

게임 프로그래밍 입문

Introduction to Game Programming

Kyung Hee University
Daeho Lee



Search for eBooks and Videos:

Enter your search terms...[Search](#)

Daeho Lee

[Account Details](#)[My Order History](#)[My eBooks](#)[eBook Sharing History](#)[My Videos](#)[My Courses](#)[My Card Details](#)[Log Out](#)

My eBooks



Python Game Programming By Example [eBook]

Alejandro Rodas de Paz, Joseph Howse

Price

\$10.00

Order Reference

PAC-18-23791011-2787595

Order Date

5 March 2018

[Read Online](#)[Upgrade to Print](#)[PDF](#)[ePub](#)[Mobi](#)[Code Files](#)[Kindle](#)

◆ Python

- ◆ Tkinter, WxPython – GUI
- ◆ Cocos2D – open source software framework for building games
- ◆ NumPy – multi-dimensional arrays/matrices, high-level mathematical functions
- ◆ Pygame, PyOpenGL – 3D games
- ◆ Pymunk, Chipmunk2D – 2D physics engine
- ◆ OpenCV – computer vision library

-1. Python programming

Kyung Hee University
Daeho Lee

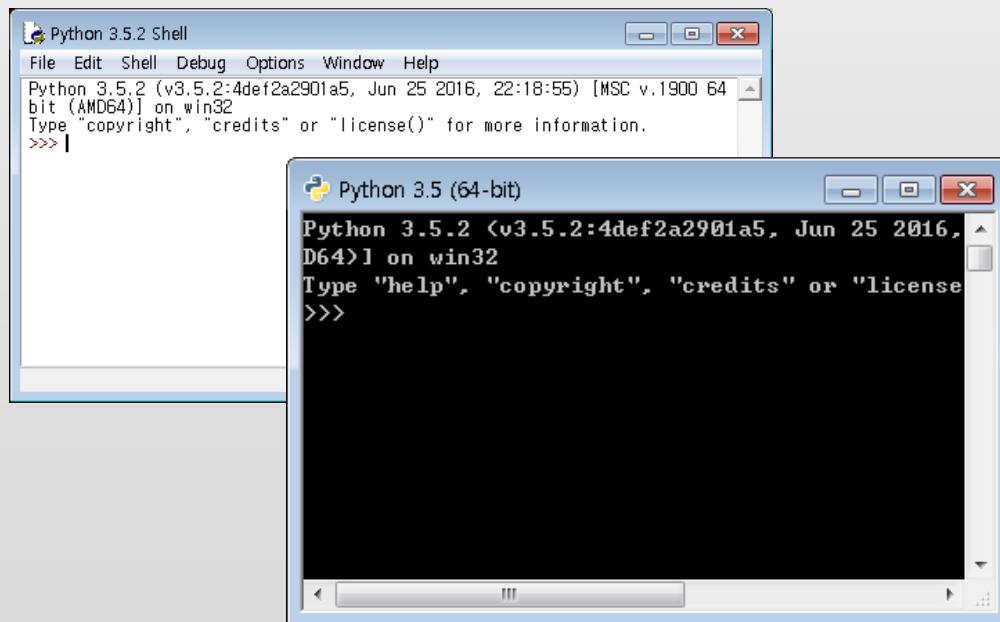
❖ <https://www.python.org/downloads/>



- ◆ High-level language
- ◆ Interpreter
- ◆ Features
 - Readability
 - Significant whitespace
 - Dynamic type
 - Automatic memory management
 - Package libraries including GUI, multimedia, database, automation, image processing, game programming,

◆ IDLE

- integrated development and learning environment
- IDE (integrated development environment)



◆ Python 2 & Python 3

- print
 - ✓ `print "Hello"; print ("Hello")`
 - ✓ `print ("Hello")`
- /
 - ✓ $3/4 \rightarrow 0, 3.0/4 = 0.75$
 - ✓ $3/4 \rightarrow 0.75$
- input
 - ✓ `ex = raw_input("Input: ")`
 - ✓ `ex = input("Input: ")`
- Long, int → int
- Unicode
 - ✓ `str, unicode`
 - ✓ `str`

◆ Variables, expressions and statements

- int, float, str, bool
- >>> name = 'Kyung Hee'
- Variable name: letters, numbers and _, start with letter or _
- Keywords
 - ✓ and del from None True as elif
 - ✓ global nonlocal try assert else
 - ✓ if not while break except import or
 - ✓ with class False in pass yield continue
 - ✓ finally is raise def for lambda
 - ✓ return

◆ Operators and operands

- +, -, *, /, ** (exponentiation), // (quotient), % (remainder)
- Operator precedence
 - ✓ (), **, unary, (*, /, //, %), (+, -), not, and, or
- String operators
 - ✓ +, *

◆ Type conversion functions

- int(), float(), str()

◆ Input and print

- input
- print

◆ Boolean

- True, False
- ==, !=, >, >=, <, <=, is, is not

◆ Logical operators

- and, or, not

◆ Comments

- #

◆ Input

```
x = int(input('x = '))
```

◆ help

```
>>> help (print)
```

```
Help on built-in function print in module builtins:
```

```
print(...)
```

```
    print(value, ..., sep=' ', end='\n', file=sys.stdout,  
flush=False)
```

Prints the values to a stream, or to sys.stdout by default.

Optional keyword arguments:

file: a file-like object (stream); defaults to the current sys.stdout.

sep: string inserted between values, default a space.

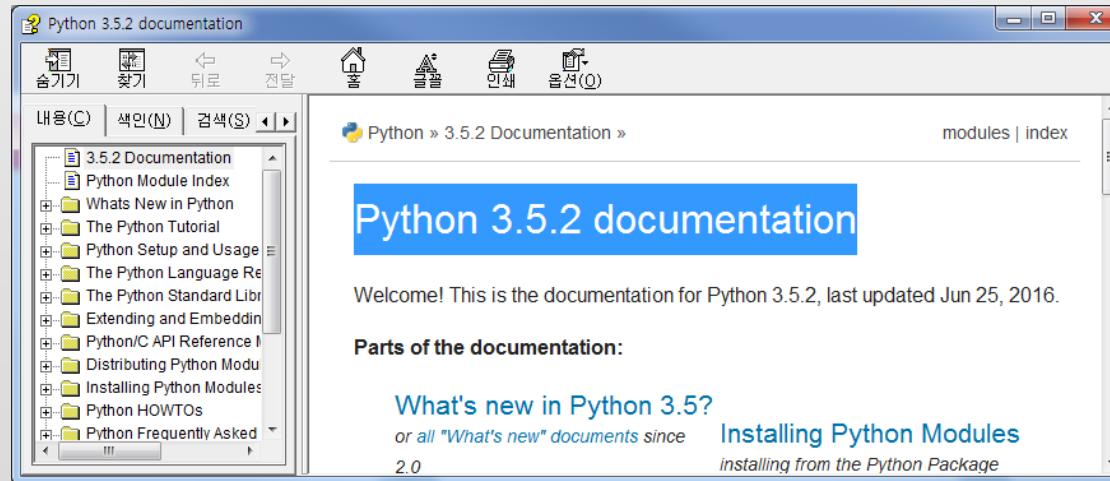
end: string appended after the last value, default a newline.

flush: whether to forcibly flush the stream.

```
>>> help (random.random)
Help on built-in function random:
```

random(...) method of random.Random instance
random() -> x in the interval [0, 1].

◆ Python 3.x.x documentation



◆ 섭씨(Celsius) 온도를 입력 받아서 화씨(Fahrenheit) 온도로 계산하여 출력하는 프로그램을 작성하라.

➤ $F = \frac{9}{5}C + 32$

◆ 두 정수를 입력 받아서 몫(quotient)과 나머지(remainder)를 계산하여 출력하는 프로그램을 작성하라.

◆ Conditional statements

```
➤ if (condition) :  
    (expression) #indented  
  
➤ if (condition1) :  
    (expression1) #indented  
elif (condtion2) :  
    (expression2) #indented  
else :  
    (expression3) #indented
```

```
➤ if (condition1) :  
    (expression1) #indented  
else :  
    if (condition2) : #indented  
        (expression2) #indented  
    else :  
        (expression3) #indented
```

◆ while

- `while (condition) :`
 `(statement) #indented`

◆ for

- `for variable in variableList :`
 `(statement) #indented`
- `continue, break`

◆ 0, 양수, 음수 판별

```
input_n = input("Input: ")
n = int(input_n)
if(n > 0) :
    print ("Positive")
elif (n < 0) :
    print ("Negative")
else :
    print("Zero")
```

◆ 0-10까지 합

```
sum = 0
for i in range (1,11) :
    sum += i
print (sum)
```

◆ range

```
>>> help (range)
Help on class range in module builtins:
```

```
class range(object)
    range(stop) -> range object
    range(start, stop[, step]) -> range
object
```

◆ Function calls

- `type(71), max('Hello world')`
- `len('Hello world'), int('56')`

◆ Math functions (`pi, sin(), cos(), tan(), sqrt()`)

- `import math #import math module`
`x = math.pi`
- `import math as ma`
`x = ma.pi`
- `from math import pi`
`x = pi`
- `from math import *`
`x = cos(pi)`

◆ 2차 방정식

```
import math  
  
input_a = input("a: ")  
input_b = input("b: ")  
input_c = input("c: ")  
a = float(input_a); b = float(input_b); c =  
float(input_c);  
D = b**2 - 4*a*c;  
  
if D >= 0 :  
    r1 = (-b + math.sqrt(D)) / (2*a)  
    r2 = (-b - math.sqrt(D)) / (2*a)  
    print(r1, r2)
```

◆ Random functions

➤ import random

```
x = random.random() # [0, 1)
```

```
y = random.randint(6, 10) # [a, b]
```

◆ Function definition

➤ def fn1() :

```
    (statement) #indented
```

```
    [return value] #indented
```

➤ def fn2(n1, n2) :

```
    (statement) #indented
```

```
    [return value] #indented
```

◆ 가변 파라메터

```
def Average(s1, *s) :  
    sum = s1; cnt = 1  
    for n in s:  
        sum += n; cnt=cnt+1  
    return sum/cnt
```

```
a = Average(1)  
print(a)  
a = Average(1, 2)  
print(a)  
a = Average(1, 2, 3, 4, 5)  
print(a)
```

◆ String

➤ String slices

- ✓ `s = 'Python string'`
- `s[0:5]` #'Pytho', 0부터 4까지
- `s[6:9]` #' st', 6부터 8까지
- `s[:3]` #'Pyt', 0부터 2까지
- `s[3:]` #'hon string', 3부터 마지막까지
- `s[-7:-1]` #' strin', -7에서 -2까지

➤ Strings are immutable

➤ Format operator

- ✓ `x = 10`
- `'x = %d' % x`
- ✓ `%d, %g, %s` # 정수, 실수, 문자열
- ✓ `print('sum %f' % a)`

◆ Lists: sequence of values

- [10, 20, 30]
- ['abc', 'def']
- ['abc', 2, 3.5, [20, 30]]
- Lists are mutable
 - ✓ ex = [10, 20]
ex[0] = 90
ex[-1] = 100
- List operators: +, *, del, []
- List slices: [:]
- List methods: append(), extend(), sort(),
pop(), remove()
- Aliasing (reference)

◆ List example 1

```
a = ['abc', 2, 3.5, [20, 30]]  
print (a[0]) #abc  
print (a[1]) #2  
print (a[2]) #3.5  
print (a[3]) #[20,30]  
print (a[3][0]) # 20  
print (a[3][1]) # 30
```

```
a[1] = 10  
a[3][0] = 100  
print (a) # ['abc', 10, 3.5, [100, 30]]
```

◆ List example 2

```
a = [1, 2, 3]
b = [4, 5, 6]
c = a + b
print(c) # [1, 2, 3, 4, 5, 6]
d = a*3
print(d) # [1, 2, 3, 1, 2, 3, 1, 2, 3]
del a[1]
print(a) # [1, 3]
```

◆ List example 3

```
a = [1, 2, 3]
b = a
print(b) # [1, 2, 3]
b[0] = 10
print(a) # [10, 2, 3]
print(b) # [10, 2, 3]
a[0] = 100
print(a) # [100, 2, 3]
print(b) # [100, 2, 3]
```

◆ List example 4

```
a = [1, 2, 3]
b = a[0:] # b = list(a)
```

```
print(b) # [1, 2, 3]
b[0] = 10
print(a) # [1, 2, 3]
print(b) # [10, 2, 3]
a[0] = 100
print(a) # [100, 2, 3]
print(b) # [10, 2, 3]
```

◆ List example 5

```
list1 = [0]*2
list2 = [list1]*3
list3 = [[0]*2]*3
print(list1) # [0, 0]
print(list2) # [[0, 0], [0, 0], [0, 0]]
print(list3) # [[0, 0], [0, 0], [0, 0]]
list1[0] = 1
list3[0][0] = 1
print(list1) # [1, 0]
print(list2) # [[1, 0], [1, 0], [1, 0]]
print(list3) # [[1, 0], [1, 0], [1, 0]]
```

◆ List example 5

```
print([0 for i in range(3)]) # [0, 0, 0]
print([i for i in range(3)]) # [0, 1, 2]
print([x for x in [1,2,3] if x in [1,3]]) #
[1, 3]
```

```
list4 = [[0]*2 for i in range(3)]
print(list4) # [[0, 0], [0, 0], [0, 0]]
list4[0][0] = 1
print(list4) # [[1, 0], [0, 0], [0, 0]]
```

