

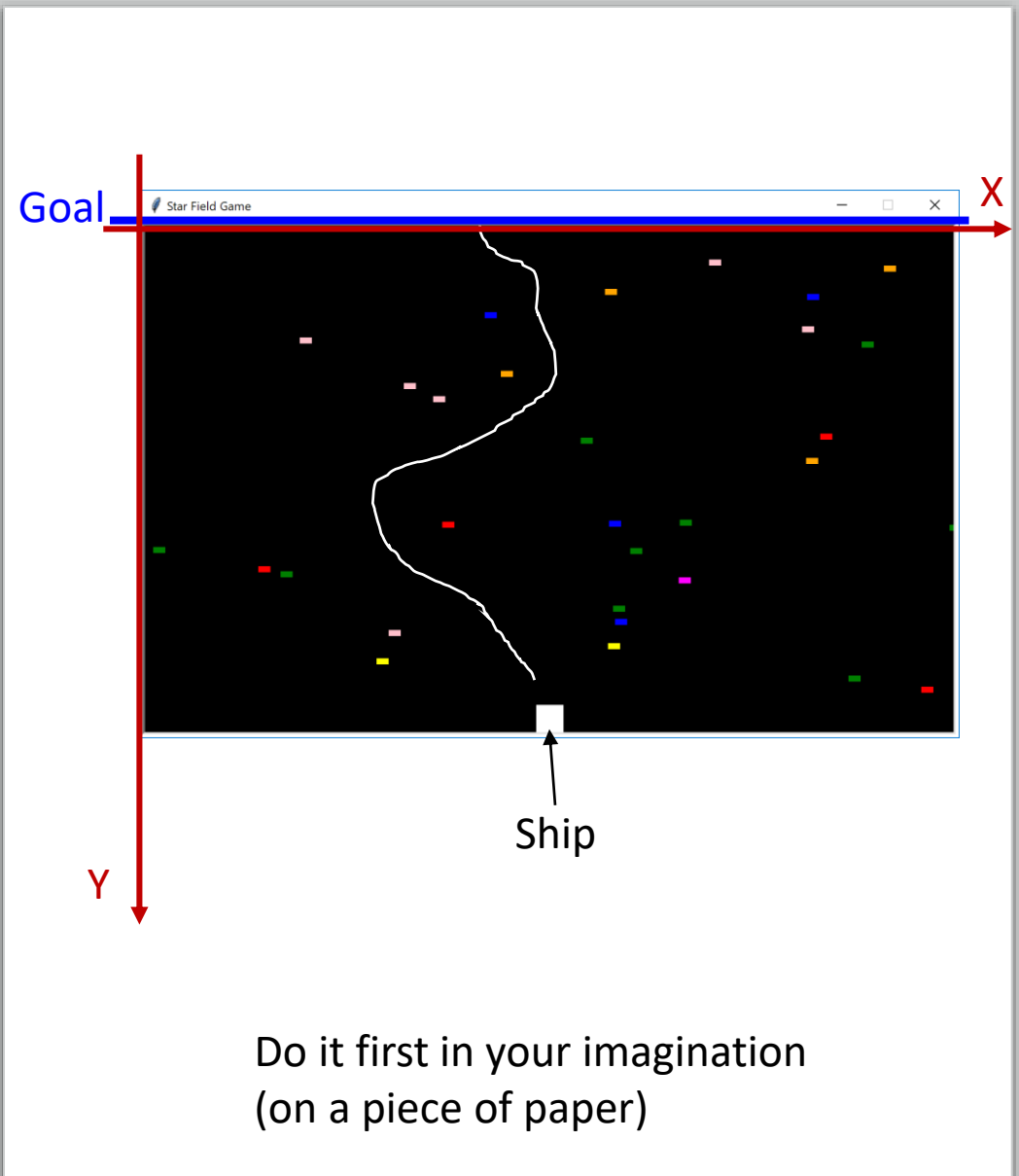
[star.py](#)  
[ship.py](#)  
[game.py](#)

