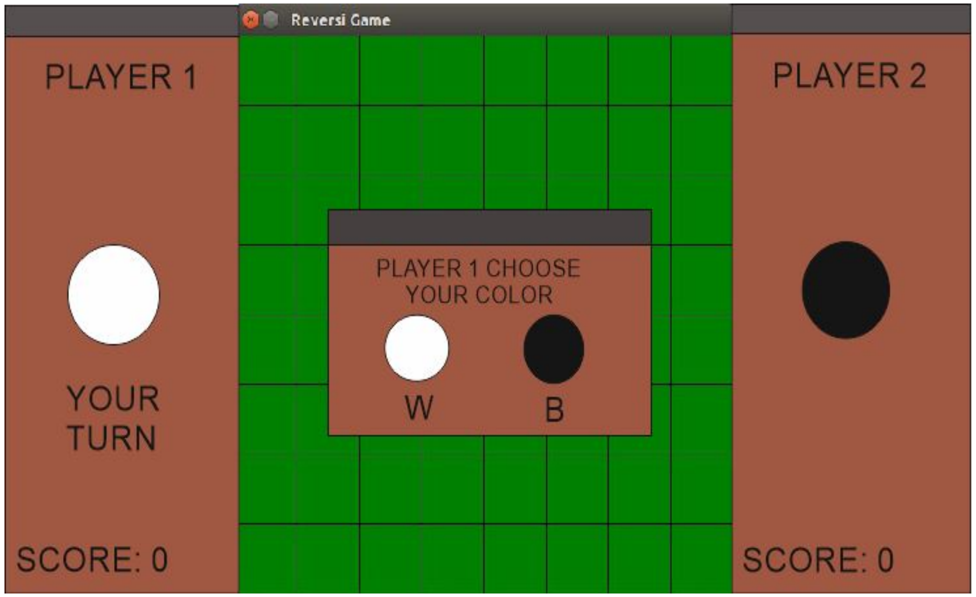
Create a canvas named “Reversi Game”.

- a board with 8*8 squares
- Each square should be 60*60 size and green color.
- side columns of the grid, which indicate the color which players have chosen along with their scores. It also indicates whose turn it is to play.



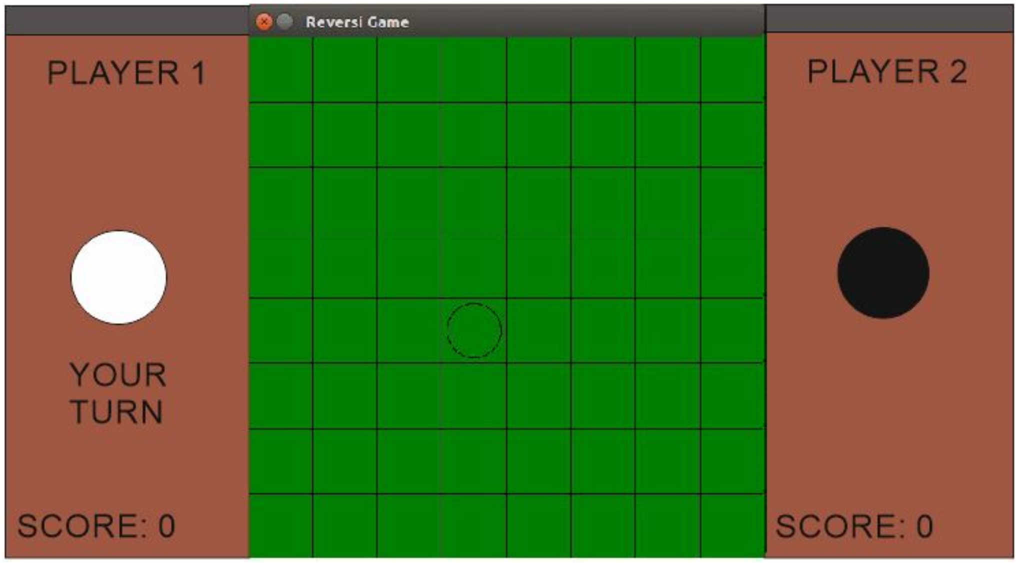
Reversi game- Step 2

At the beginning of the game, there should be a pop up window which allows the player to choose a color. If they type “W” then white is chosen, if they type “B” then black is chosen.



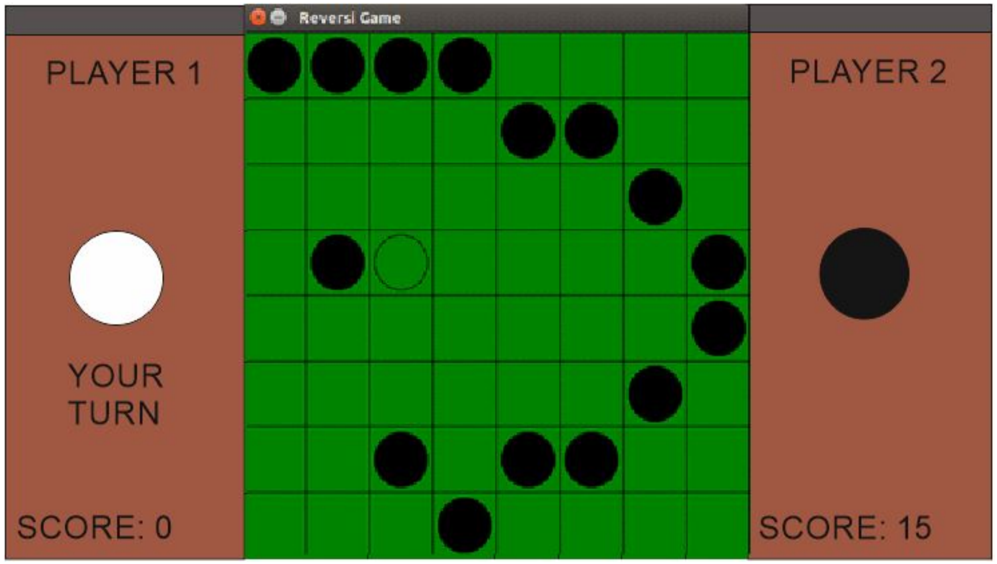
Reversi game- Step 3

- Create a moveable empty piece (by setting the color as “green”) with a size of 50*50.
- Move the piece by pressing the arrows in your keyboard. Make sure the piece is always at the center of a square.
- You are free to use any control method to determine the corner cases (i.e., What will happen when the moveable piece is at a right most square and you press <Right>? How about the bottom right corner?)



Reversi game- Step 4

- Add a function of the moveable piece so that when you press space, a fixed piece is created at the current location of the moveable piece (self.canvas.bind_all("<space>", ...)).
- The fixed piece should be "black" colored and also 50*50 size.
- Use any method to make sure that the moveable piece will never overlap with the fixed piece.



Reversi Game Part 1 Requirements

- Must create a class for each square in the board
- Must create a class for the moveable piece
- Must create a class for the fixed piece
- You can choose to combine the class of the moveable piece and the fixed piece or create a class with several subclasses

- **Submissions:**
- Submit one file “reversi_game.py” on black board.