◆ Dictionaries

- Index of lists should be integers
- In dictionaries, the indices can be (almost) any type
- ```
eng2sp = dict()
eng2sp['one'] = 'uno'
```
- ```
eng2sp = {'one': 'uno', 'two': 'dos',
'three': 'tres'}
```
- ```
vals = list(eng2sp.values())
```
- ```
indices = list(eng2sp)
```

◆ Dictionary example

```
eng2sp = { 'one': 'uno', 'two': 'dos', 'three':  
          'tres' }  
  
print (eng2sp) # {'two': 'dos', 'one': 'uno', 'three': 'tres'}  
print (eng2sp['one']) # uno  
eng2sp['two'] = 'DOS'  
print (eng2sp) # {'two': 'DOS', 'one': 'uno', 'three': 'tres'}  
vals = list(eng2sp.values())  
print (vals) # ['DOS', 'uno', 'tres']  
indices = list(eng2sp)  
print (indices) # ['two', 'one', 'three']
```

◆ Tuples

- Sequence of values much like list
- Tuples are immutable
- `t = 'a', 'b', 'c', 'd', 'e'`
- `t = ('a', 'b', 'c', 'd', 'e')`
- `t = ('a',) → tuple`
- `t = ('a') → str`
- `t = tuple('abcde')`
- `[] , [:]`

- ◆ $n \times n$ 마방진(magic square)을 구성하여 출력하는 프로그램을 작성하라(n 은 홀수, $3 \leq n \leq 15$)

✓ Output>>> size: 5

15 8 1 24 17

16 14 7 5 23

22 20 13 6 4

3 21 19 12 10

9 2 25 18 11

◆ Object-oriented programming

- class className :
 static_variable = xxx
 def methodName (self) :
 self.variable = xxx;

- Constructor and destructor

```
def __init__(self)  
def __del__(self)
```

- x = className()
x.methodName()

- del x

```
class A:  
    x = 0  
    def __init__(self):  
        print ('init')  
    def __del__(self):  
        print ('del')  
  
a = A()  
del a
```

```
class A:  
    x = 0  
    def __init__(self):  
        self.x = 20  
        self.__x = 30  
    def fn(self):  
        print(self.__x)  
a = A()  
b = A()  
  
A.x += 1  
print(a.x) # 20  
a.fn() # 30  
print (A.x) # 1
```

➤ Inheritance

```
class baseClassName(object) :  
    class className(baseClassName) :  
        # object is made ancestor of all classes  
        # Python 3 "class base" is equivalent to  
        # "class base(object)"
```

- ✓ super() # accessing parent class members
 - super(Derived, self).xxx
 - super().__init__ # Python 3

```
class D(A) :  
    def __init__(self, x = 0):  
        super(D, self).__init__()
```

```
class A:  
    x = 0  
  
    def __init__(self):  
        print ('init')  
  
    def __del__(self):  
        print('del')  
  
class D(A):  
    def __init__(self, x = 0):  
        super(D, self).__init__()  
        self.x = x  
  
d1 = D()  
print(d1.x)  
  
d2 = D(10)  
print(d2.x)
```

- ◆ if __name__ == '__main__' :

- ◆ Python GUI
 - tkinter
 - ✓ Tcl/Tk-based GUI module
 - wxPython
 - ✓ Based on C++ GUI toolkit

- ◆ Object와 Ball 클래스를 아래와 같이 작성하라.
- Object는 x, y, color의 멤버를 가지며, color는 생성자에사 'white'로 초기화 된다.
 - Ball은 Object의 child 클래스이며 반지름 (radius)의 멤버를 가진다. 모든 데이터 멤버는 생성자를 통하여 초기화 된다.
 - 두 클래스의 동작을 확인할 수 있는 예제 코드도 작성하라.

0. Snake game

Kyung Hee University
Daeho Lee

Snake game (1)

```
import random
import os
import time
import msvcrt

class Snake:
    def __init__(self, n):
        self.length = n
        self.head = []
        self.tail = []
```



Snake game (2)

```
class SnakeGame:  
  
    direction = {"LEFT": -2, "DOWN": -1, "NON_DIR": 0, "UP": 1, "RIGHT": 2}  
    sprite = {"EMPTY": 0, "BODY": 1, "HEAD": 2, "FOOD": 3}  
    element = {"SPRITE": 0, "DIRECTION": 1}
```

Snake game (3)

```
def __init__(self, w, h, length, delay):  
    self.W = w  
    self.H = h  
    self.initLen = length  
    self.snake = Snake(length)  
    self.delay = delay  
    self.board = [[[0]*2 for x in range(self.W)] for y in range(self.H)]  
  
    self.snake.head = [self.H//2, self.snake.length-1]  
    self.snake.tail = [self.H//2, 0]
```

Snake game (4)

```
for i in range(0, self.snake.length):
    self.board[self.H//2][i][SnakeGame.element["SPRITE"]] \
        = SnakeGame.sprite["BODY"]
    self.board[self.H//2][i][SnakeGame.element["DIRECTION"]] \
        = SnakeGame.direction["RIGHT"]

self.board[self.H//2][self.snake.length-1][SnakeGame.element["SPRITE"]] \
    = SnakeGame.sprite["HEAD"]
self.board[self.H//2][self.snake.length-1][SnakeGame.element["DIRECTION"]] \
    = SnakeGame.direction["RIGHT"]

x = random.randint(0, self.W-1)
y = random.randint(0, self.H-1)
while self.board[y][x][SnakeGame.element["SPRITE"]] \
    != SnakeGame.sprite["EMPTY"]:
    x = random.randint(0, self.W-1)
    y = random.randint(0, self.H-1)

self.board[y][x][SnakeGame.element["SPRITE"]] = SnakeGame.sprite["FOOD"]
```

Snake game (5)

```
def DrawScene(self):  
    os.system('cls||clear')  
    for x in range(0, self.W+2):  
        print("=", end="")  
    print("")
```

Snake game (5)

```
for y in range(0, self.H):
    print("|", end="")
    for x in range(0, self.W):
        if self.board[y][x][SnakeGame.element["SPRITE"]] \
            == SnakeGame.sprite["BODY"]:
            print("+", end="")
        elif self.board[y][x][SnakeGame.element["SPRITE"]] \
            == SnakeGame.sprite["HEAD"]:
            print("@", end="")
        elif self.board[y][x][SnakeGame.element["SPRITE"]] \
            == SnakeGame.sprite["FOOD"]:
            print("*", end="")
        else:
            print(" ", end="")
    print("|")

for x in range(0, self.W+2):
    print("=", end="")
print("")
```

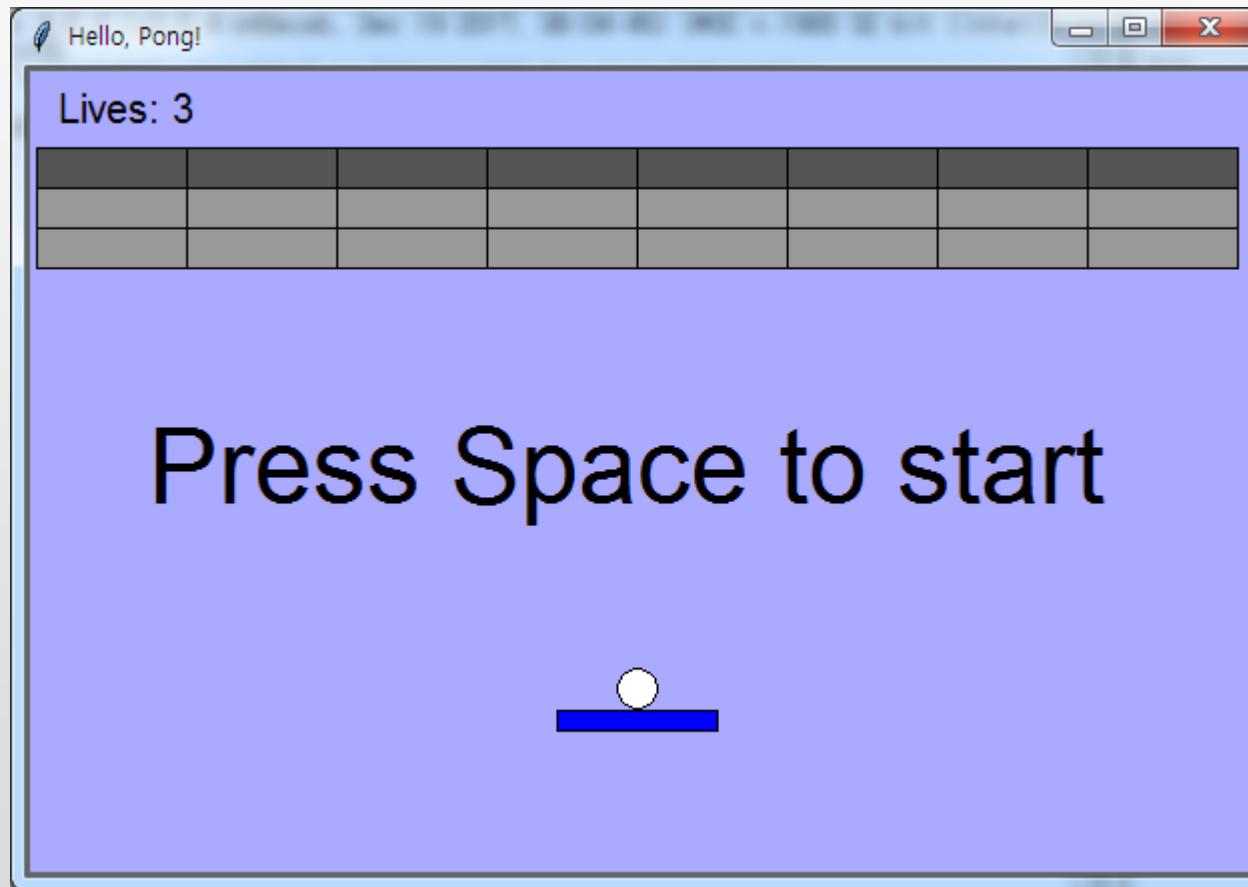
```
@staticmethod  
def GetDirection():  
    rtn = SnakeGame.direction["NON_DIR"]  
    msvcrt.getch()  
    ch = msvcrt.getch().decode()  
    if ch == chr(72):  
        print("UP")  
        rtn = SnakeGame.direction["UP"]  
    elif ch == chr(75):  
        print("LEFT")  
        rtn = SnakeGame.direction["LEFT"]  
    elif ch == chr(77):  
        print("RIGHT")  
        rtn = SnakeGame.direction["RIGHT"]  
    elif ch == chr(80):  
        print("DOWN")  
        rtn = SnakeGame.direction["DOWN"]  
  
    return rtn
```

Snake game (7) – 실습 1

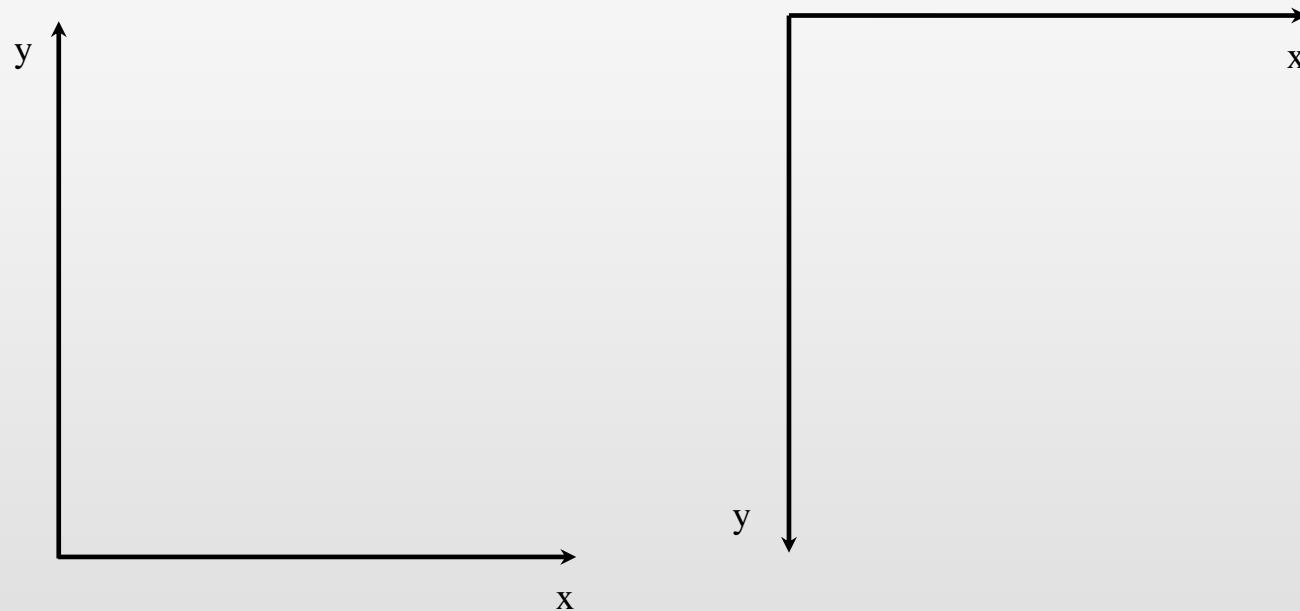
```
def GameLoop(self):  
    self.DrawScene()  
    current = SnakeGame.direction["RIGHT"]  
    ret = SnakeGame.direction["RIGHT"]  
    while True:  
        start = time.time()  
        while (time.time() - start) <= self.delay/1000:  
            if msvcrt.kbhit():  
                current = SnakeGame.GetDirection()  
        # codes 실습 1  
        self.DrawScene()  
        print("Score: {}".format(self.snake.length - self.initLen))  
  
if __name__ == '__main__' :  
    game = SnakeGame(60, 24, 4, 300)  
    game.GameLoop()
```