# Programs as interacting objects

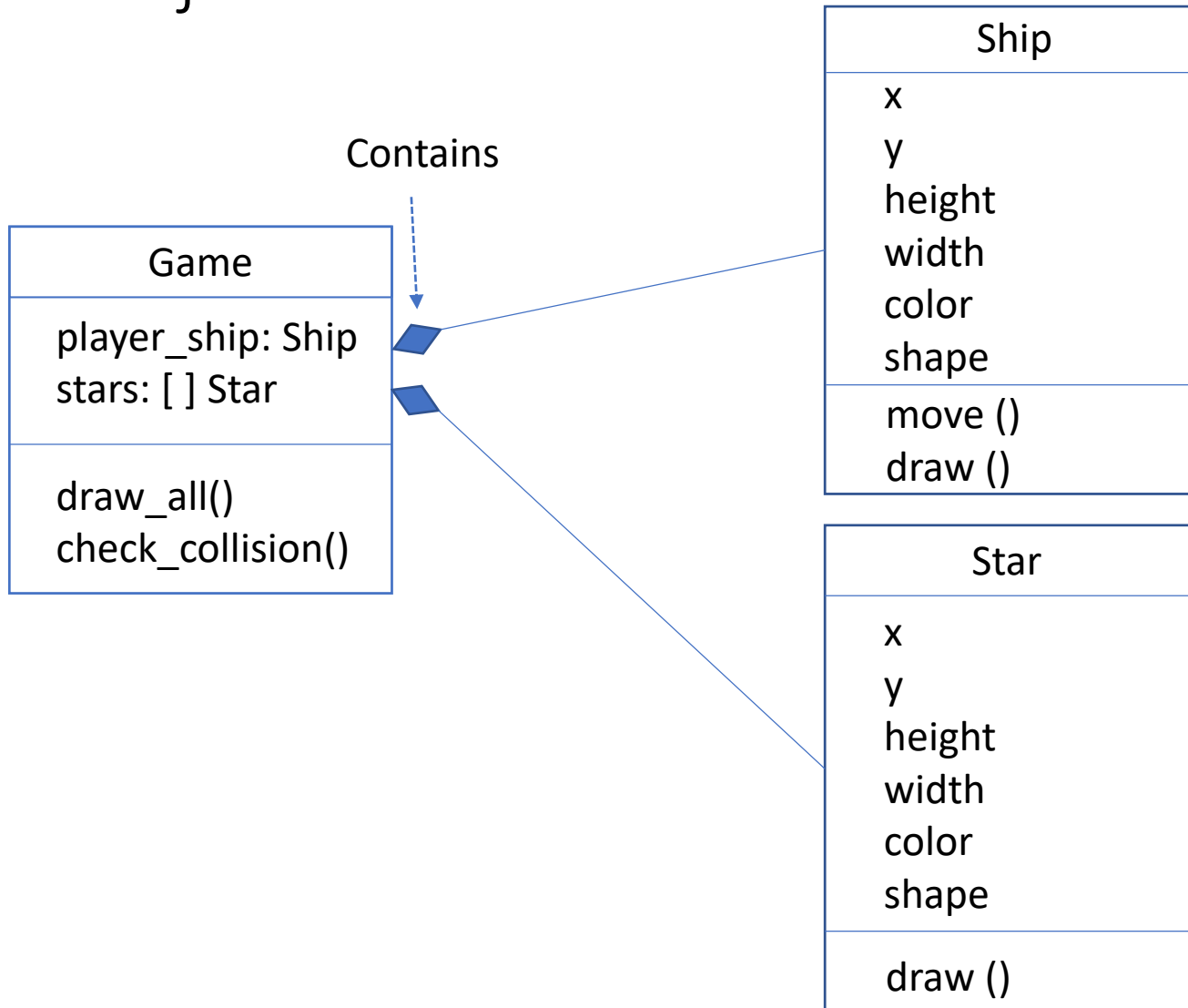
Lecture 07.04  
*by Marina Barsky*

# Star Field game - idea

- The goal of the game is to move the ship through the starfield without colliding with moving stars
- Stars move on their own
- Ship is controlled by arrow keys to avoid collisions with the stars
- If ship collides with a star – game over, player lost
- If ship makes it to the top – game over, player won