1. Hello, Pong!

Kyung Hee University
Daeho Lee



Coordinates



```
from tkinter import Tk  
root = Tk()  
root.title('Hello')  
root.mainloop()
```

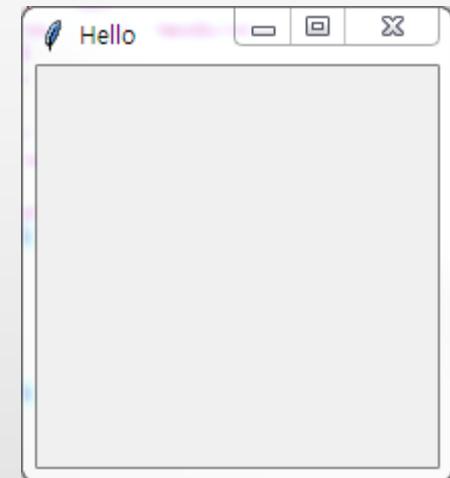
tkinter: Python interface to Tcl/Tk

Tk: widget toolkit

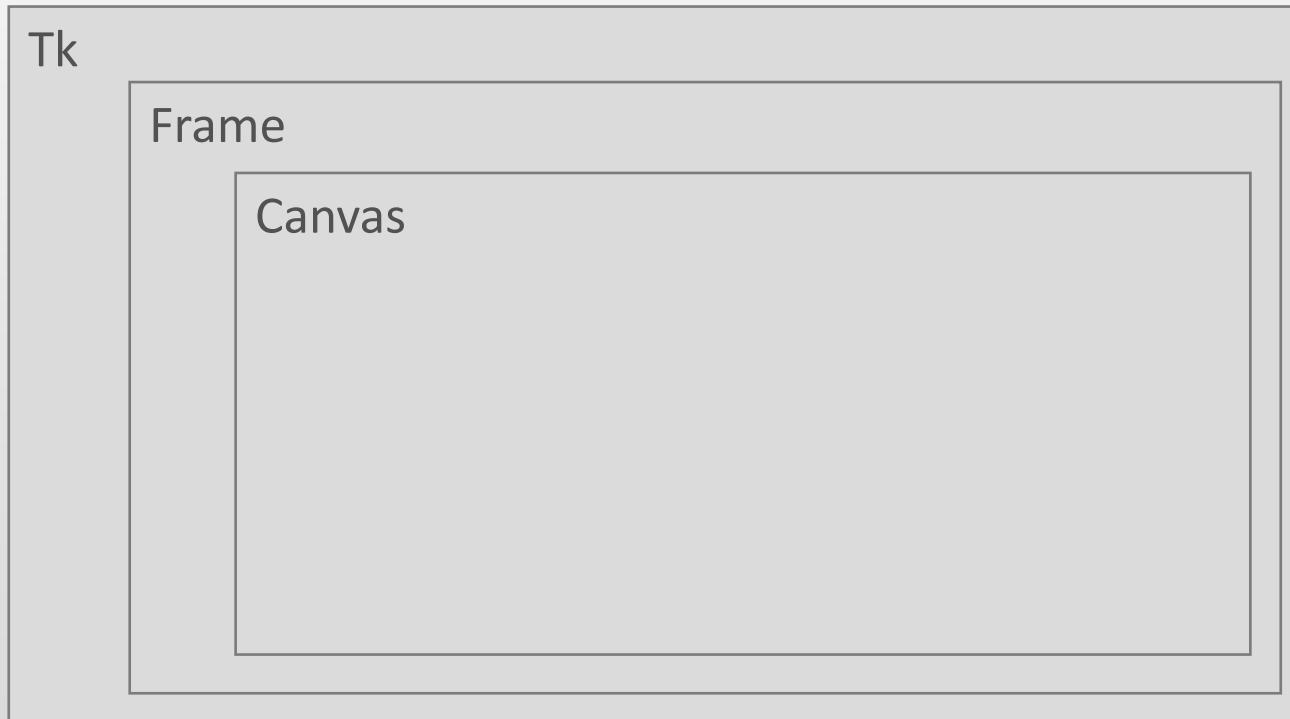
Tcl: script language

tkinter.Tk() : toplevel widget

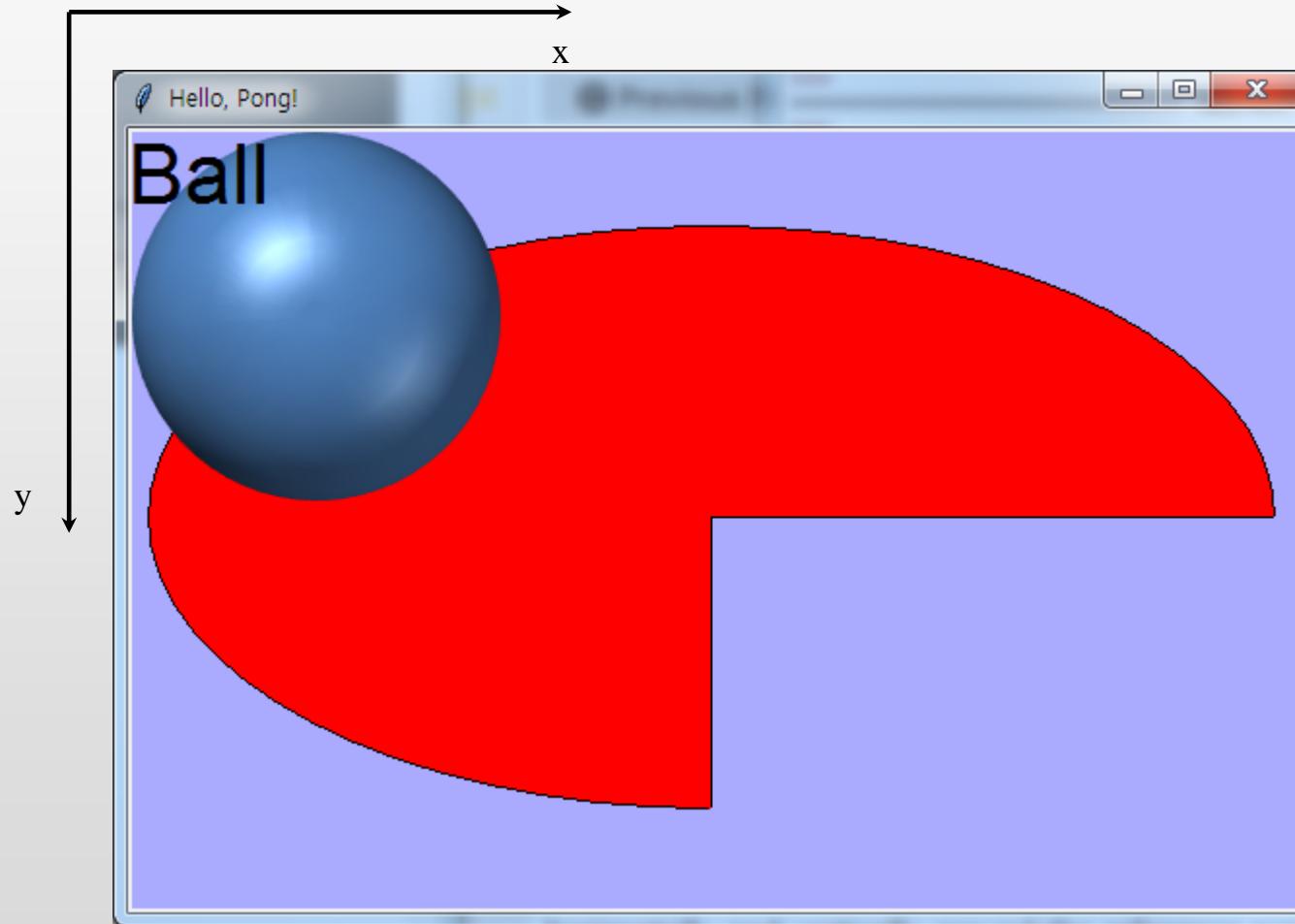
Tk().mainloop(): starting the event loop



- Frame: container widget to organize other widgets
- Canvas: widget for drawing shapes



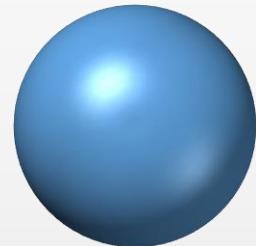
Simple GUI and drawing (1)



Simple GUI and drawing (2)

```
import tkinter as tk

nW = 600
nH = 400
root = tk.Tk()
frame = tk.Frame(root)
canvas = tk.Canvas(frame,
                   width=nW, height=nH, bg='#aaaaff')
# Canvas (master, option=value, ... )
# #ff0000 -> red
```



Simple GUI and drawing (3)

```
coord = 10, 50, nW-10, nH-50 #tuple
arc = canvas.create_arc(coord, start=0,
                        extent=270, fill='red')
filename = tk.PhotoImage(file = 'ball.png')
Image = canvas.create_image(97, 97,
                           anchor=tk.CENTER, image=filename)
# anchor: NW, N, NE, W, CENTER, E, SW, S, SE
text = canvas.create_text(0, 0, text = 'Ball',
                         anchor = tk.NW, font = ('Arial', '32'))
frame.pack()      # pack: displaying the widget
canvas.pack()     # on its parent container
root.title('Hello, Pong!')
root.mainloop()
```

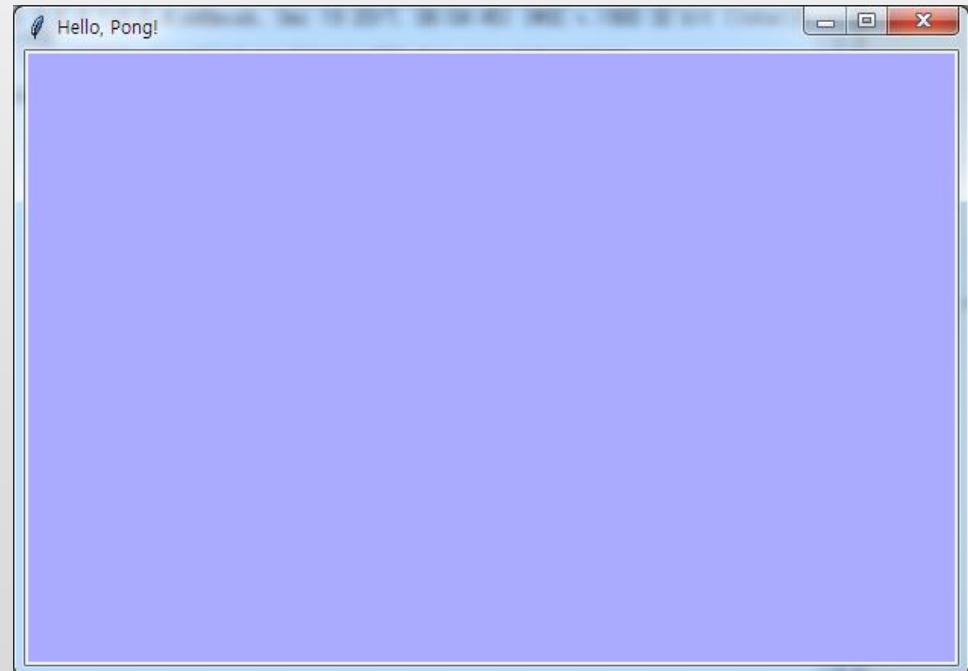
Simple GUI by class (1)

```
import tkinter as tk

class Game(tk.Frame):
    def __init__(self, master):
        # Frame(master, option, ...)
        super(Game, self).__init__(master)
        self.lives = 3
        self.width = 610
        self.height = 400
        self.canvas = tk.Canvas(self, bg='#aaaaff',
                               width=self.width,
                               height=self.height)
        self.canvas.pack()
        self.pack()
```

Simple GUI by class (2)

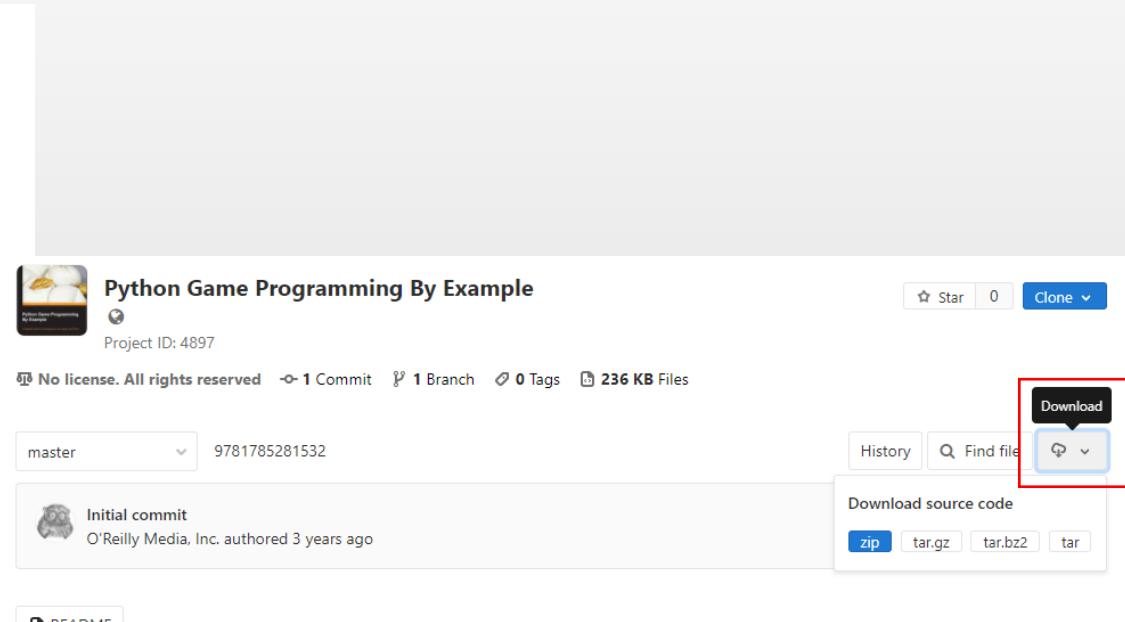
```
if __name__ == '__main__':
    root = tk.Tk()
    root.title('Hello, Pong!')
    game = Game(root)
    game.mainloop()
```



Canvas methods

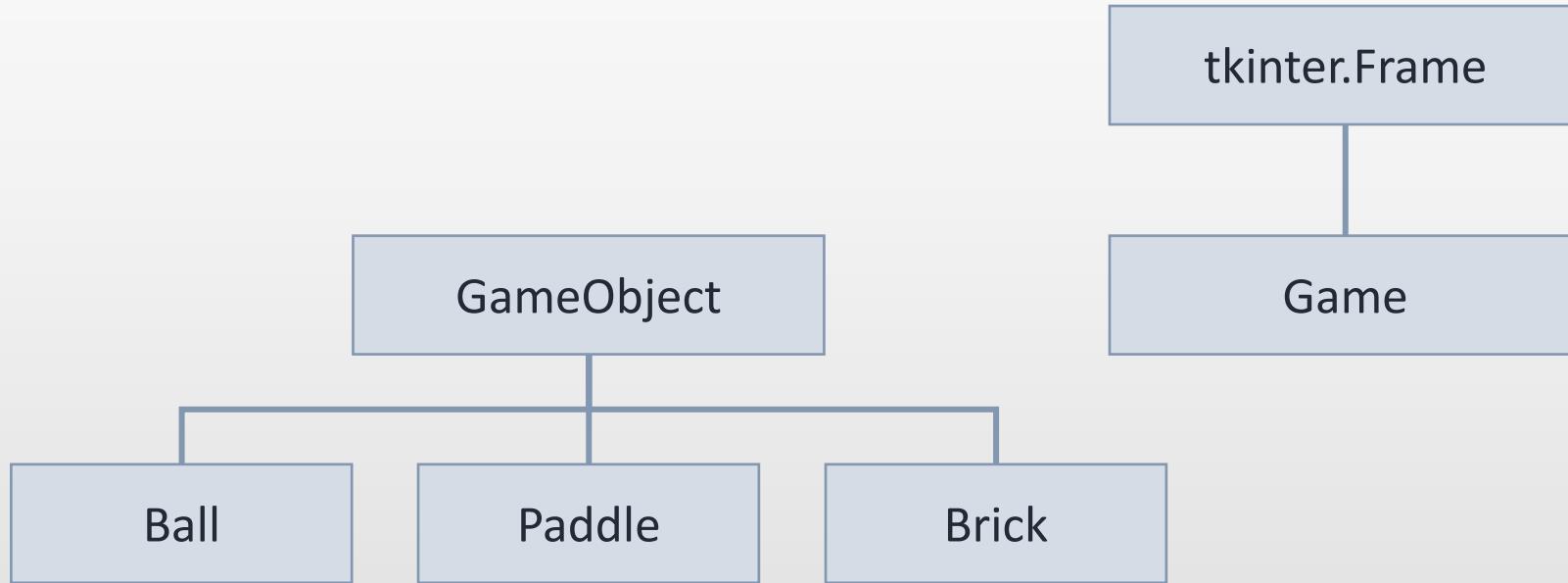
canvas.coords(item)	Returns the coordinate of the BB of an item
canvas.move(item, x, y)	Moves an item by offset
canvas.delete(item)	Deletes an item
canvas.winfo_width()	Returns the canvas width
canvas.itemconfig(item, **options)	Changes the options of an item (** keyword arguments)
canvas.bind(event, callback)	Binds an input event with the callback function
canvas.unbind(event)	Unbinds an input event
canvas.create_text(*position, **opts)	Draws text
canvas.find_withtag(tag)	Returns the items with a specific tag
canvas.find_overlapping(*position)	Returns the items that overlap or are included in the rectangle

- ◆ <https://www.packtpub.com/game-development/python-game-programming-example>



- ◆ <https://resources.oreilly.com/examples/9781785281532>

Class configuration



Class objects



◆ Game (Base: tkinter.Frame)

➤ __init__

- ✓ self.lives, self.width, self.height, self.canvas, self.items (empty)
 self.ball (None), self.paddle
- ✓ self.items (Brick, self.add_brick)
- ✓ self.hud (#lives text id)
- ✓ self.setup_game
 - self.add_ball # paddle's ball
 - self.update_lives_text
 - <space> binding:
 self.start_game: <space> unbinding, self.game_loop
- ✓ self.canvas.focus_set() # for capturing events
- ✓ <left> and <right> binding: self.paddle.move

◆ game_loop (Game method)

- self.check_collisions
 - ✓ Collision detection & response
- Count bricks
- Display, update, game_loop

```
class GameObject(object): # for Ball, Paddle and Brick
    def __init__(self, canvas, item):
        self.canvas = canvas
        self.item = item
    def get_position(self):
        return self.canvas.coords(self.item)
    def move(self, x, y):
        self.canvas.move(self.item, x, y)
    def delete(self):
        self.canvas.delete(self.item)
```

```
class Ball(GameObject):  
    def __init__(self, canvas, x, y):  
        self.radius = 10  
        self.direction = [1, -1]  
        self.speed = 1  
        item = canvas.create_oval(x-self.radius,  
                               y-self.radius, x+self.radius,  
                               y+self.radius, fill='white')  
        super(Ball, self).__init__(canvas, item)
```

```
def update(self):  
    coords = self.get_position()  
    width = self.canvas.winfo_width()  
    if coords[0] <= 0 or coords[2] >= width: #sides  
        self.direction[0] *= -1  
    if coords[1] <= 0: # top  
        self.direction[1] *= -1  
    x = self.direction[0] * self.speed  
    y = self.direction[1] * self.speed  
    self.move(x, y)
```

```
def collide(self, game_objects):  
    coords = self.get_position()  
    x = (coords[0] + coords[2]) * 0.5  
    if len(game_objects) > 1: # two or more bricks  
        self.direction[1] *= -1
```

```
elif len(game_objects) == 1: # paddle & bricks
    game_object = game_objects[0]
    coords = game_object.get_position()
    if x > coords[2]: # object right
        self.direction[0] = 1
    elif x < coords[0]: # object left
        self.direction[0] = -1
    else:
        self.direction[1] *= -1
for game_object in game_objects:
    if isinstance(game_object, Brick):
        game_object.hit()
```

```
class Paddle(GameObject):
    def __init__(self, canvas, x, y):
        self.width = 80
        self.height = 10
        self.ball = None
        item = canvas.create_rectangle(x - self.width/2,
                                       y - self.height/2, x + self.width/2,
                                       y + self.height/2, fill='blue')

    super(Paddle, self).__init__(canvas, item)
```

```
def set_ball(self, ball):  
    self.ball = ball  
  
def move(self, offset):  
    coords = self.get_position()  
    width = self.canvas.winfo_width()  
    if coords[0] + offset >= 0 and \  
        coords[2] + offset <= width:  
        super(Paddle, self).move(offset, 0)  
        if self.ball is not None:  
            self.ball.move(offset, 0)
```

```
class Brick(GameObject):
    COLORS = {1: '#999999', 2: '#555555', 3: '#222222'}

    #static member

    def __init__(self, canvas, x, y, hits):
        self.width = 75
        self.height = 20
        self.hits = hits
        color = Brick.COLORS[hits]
        item = canvas.create_rectangle \
            (x - self.width / 2, y - self.height / 2,
             x + self.width / 2, y + self.height / 2,
             fill=color, tags='brick')
```

```
super(Brick, self).__init__(canvas, item)

def hit(self):
    self.hits -= 1
    if self.hits == 0:
        self.delete()
    else:
        self.canvas.itemconfig(self.item,
                               fill=Brick.COLORS[self.hits])
```

```
class Game(tk.Frame) :  
    def __init__(self, master) :  
        super(Game, self). __init__(master)  
        self.lives = 3  
        self.width = 610  
        self.height = 400  
        self.canvas = tk.Canvas(self, bg='#aaaaff',  
                               width=self.width,  
                               height=self.height,)  
        self.canvas.pack()  
        self.pack()
```

```
self.items = {} # items dict
self.ball = None
self.paddle = Paddle(self.canvas,
                      self.width/2, 326)
self.items[self.paddle.item] = self.paddle
# self.paddle.item: 1
for x in range(5, self.width - 5, 75):
    self.add_brick(x + 37.5, 50, 2)
    self.add_brick(x + 37.5, 70, 1)
    self.add_brick(x + 37.5, 90, 1)
# self.add_brick(x, y, hits)
# Brick: w=75, h=20
```

```
self.hud = None # lives text  
self.setup_game()# add_ball, paddle's ball,  
                 # binding  
self.canvas.focus_set() # capturing events  
self.canvas.bind('<Left>',  
                 lambda _: self.paddle.move(-10))  
self.canvas.bind('<Right>',  
                 lambda _: self.paddle.move(10))
```

```
# lambda function (anonymous function)

F1 = lambda x: print(x); F1(10)
F2 = lambda x, y: print(x+y); F2(10, 20)
F3 = lambda : print(40); F3()
F4 = lambda _: F1(50); F4(1)
```

```
# binding
def callback(event):
    print ('clicked at', event.x, event.y)
def key(event):
    print ('pressed', event.char)

bind('<Button-1>', callback)
bind('<Key>', Key)
```

```
def setup_game(self):  
    self.add_ball()  
    self.update_lives_text()  
    self.text = self.draw_text(300, 200,  
                             'Press Space to start')  
    self.canvas.bind('<space>',  
                    lambda _: self.start_game())
```

```
def add_ball(self):  
    if self.ball is not None:  
        self.ball.delete()  
    paddle_coords = self.paddle.get_position()  
    x = (paddle_coords[0] + paddle_coords[2]) * 0.5  
    self.ball = Ball(self.canvas, x, 310)  
    self.paddle.set_ball(self.ball) # paddle's ball  
  
def add_brick(self, x, y, hits):  
    brick = Brick(self.canvas, x, y, hits)  
    self.items[brick.item] = brick # frame items
```

```
def draw_text(self, x, y, text, size='40'):  
    font = ('Helvetica', size)  
    return self.canvas.create_text(x, y,  
                                   text=text, font=font)  
# draw text, return id  
  
def update_lives_text(self):  
    text = 'Lives: %s' % self.lives # %d  
    if self.hud is None:  
        self.hud = self.draw_text(50, 20, text, 15)  
    else: # update  
        self.canvas.itemconfig(self.hud, text=text)
```

```
def start_game(self):
    self.canvas.unbind('<space>')
    self.canvas.delete(self.text) # 'Press Space...'
    self.paddle.ball = None # delete paddle's ball
    self.game_loop()

def game_loop(self):
    self.check_collisions()
    num_bricks = \ # brick's tag is 'brick'
                  len(self.canvas.find_withtag('brick'))
    if num_bricks == 0:
        self.ball.speed = None
        self.draw_text(300, 200, 'You win!')
```

```
elif self.ball.get_position() [3] >= self.height:  
    # bottom of ball  
    self.ball.speed = None  
    self.lives -= 1  
    if self.lives < 0:  
        self.draw_text(300, 200, 'Game Over')  
    else:  
        self.after(1000, self.setup_game)  
else:  
    self.ball.update()  
    self.after(5, self.game_loop) # delay
```

```
def check_collisions(self):  
    ball_coords = self.ball.get_position()  
    items = \  
        self.canvas.find_overlapping \  
        (*ball_coords)  
    objects = \  
    [self.items[x] for x in items if x in self.items]  
    self.ball.collide(objects)
```

Unpacking list into two arguments

```
def Add (a, b):  
    print (a + b)  
  
x = [2, 3]  
Add(*x)
```

```
# Create list

new_list = [expr(elem) for elem in collection]

new_list = [expr(elem) for elem in collection if elem is not
None]

new_list = []
for elem in collection :
    if elem is not None :
        new_list.append(elem)
```