# What objects do we need?



# Outline of a *Star* type

- What are the fields?
- How the fields are initialized?
- How they are updated?
- What are the methods?

```
class Star():
    def __init__(self, canvas, w=12, h=6):
        self.canvas = canvas
        self.w = w
        self.h = h

        self.x = random.uniform(0, int(canvas.cget("width")))
        self.y = random.randint(0, int(canvas.cget("height")) - 40)
        self.dx = random.uniform(0.1, 0.6)

        self.color = random.choice(["red", "blue", "green",
                                    "yellow", "orange", "pink",
                                    "magenta"])

    def draw(self):
        self.x += self.dx
```

Star coordinate changes along X axis

# What should happen when a star moves out of the window?

```
class Star():
```

```
...
```

```
def draw(self):  
    self.x += self.dx
```

```
if self.x > canvas_w:  
    self.x -= canvas_w
```

```
print ("Star of shape {} at {},{}".format(  
        self.shape, self.x, self.y)
```

Star moves to the right (x increases)  
When  $x \geq 800$  (our canvas width) -  
what should happen with the star  
which we cannot see?

# Outline of a *Ship* type

```
class Ship:
    def __init__(self):
        self.w = 28
        self.h = 28
        self.x = 800/ 2 - self.w/2
        self.y = 600 - self.h
        self.delta = 5
        self.color = "white"
        self.shape = "rectangle"

    def move_right(self, event):
        self.x += self.delta

    def move_left(self, event): ...

    def move_up(self, event):
        self.y -= self.delta

    def move_down(self, event): ...

    def draw(self):
        print ("Ship of shape {} at {},{}".format(
            self.shape, self.x, self.y)
```

- What are the fields?
- What is initial position of the ship?
- When the position should be updated?
- What are the methods?

# Collaborate with *tkinter* canvas class

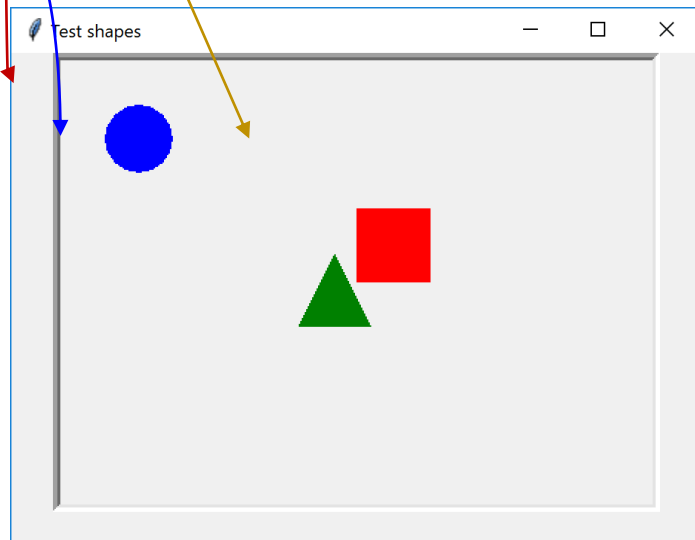
- In order to make the game playable, we need to connect it to the drawing library
- We need to be able to actually draw our moving stars and the ship on the screen and let the player to control the ship
- This can be done with many different packages:
  - Turtle
  - [Pygame](#)
  - ...
- We will use the built-in *tkinter* package
- Note that *tkinter* contains multiple **classes** which hide the complexity of actual drawing and user interaction under implemented methods, which we can simply call