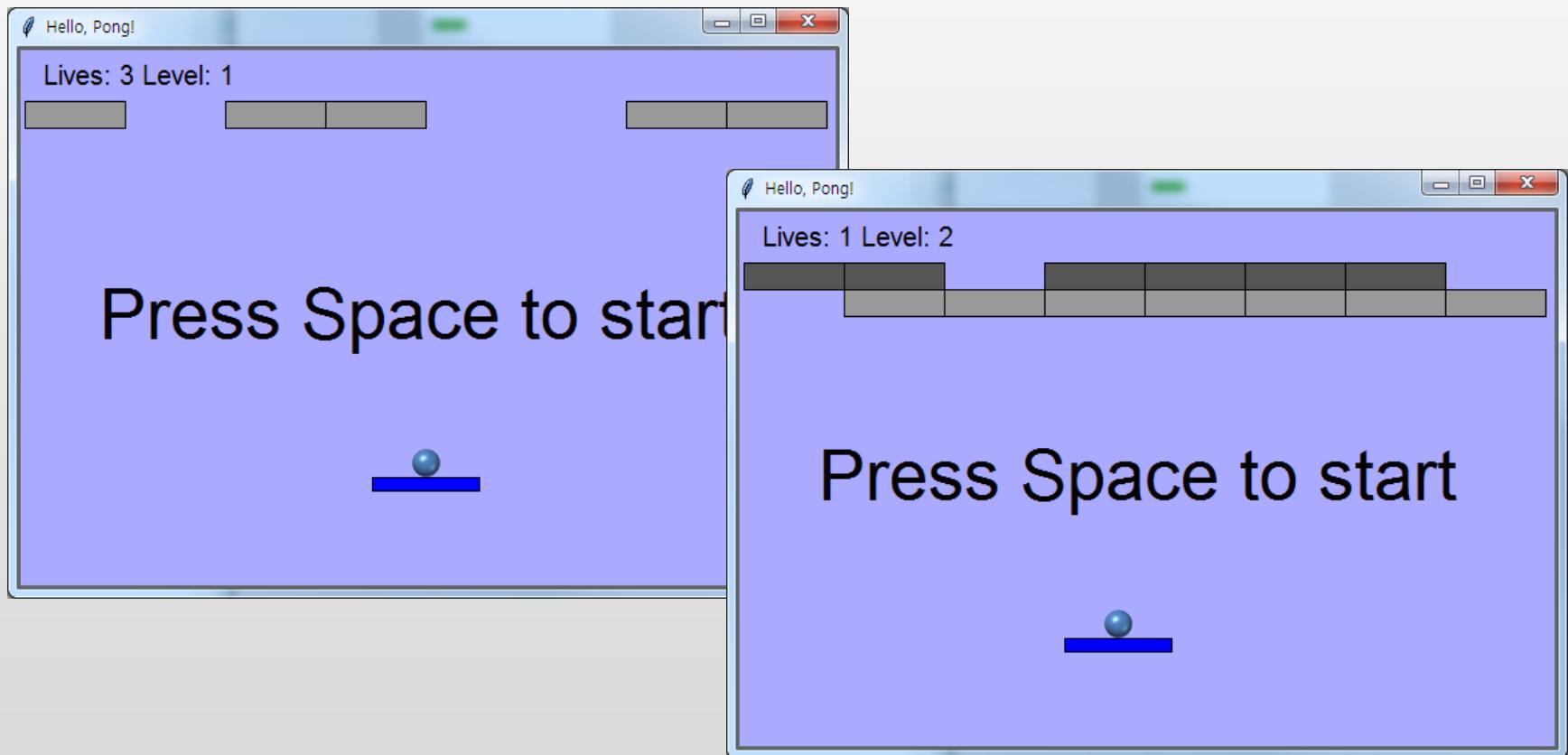
```
if __name__ == '__main__':
    root = tk.Tk()
    root.title('Hello, Pong!')
    game = Game(root)
    game.mainloop()
```

◆ 공을 ball2.png로 변경

- tkinter.PhotoImage, Canvas.create_image 사용
- 그림의 Canvas.coords의 결과는 중심 좌표
 - ✓ 이전 소스에서 사용된 oval의 coords는 left, top, right, bottom
- 방법1)
 - ✓ 좌표 사용되는 위치에서 모두 수정
 - ✓ Ball.__init__, Ball.update, Ball.collide, Game.check_collisions 수정
- 방법2)
 - ✓ Ball.get_position을 overriding

실습 2 – Pong 게임 수정 (2)

- ◆ Level 추가, 벽돌 층수 변경, 임의의 벽돌만 생성



◆ Level 추가, 벽돌 층수 변경, 임의의 벽돌만 생성

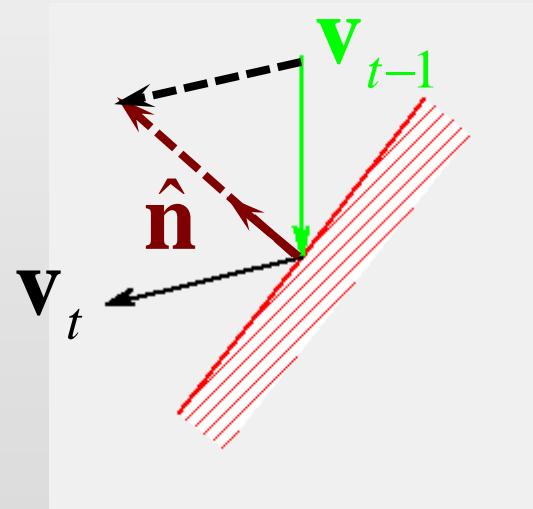
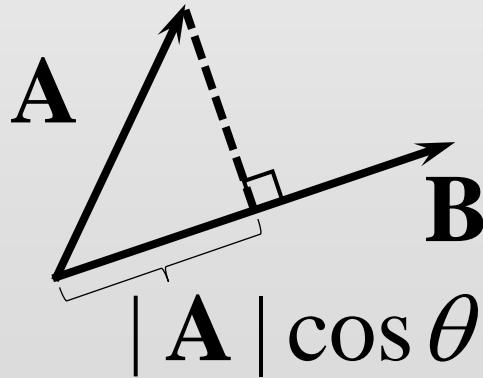
- Game.__init__에서 self.level = 1로 초기화
- Game.game_loop에서 num_bricks == 0인 경우에 level 증가, <space> binding 등의 추가 필요
- Game.update_lives_text에 level 표시: '%d %d' % (x, y)
- Level 시작할 때, 벽돌 생성을 함수로 변경
ex) Game.setup_level(self)
 - ✓ Game.level에 따른 벽돌 층수 변경 및 임의의 벽돌만 생성
 - ✓ random.randint(0, 9) 사용

```
>>> import random
>>> help(random.randint)
Help on method randint in module random:
```

```
randint(a, b) method of random.Random instance
    Return random integer in range [a, b], including both end
```

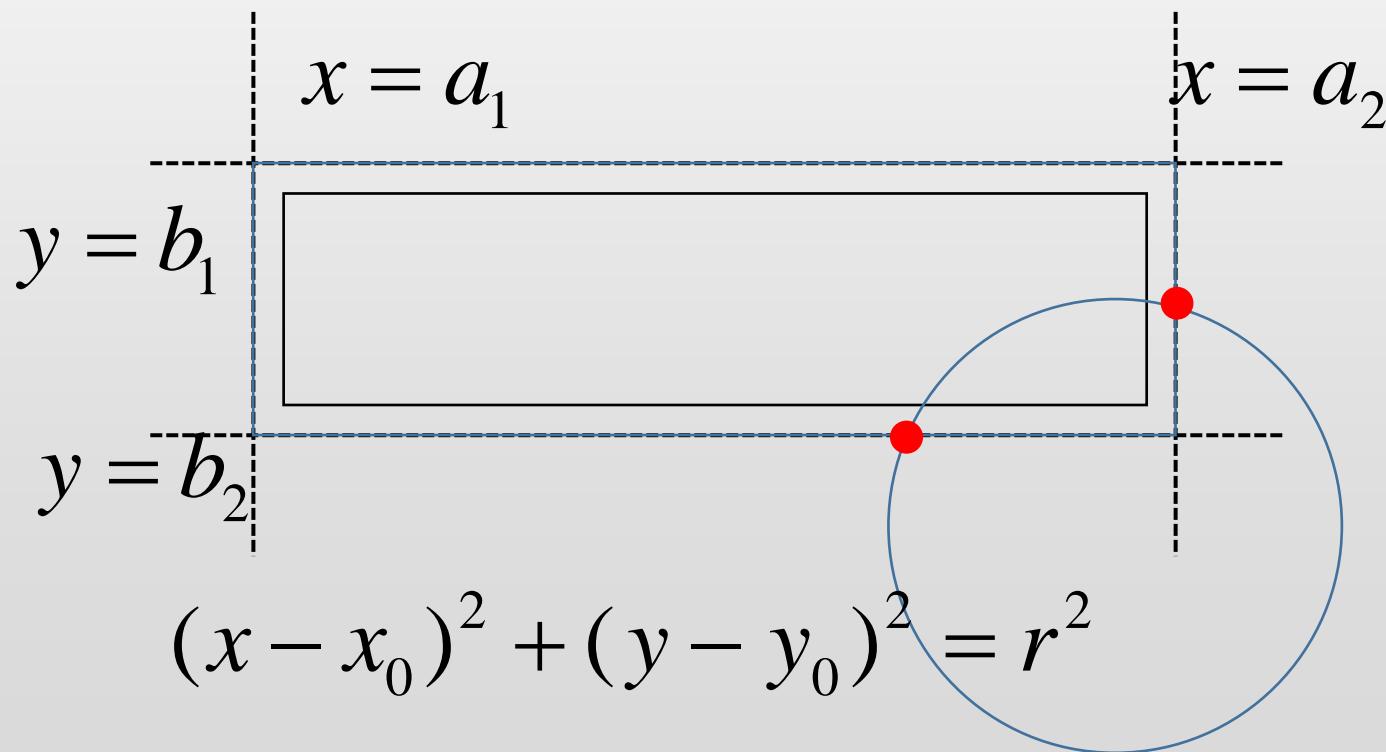
◆ 모서리 충돌 반응 수정

$$\mathbf{v}_t = 2(-\mathbf{v}_{t-1} \cdot \hat{\mathbf{n}})\hat{\mathbf{n}} + \mathbf{v}_{t-1}$$



◆ Ball

- 생성자에서 방향 벡터의 크기를 1로 수정
- 네 개의 직선과 원의 교점을 리스트에 추가



➤ 교점 판별

- ✓ 교점의 x 또는 y를 계산하기 위하여 이차 방정식 생성
- ✓ 이차 방정식의 판별식이 0보다 작으면 교점이 없음
- ✓ 교점의 x 또는 y의 좌표가 brick이나 paddle와 겹침 확인
(예, $y=b_2$ 와의 교점의 x좌표가 a_1 과 a_2 사이에 있는지 확인)
- ✓ 위의 조건을 만족하면 교점을 리스트에 추가

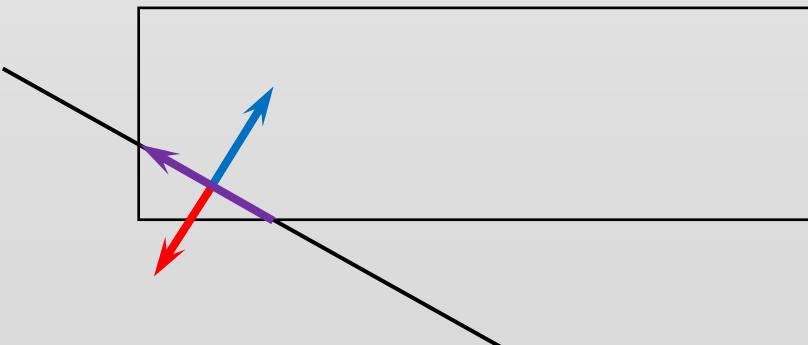
➤ 반응 계산

- ✓ 교점의 2개이면 아래의 수식을 이용하여 방향 벡터 수정

$$\mathbf{v}_t = 2(-\mathbf{v}_{t-1} \cdot \hat{\mathbf{n}})\hat{\mathbf{n}} + \mathbf{v}_{t-1}$$

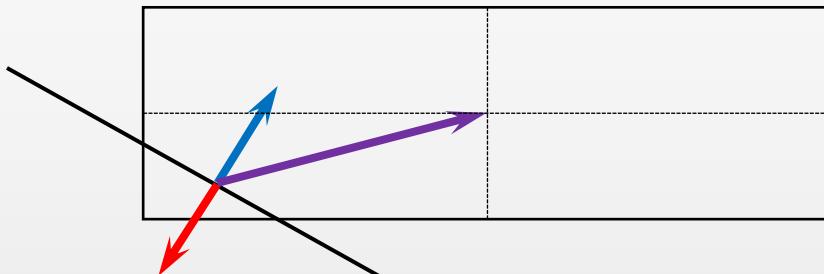
- ✓ 교점을 잇는 벡터를 +90/-90도로 회전하여 두 개의 법선 벡터 생성

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

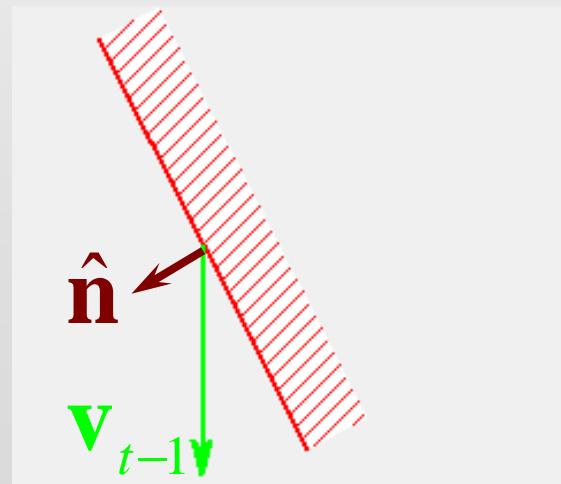


실습 2 – Pong 게임 수정 (8)

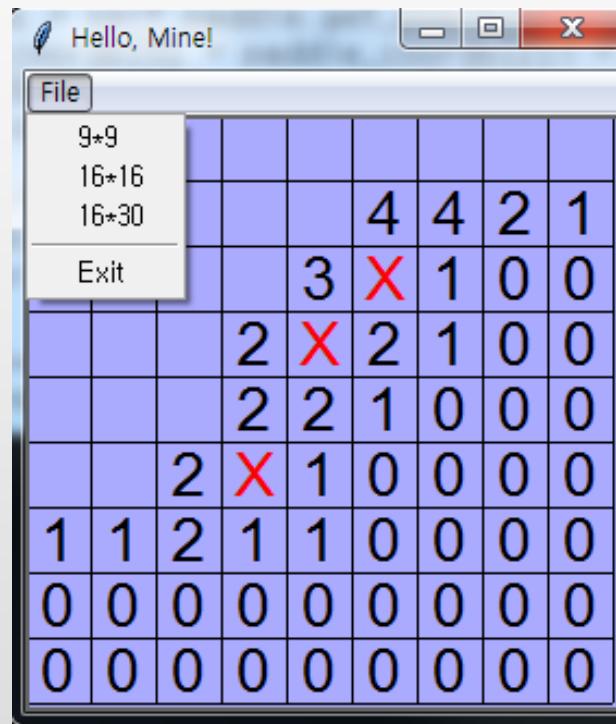
- ✓ 교점의 중심에서 물체의 중심을 잇는 벡터와의 각도가 크면 바깥쪽으로 향하는 법선 벡터



- ✓ 법선 벡터가 충돌하기 전의 진행 방향과 사잇각이 90도 이하이면 충돌 반응하지 않음



◆ 지뢰 찾기 게임



◆ 레벨

- column*row, mines; 9*9, 10; 16*16, 40; 30*16, 99

◆ 메뉴

```
menubar = tk.Menu(master)
filemenu = tk.Menu(menubar, tearoff=0)
filemenu.add_command(label="9*9", command = self.begin)
...
filemenu.add_separator()
filemenu.add_command(label="Exit",
                     command=master.destroy)
menubar.add_cascade(label="File", menu=filemenu)
master.config(menu=menubar)
```

◆ 메시지 박스

- `from tkinter import messagebox`
- `tk.messagebox.showinfo('Win', 'You win!')`

◆ 3차원 행렬 사용

- Import numpy as np **# pip install numpy**
- `pattern = np.arange(5*5*5).reshape(5, 5, 5)`
- `pattern[0][0][0] = 10`
- `pattern = [[[0]*5 for i in range(5)] for k in \range(5)]`

◆ 마우스 버튼 이벤트

```
self.canvas.bind('<Button-1>',  
self.left_button)
```

```
self.canvas.bind('<Button-3>',  
self.right_button)
```

```
def left_button(self, event):  
    x = event.x//self.square  
    y = event.y//self.square  
def right_button(self, event):  
    x = event.x//self.square  
    y = event.y//self.square
```

◆ Recursion

```
def F(n):  
    print(n%10)  
    if n//10 > 0:  
        F(n//10)
```

```
F(123)
```

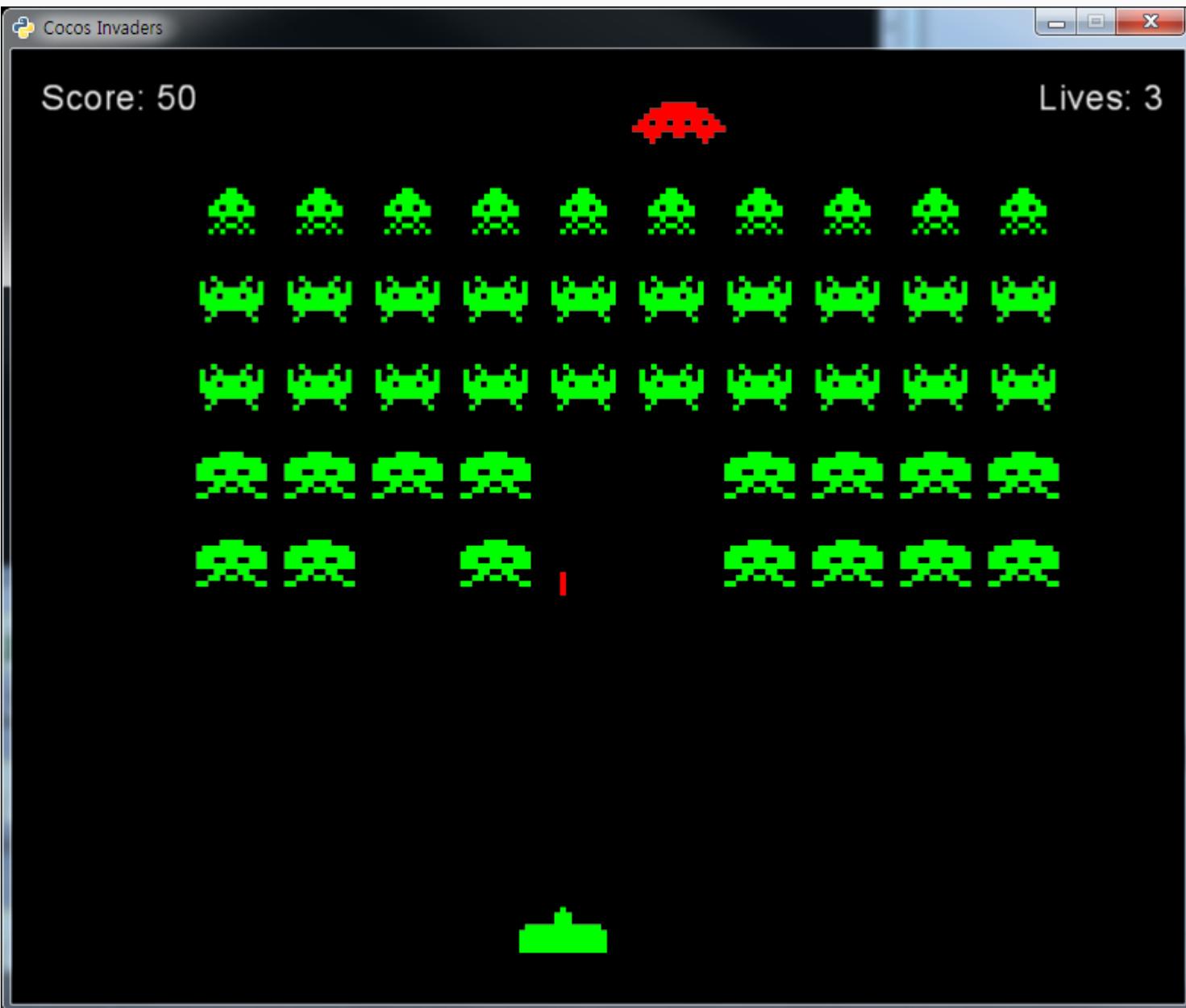
◆ Recursion

```
def detect_region(self, x, y):
    for yy in range(-1, 2):
        for xx in range(-1, 2):
            if x+xx < 0: continue
            if x+xx >= self.column:
                continue
            if y+yy < 0: continue
            if y+yy >= self.row: continue
            ...
            if
self.pattern[y+yy][x+xx][2] != 0:
                continue
            self.detect_region(x+xx, y+yy)
```

- ◆ > pip install pyinstaller
- ◆ pip: package management system for Python packages
- ◆ > pyinstaller ex.py
- ◆ > pyinstaller -F -n newname.exe ex.py

2. Cocos Invaders

Kyung Hee University
Daeho Lee



◆ Cocos2d

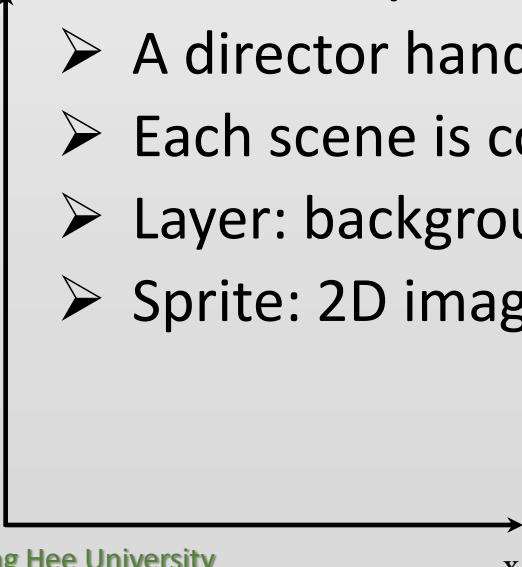
- Open source S/W framework to build games, apps and interactive programs
- Install
 - > pip install cocos2d
- Platforms
 - ✓ Cocos2d: Windows, Linux (Python)
 - ✓ Cocos2d-x: iOS, Android, Linux (C++)
 - ✓ Cocos2d-xna: Windows 7, 8, Xbox 360 (C#)
 - ✓ ...
- <http://cocos2d.org/>

```
import cocos  
  
cocos.director.director.init(caption='Hello')  
  
layer = cocos.layer.Layer()  
  
scene = cocos.scene.Scene(layer)  
  
cocos.director.director.run(scene)
```



Basic concepts

y

- 
- A director handles the workflow between scenes
 - Each scene is composed of an arbitrary number of layers
 - Layer: background, text, object, ...
 - Sprite: 2D image or animation

Basic application (2)

```
import cocos

class MainLayer(cocos.layer.Layer):
    def __init__(self):
        super(MainLayer, self).__init__()
        self.player = cocos.sprite.Sprite('ball.png')
        self.player.position = (320, 240)
        self.player.color = (0, 0, 255) # AND coloring
        self.add(self.player)

if __name__ == '__main__':
    cocos.director.director.init(caption='Hello, Cocos')
    layer = MainLayer()
    scene = cocos.scene.Scene(layer)
    cocos.director.director.run(scene)
```

◆ Scene, Layer, Sprite: inherit from CocosNode

add(child, z = 0, name = None)	Adds a child
remove(child)	Removes a child
kill()	Removes itself
get_children()	Returns a list of the children of the node, ordered by their z index
parent	Parent node
position	Position coordinates relative to the parent node
x, y	x, y coordinates
schedule(callback)	Callback is called every frame
on_enter()	Called when added while the parent is in the active or if the scene goes active
on_exit()	Called when removed while the parent is in the active or if the scene goes active

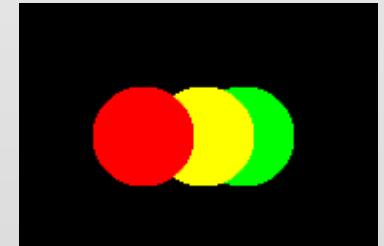
Basic application (4)

```
import cocos

class MainLayer(cocos.layer.Layer):
    def __init__(self, color, x, y):
        super(MainLayer, self).__init__()
        self.player = cocos.sprite.Sprite('ball.png')
        self.player.position = (x, y)
        self.player.color = color
        self.add(self.player)
```

Basic application (5)

```
if __name__ == '__main__':
    cocos.director.director.init(caption='Hello, \
                                    Cocos')
    layer1 = MainLayer((255, 0, 0), 320, 240)
    layer2 = MainLayer((255, 255, 0), 350, 240)
    layer3 = MainLayer((0, 255, 0), 370, 240)
    scene = cocos.scene.Scene(layer1)
scene.add(layer2, 5)
layer1.kill()
scene.add(layer1, 6)
scene.add(layer3, 2)
cocos.director.director.run(scene)
```



◆ Cocos2d

- Open source S/W framework to build games, apps and interactive programs

◆ Pyglet

- Cross-platform windowing and multimedia library for Python

◆ Pygame

- Cross-platform set of Python modules designed for writing video games

◆ cocos.director.director.init

- Creates the main window
- Pyglet window
- `fullscreen` : bool, Window is created in fullscreen. Default is False
- `resizable` : bool, Window is resizable. Default is False
- `vsync` : bool, Sync with the vertical retrace. Default is True
- `width` : int, Window width size. Default is 640
- `height` : int, Window height size. Default is 480
- `caption` : string, Window title.
- `visible` : bool, Window is visible or not. Default is True.

cocos.sprite.Sprite.__init__

- `image` : string or image name of the image resource or a pyglet image.
- `position` : tuple position of the anchor. Defaults to (0,0)
- `rotation` : float the rotation (degrees). Defaults to 0.
- `scale` : float the zoom factor. Defaults to 1.
- `opacity` : int the opacity (0=transparent, 255=opaque). Defaults to 255.
- `color` : tuple the color to colorize the child (RGB 3-tuple). Defaults to (255,255,255).
- `anchor` : (float, float) (x,y)-point from where the image will be positions, rotated and scaled in pixels. For example (image.width/2, image.height/2) is the center (default).