# Drawing shapes: tkinter *canvas*

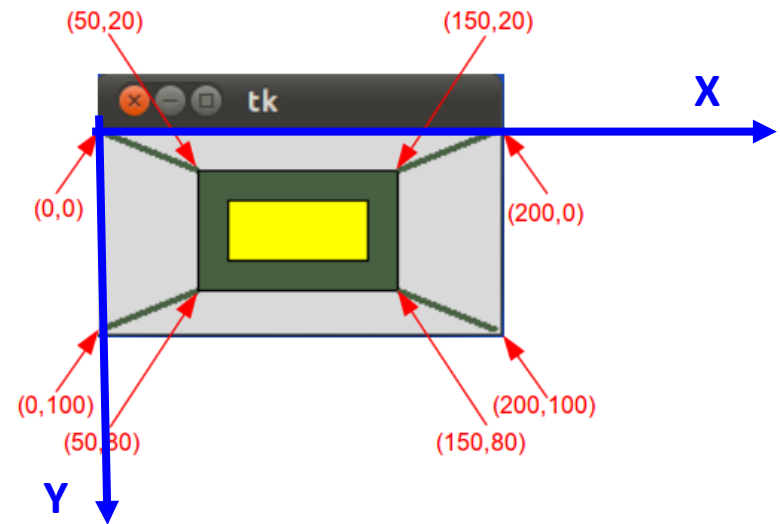
```
window = tk.Tk()
window.title("Test shapes")

frame = tk.Frame(window)
frame.pack()

canvas = tk.Canvas(frame)
canvas.pack()
```



*canvas inside frame inside window*



*canvas coordinate system*



# Graphical User Interfaces (Going gooey)

- A graphical user interface (GUI) allows a user to interact with a computer program using a keyboard or a pointing device that manipulates small pictures on a computer screen
- The small pictures are called *widgets*
- We refer to programs that use a graphical user interface as “GUI programs”

# GUI programs are different

- A GUI program is very different from a program that uses a command line interface which receives user input from typed characters on a keyboard
- Typically programs that use a **command line** interface perform **a series of tasks in a predetermined order and then terminate**
- **GUI** program creates the widgets that are displayed to a user and then it **simply waits for the user to interact with them**

# How to use tkinter properly

- The top-level class is *window* and can be created like this:

```
root = tkinter.Tk()
```

- The **GUI has to be implemented as a class** – in this case you will have access to all widgets that you add to your top window, as they will become the attributes of the same class, and can interact with each other
- Always place all widgets inside a frame – not the top window

```
import tkinter as tk
```

```
class Game:
```

```
    def __init__(self, num_stars=30):  
        self.window = tk.Tk()  
        self.window.title("Star Field Game")  
        self.window.resizable(0, 0)
```

We are going to call methods of the window object – so store it inside the Game object

We are never calling the Frame object to do anything for us – we do not need to store reference to it inside the Game object

```
        frame = tk.Frame(self.window, bd=5, relief=tk.SUNKEN)  
        frame.pack()
```

```
        self.canvas = tk.Canvas(frame, width=800, height=500)  
        self.canvas.configure(background='black')  
        self.canvas.pack()
```

# Adding widgets – nested under parent

```
import tkinter as tk
```

```
class Game:
```

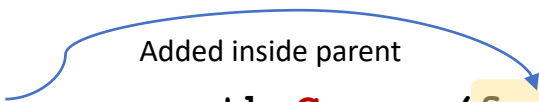
```
    def __init__(self, num_stars=30):  
        Top window
```

```
        self.root = tk.Tk()  
        self.root.title("Star Field Game")  
        self.root.resizable(0, 0)
```

```
        frame = tk.Frame(self.root, bd=5, relief=tk.SUNKEN)  
        frame.pack()
```



```
        self.canvas = tk.Canvas(frame, width=800, height=500)  
        self.canvas.configure(background='black')  
        self.canvas.pack()
```



Sample code for all tkinter widgets from your textbook:

[http://interactivepython.org/runestone/static/thinkcspy/static/Programs/all\\_user\\_input\\_widgets.py](http://interactivepython.org/runestone/static/thinkcspy/static/Programs/all_user_input_widgets.py)

# Making stars move on canvas

```
class Star():  
    def __init__(self, canvas, w=12, h=6):  
        ...  
        self.color = random.choice(["red", "blue", "green",  
                                    "yellow", "orange", "pink",  
                                    "magenta"])  
        self.shape = self.canvas.create_rectangle(  
            ...  
            )  
  
    def draw(self):  
        self.x += self.dx  
        self.canvas.move(self.shape, self.dx, 0)
```