◆ Layer

- `is_event_handler = True`
- By Pyglet

- `on_key_press(key, modifiers)`
- `on_key_release(key, modifiers)`

- `on_mouse_motion(x, y, dx, dy)`
- `on_mouse_press(x, y, buttons, modifiers)`
- `on_mouse_drag(x, y, dx, dy, buttons, modifiers)`
- `modifiers`: bitwise or of several constants indicating which modifiers are active at the time of the press (ctrl, shift, capslock, etc.)

```
import cocos
from pyglet.window import key

class MainLayer(cocos.layer.Layer):
    is_event_handler = True

    def __init__(self):
        super(MainLayer, self).__init__()
        self.player = cocos.sprite.Sprite('ball.png')
        self.player.position = (320, 240)
        self.player.color = (255, 0, 0)
        self.add(self.player)

    def on_key_press(self, k, m):
        print('Pressed', key.symbol_string(k))
```

```
if __name__ == '__main__':
    cocos.director.director.init(caption='Hello, Cocos')
    layer = MainLayer()
    scene = cocos.scene.Scene(layer)
    cocos.director.director.run(scene)
```

- ◆ CocosNode.schedule(callback)
 - The callback function is called on every frame
 - ```
def callback(dt, *args, **kwargs):
```

    - ✓ dt: elapsed time, in seconds

```
import cocos
from collections import defaultdict # for key dict
from pyglet.window import key

class MainLayer(cocos.layer.Layer):
 is_event_handler = True
 def __init__(self):
 super(MainLayer, self).__init__()
 self.player = cocos.sprite.Sprite('ball.png')
 self.player.position = (320, 240)
 self.player.color = (255, 0, 0)
 self.add(self.player)
 self.speed = 100.0
 self.pressed = defaultdict(int)
 self.schedule(self.update)
```

```
def on_key_press(self, k, m):
 self.pressed[k] = 1

def on_key_release(self, k, m):
 self.pressed[k] = 0

def update(self, dt):
 x = self.pressed[key.RIGHT]-self.pressed[key.LEFT]
 y = self.pressed[key.UP]-self.pressed[key.DOWN]

 if x != 0 or y != 0:
 pos = self.player.position
 newX = pos[0] + self.speed*x*dt
 newY = pos[1] + self.speed*y*dt
 self.player.position = (newX, newY)
```

```
if __name__ == '__main__':
 cocos.director.director.init(caption='Hello, Cocos')
 layer = MainLayer()
 scene = cocos.scene.Scene(layer)
 cocos.director.director.run(scene)
```

◆ cocos.collision\_model package called  
**CollisionManager**

- Proximity & collisions
- CollisionManagerBruteForce
  - ✓ Straightforward implementation
  - ✓ For debugging purposes
- CollisionManagerGrid
  - ✓ Space is divided into rectangular cells
  - ✓ \_\_init\_\_: sets minimum and maximum coordinates, cell width and height
  - ✓ \_\_init\_\_(self, xmin, xmax, ymin, ymax, cell\_width, cell\_height)
  - ✓ Cell size: maximum object \* 1.25

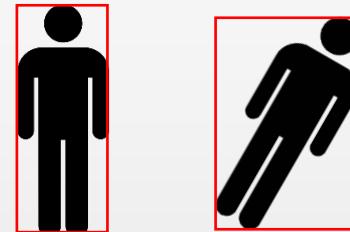
- `cocos.collision_model.Cshape`
  - ✓ Abstract geometric shape
- ✓ `cocos.collision_model.CircleShape`
  - Circle
  - Center and radius
  - Euclidean distance between two centers < sum of two radii
- ✓ `cocos.collision_model.AARectShape`
  - Rectangle
  - Axis-aligned
  - Manhattan distance (City block, taxicab, ...)

$$\sum_{i=1}^n |p_i - q_i|$$

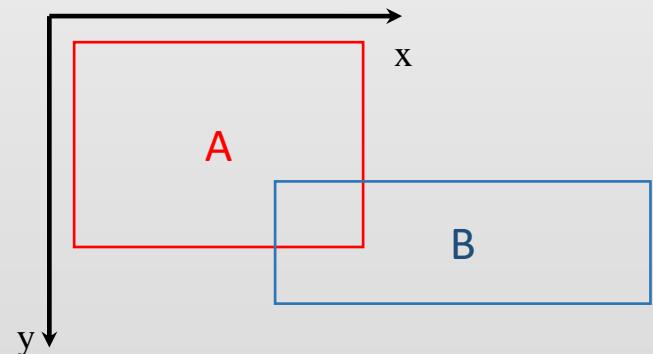
## ◆ Collisions

### ➤ AABB: Axis-Aligned Bounding Box

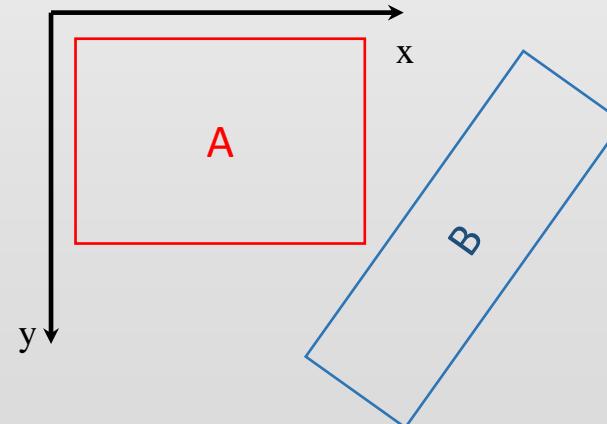
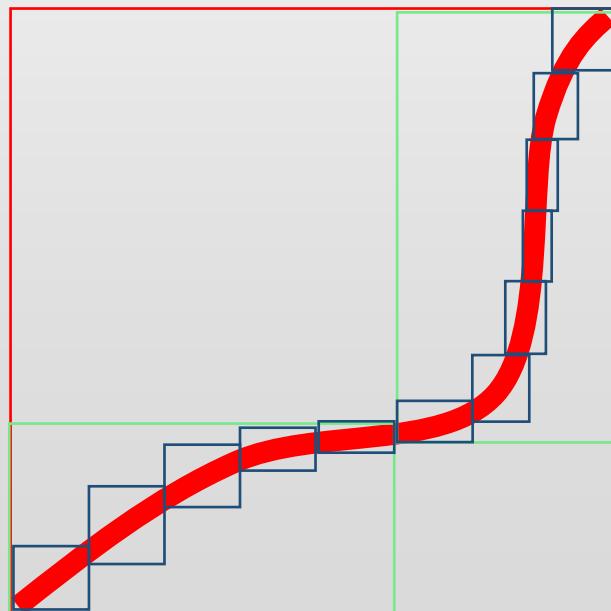
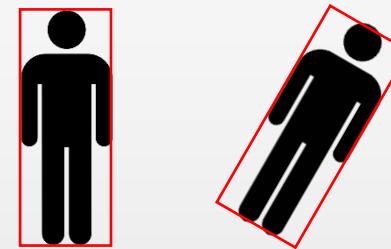
- ✓  $A_{\text{left}} \leq B_{\text{right}}$  and
- ✓  $A_{\text{right}} \geq B_{\text{left}}$  and
- ✓  $A_{\text{top}} \leq B_{\text{bottom}}$  and
- ✓  $A_{\text{bottom}} \geq B_{\text{top}}$



- ✓  $A_{\text{left}} > B_{\text{right}}$  or
- ✓  $A_{\text{right}} < B_{\text{left}}$  or
- ✓  $A_{\text{top}} > B_{\text{bottom}}$  or
- ✓  $A_{\text{bottom}} < B_{\text{top}}$



- OBB: Oriented Bounding Box
  - ✓ Using plan equations (3D) or line equations (2D)
  - ✓ ...
- Sphere trees, AABB trees, OBB Trees



```
import cocos # chapter2_01.py
import cocos.collision_model as cm
import cocos.euclid as eu

from collections import defaultdict
from pyglet.window import key

class Actor(cocos.sprite.Sprite):
 def __init__(self, x, y, color):
 super(Actor, self).__init__('ball.png',
 color = color)
 self.position = pos = eu.Vector2(x, y)
 self.cshape = cm.CircleShape(pos, self.width/2)
```

```
class MainLayer(cocos.layer.Layer):
 is_event_handler = True
 def __init__(self):
 super(MainLayer, self).__init__()
 self.player = Actor(320, 240, (0, 0, 255))
 self.add(self.player)

 for pos in [(100,100), (540,380), \
 (540,100), (100,380)]:
 self.add(Actor(pos[0], pos[1], (255, 0, 0)))

 cell = self.player.width * 1.25
 self.collman = cm.CollisionManagerGrid(0, 640,
 0, 480, cell, cell)
```

```
self.speed = 100.0
self.pressed = defaultdict(int)
self.schedule(self.update)

def on_key_press(self, k, m):
 self.pressed[k] = 1

def on_key_release(self, k, m):
 self.pressed[k] = 0
```

## Collisions (8)

```
list of tuples, unpacking
for x, y in [(1,'A'), (2,'B')]:
 print(x, y)
```

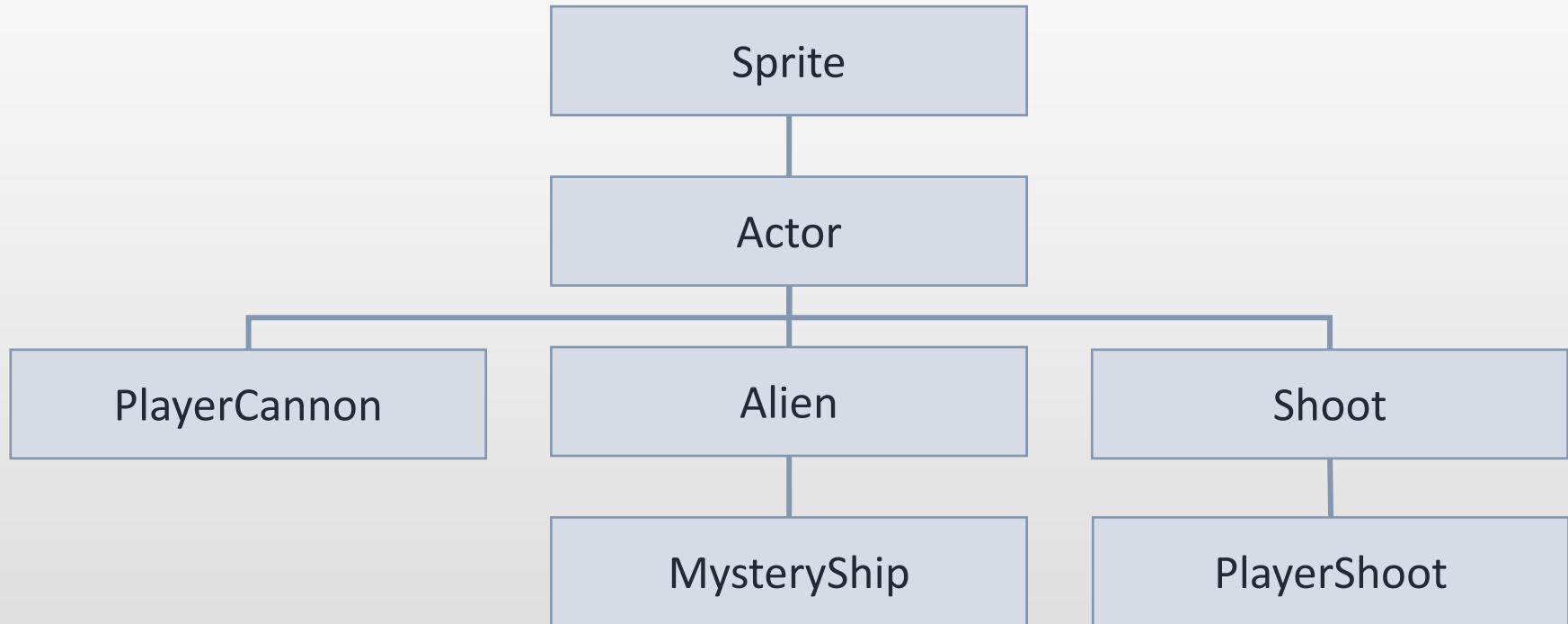
```
def update(self, dt):
 # clear collision manager (clear the known set)
 self.collman.clear()
 # add actors to the set
 for _, node in self.children:
 self.collman.add(node)
 # iterate over objects' collisions
 for other in self.collman.iter_colliding(self.player):
 self.remove(other) # other iterator

 x = self.pressed[key.RIGHT]-self.pressed[key.LEFT]
 y = self.pressed[key.UP]-self.pressed[key.DOWN]
```