Doing rectangular stars – they are simple

Stars move by themselves

# Outline of a Game class

```
from ship import Ship
from star import Star

class Game:
    def __init__(self, num_stars=30):
        self.stars = []
        for i in range(num_stars):
            self.stars.append(Star())

        self.main_ship = Ship()

    def collision_with_stars(self, ship):
        if collision of ship with any star:
            return True
        return False

    def game_loop(self):
        if self.collision_with_stars(self.main_ship):
            destroy_ship, game over - player lost
        elif self.player_ship.y <= 0: reached the top
            destroy_ship, game over - player won
        else:
            self.player_ship.draw()
            for i in range(len(self.stars)):
                self.stars[i].draw()
```

- What are the fields in class Game?
- What is the type of these fields?
- What are the methods?

# Game class: collision of rectangles

```
class Game:
    def __init__(self, num_stars=30):
        ...

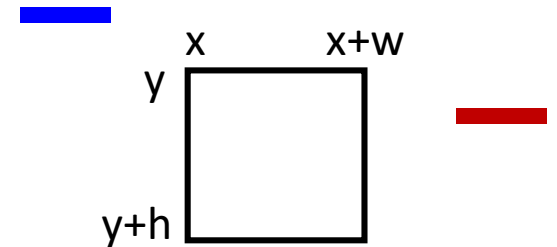
    def check_collision(self, ship):

        x = ship.x
        y = ship.y
        w = ship.w
        h = ship.h
        for star in self.stars:
            if not ((x + w) < star.x or
                    x > (star.x + star.w) or
                    (y + h) < star.y or
                    y > (star.y + star.h)):
                return True

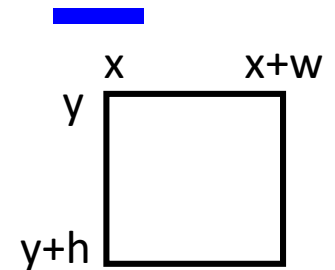
        return False

    def game_loop(self):
        ...
```

- When two rectangles collide (intersect)?
- How to implement collision detection?



No collision possible on X



No collision possible on Y

# Adding event handlers for *arrow key press* events

Event: pressing  
left arrow key

What to do:  
event handler

```
frame.bind('<Left>', self.player_ship.move_left)
frame.bind('<Right>', self.player_ship.move_right)
frame.bind('<Up>', self.player_ship.move_up)
frame.bind('<Down>', self.player_ship.move_down)
```



Finally: **game loop**

based on *event loop* implemented in *tkinter*

```
while True:
    if self.player_ship:
        if self.check_collision():
            self.canvas.delete(self.player_ship.shape)
            self.player_ship = None
            self.canvas.create_text(200, 200, fill="white",
                                     font="Times 20 italic",
                                     text="Game over you lost.")
        elif self.player_ship.y <= 0:
            self.canvas.delete(self.player_ship.shape)
            self.player_ship = None
            self.canvas.create_text(200, 200, fill="white",
                                     font="Times 20 bold",
                                     text="Game over you won!")
        else:
            self.player_ship.draw()

    for i in range(len(self.stars)):
        self.stars[i].draw()

self.root.update_idletasks() # redraw
self.root.update() # process events
```

# What did we gain by using Objects

- We can modify:
  - shape of stars
  - star speed
  - star direction
  - shape and color of the ship
  - add more objects – such as missiles that ship can shoot while moving through the star field
  
- All this without changing code in our Game class

# Exercise: messing with the game

- First, play the game and lose
- Second, play the game and win: make all the way to the top without colliding with the stars. If you want an easy win, in *main()* call Game constructor with 5 stars instead of 30
- Modify the *Star* class so that the width of the star is also selected at random: between 20 and 100 pixels
- Try to modify the initial position of the *main\_ship* by passing a start X coordinate to the Ship class in the constructor of a *Game* class.
- Finally, add the second *Ship* object to the game (*helper\_ship*)
- Add event handlers to move *helper\_ship* using WASD keys.
- Add collision testing logic for the second ship as well.
- Update the test for winning condition (for example if one of the ships makes it to the top, player wins).
- Now play the game with two ships and win!