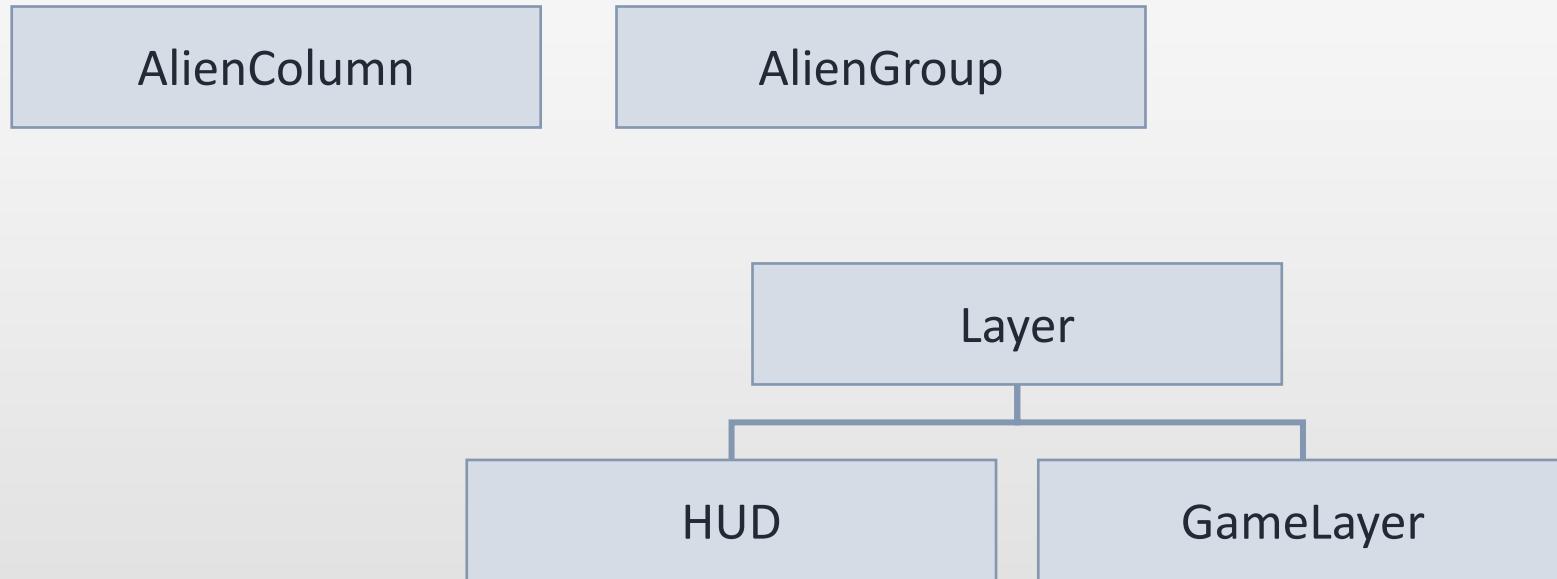
```
if x != 0 or y != 0:
 pos = self.player.position
 newX = pos[0] + self.speed*x*dt
 newY = pos[1] + self.speed*y*dt
 self.player.position = (newX, newY)
self.player.cshape.center = self.player.position

if __name__ == '__main__':
 cocos.director.director.init(caption='Hello, Cocos')
 layer = MainLayer()
 scene = cocos.scene.Scene(layer)
 cocos.director.director.run(scene)
```

# Space Invaders (1)



# Space Invaders (2)



```
import random

from collections import defaultdict

from pyglet.image import load, ImageGrid, Animation
from pyglet.window import key

import cocos.layer
import cocos.sprite
import cocos.collision_model as cm
import cocos.euclid as eu
```

```
class Actor(cocos.sprite.Sprite):
 def __init__(self, image, x, y):
 super(Actor, self).__init__(image)
 self.position = eu.Vector2(x, y)
 self.cshape = cm.AARectShape(self.position,
 self.width * 0.5, self.height * 0.5)
AARectShape.__init__(self, center, half_width,
half_height)
 def move(self, offset):
 self.position += offset
 self.cshape.center += offset
 def update(self, elapsed):
 pass # null operation
 def collide(self, other):
 pass
```

# PlayerCannon (1)

```
class PlayerCannon(Actor):
 KEYS_PRESSED = defaultdict(int) # updated in GameLayer

 def __init__(self, x, y):
 super(PlayerCannon, self).__init__\
 ('img/cannon.png', x, y)
 self.speed = eu.Vector2(200, 0) # move left or right

 def update(self, elapsed): # called in GameLayer.update
 #- GameLayer.schedule callback
 pressed = PlayerCannon.KEYS_PRESSED
 space_pressed = pressed[key.SPACE] == 1
 if PlayerShoot.INSTANCE is None and space_pressed:
 self.parent.add(PlayerShoot(self.x, self.y + 50))
```

```
movement = pressed[key.RIGHT] - pressed[key.LEFT]
w = self.width * 0.5
if movement != 0 and \
 w <= self.x <= self.parent.width - w:
 self.move(self.speed * movement * elapsed)
좌/우 벽면으로 이동시 오류 발생
def collide(self, other):
 other.kill()
 self.kill()
```

```
class GameLayer(cocos.layer.Layer):
 is_event_handler = True

 def on_key_press(self, k, _):
 PlayerCannon.KEYS_PRESSED[k] = 1

 def on_key_release(self, k, _):
 PlayerCannon.KEYS_PRESSED[k] = 0
```

```
def __init__(self, hud):
 super(GameLayer, self).__init__()
 w, h = cocos.director.director.get_window_size()
 self.hud = hud
 self.width = w
 self.height = h
 self.lives = 3
 self.score = 0
 self.update_score()
 self.create_player()
 self.create_alien_group(100, 300)
 cell = 1.25 * 50
 self.collman = cm.CollisionManagerGrid(0, w, 0, h,
 cell, cell)
 self.schedule(self.update)
```

```
def create_player(self):
 self.player = PlayerCannon(self.width * 0.5, 50)
 self.add(self.player)
 self.hud.update_lives(self.lives)

def update_score(self, score=0):
 self.score += score
 self.hud.update_score(self.score)

def create_alien_group(self, x, y):
 self.alien_group = AlienGroup(x, y)
 for alien in self.alien_group: # iterable
 self.add(alien)
```

```
def update(self, dt):
 self.collman.clear()
 for _, node in self.children:
 self.collman.add(node)
 if not self.collman.knows(node):
 self.remove(node) # objects not overlapped
 # with the cm rectangle
 self.collide(PlayerShoot.INSTANCE)
 if self.collide(self.player):
 self.respawn_player()

for column in self.alien_group.columns:
 shoot = column.shoot()
 if shoot is not None:
 self.add(shoot)
```

```
for _, node in self.children:
 node.update(dt)
self.alien_group.update(dt)
if random.random() < 0.001:
 self.add(MysteryShip(50, self.height - 50))

def collide(self, node):
 if node is not None:
 for other in self.collman.iter_colliding(node):
 node.collide(other)
 return True

return False
```

```
def respawn_player(self):
 self.lives -= 1
 if self.lives < 0:
 self.unschedule(self.update)
 self.hud.show_game_over()
 else:
 self.create_player()
```

```
class Alien(Actor):

 def load_animation(imgage):
 seq = ImageGrid(load(imgage), 2, 1)
 return Animation.from_image_sequence(seq, 0.5)

TYPES = {

 '1': (load_animation('img/alien1.png'), 40),
 '2': (load_animation('img/alien2.png'), 20),
 '3': (load_animation('img/alien3.png'), 10)
}
```

```
pyglet.image.ImageGrid
 __init__(self, image, rows, columns, ...)
```

```
pyglet.image.Animation.from_image_sequence
(sequence, period, loop=True) # period: second
```

```
def from_type(x, y, alien_type, column):
 animation, score = Alien.TYPES[alien_type]
 return Alien(animation, x, y, score, column)

def __init__(self, img, x, y, score, column=None):
 super(Alien, self).__init__(img, x, y)
 self.score = score
 self.column = column

def on_exit(self):
 super(Alien, self).on_exit()
 if self.column:
 self.column.remove(self)
```

```
class AlienColumn(object):
 def __init__(self, x, y):
 alien_types = enumerate(['3', '3', '2', '2', '1'])
 self.aliens = [Alien.from_type(x, y+i*60,
 alien, self)
 for i, alien in alien_types]

 def should_turn(self, d):
 if len(self.aliens) == 0: enumerate(iterable, start=0)
 return False
 alien = self.aliens[0]
 x, width = alien.x, alien.parent.width
 return x >= width - 50 and d == 1 or \
 x <= 50 and d == -1
```

```
def remove(self, alien):
 self.aliens.remove(alien)

def shoot(self):
 if random.random() < 0.001 and len(self.aliens) > 0:
 pos = self.aliens[0].position
 return Shoot(pos[0], pos[1] - 50)
 return None
```

## ◆ map: applies a function to all the items

- `map(function, list)`
- `items = [0, 1, 2, 3, 4, 5]`
- `squared = list(map(lambda x: x**2, items))`

## ◆ filter: creates a list of elements for which a function returns true

- `nonzero = list(filter(lambda x: x != 0, items))`

## ◆ reduce: applies two arguments cumulatively to the items of iterable, from left to right

- `from functools import reduce`
- `reduce(lambda x, y: x+y, items) # 15`
- `reduce(lambda x, y: x if (x>y) else y, items) # 5`

```
class AlienGroup(object):
 def __init__(self, x, y):
 self.columns = [AlienColumn(x + i * 60, y)
 for i in range(10)]
 self.speed = eu.Vector2(10, 0)
 self.direction = 1
 self.elapsed = 0.0
 self.period = 1.0
```

```
def update(self, elapsed):
 self.elapsed += elapsed
 while self.elapsed >= self.period:
 self.elapsed -= self.period
 offset = self.direction * self.speed
 if self.side_reached():
 self.direction *= -1
 offset = eu.Vector2(0, -10)
 for alien in self:
 alien.move(offset)
```

```
def side_reached(self):
 return any(map(\n lambda c: c.should_turn(self.direction),\n self.columns))

def __iter__(self):
 for column in self.columns:
 for alien in column.aliens:
 yield alien
```

**any()** takes an iterable (list, string, dictionary etc.)

- Return True if at least one element of an iterable is true
- Return False if all elements are false or if an iterable is empty

```
__iter__ # for iterator
for alien in alien_group
 yield and return will return some value from a generator

def my_gen():
 n = 1
 print('This is printed first')
 yield n

 n += 1
 print('This is printed second')
 yield n

 n += 1
 print('This is printed at last')
 yield n

for item in my_gen():
 print(item)
```

```
my_list = [1, 3, 6, 10]

a = (x**2 for x in my_list)
print(next(a))

print(next(a))

print(next(a))

print(next(a))

Output: StopIteration
print(next(a))
```

```
class Shoot(Actor):
 def __init__(self, x, y, img='img/shoot.png'):
 super(Shoot, self).__init__(img, x, y)
 self.speed = eu.Vector2(0, -400)

 def update(self, elapsed):
 self.move(self.speed * elapsed)
```

```
class PlayerShoot(Shoot):
 INSTANCE = None
 def __init__(self, x, y):
 super(PlayerShoot, self).__init__(x, y, 'img/laser.png')
 self.speed *= -1
 PlayerShoot.INSTANCE = self
 def collide(self, other):
 if isinstance(other, Alien):
 self.parent.update_score(other.score)
 other.kill()
 self.kill()
 def on_exit(self):
 super(PlayerShoot, self).on_exit()
 PlayerShoot.INSTANCE = None
```

```
class HUD(cocos.layer.Layer):
 def __init__(self):
 super(HUD, self).__init__()
 w, h = cocos.director.director.get_window_size()
 self.score_text = cocos.text.Label('', font_size=18)
 self.score_text.position = (20, h - 40)
 self.lives_text = cocos.text.Label('', font_size=18)
 self.lives_text.position = (w - 100, h - 40)
 self.add(self.score_text)
 self.add(self.lives_text)
```

```
def update_score(self, score):
 self.score_text.element.text = 'Score: %s' % score

def update_lives(self, lives):
 self.lives_text.element.text = 'Lives: %s' % lives

def show_game_over(self):
 w, h = cocos.director.director.get_window_size()
 game_over = cocos.text.Label('Game Over', font_size=50,
 anchor_x='center',
 anchor_y='center')
 game_over.position = w * 0.5, h * 0.5
 self.add(game_over)
```

```
class MysteryShip(Alien):
 SCORES = [10, 50, 100, 200]

 def __init__(self, x, y):
 score = random.choice(MysteryShip.SCORES)
 super(MysteryShip, self).__init__('img/alien4.png',
 x, y, score)
 self.speed = eu.Vector2(150, 0)

 def update(self, elapsed):
 self.move(self.speed * elapsed)
```

```
if __name__ == '__main__':
 cocos.director.director.init(caption='Cocos Invaders',
 width=800, height=650)
 main_scene = cocos.scene.Scene()
 hud_layer = HUD()
 main_scene.add(hud_layer, z=1)
 game_layer = GameLayer(hud_layer)
 main_scene.add(game_layer, z=0)
 cocos.director.director.run(main_scene)
```

## ◆ Container (Collection)

- Variable number of data items

## ◆ Iteration

- Process of taking an item from container

## ◆ Iterable

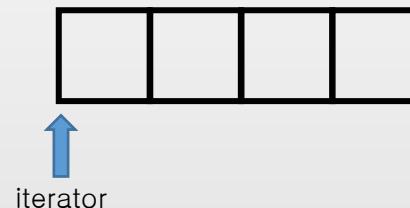
- Object that can be iterated over
- `__iter__` or `__getitem__`

## ◆ Iterator

- Object that enables a user to traverse a container

## ◆ Generator

- Special type of routine to implement iterators



# Iteration, iterable and iterator (2)

```
l = [1, 2, 3, 4, 5] # iterable __iter__
x = iter([1, 2, 3, 4, 5]) # iterator __next__

print(type(x)) # <class 'list_iterator'>
print(next(x)) # 1

for y in l:
 print(y, type(y)) # [y] <class 'int'>
```

# Iteration, iterable and iterator (3)

```
def new_str(str_in):
 length = len(str_in)
 for i in range(0, length):
 yield str_in[i]

for char in new_str("hello"):
 print(char)

list1 = [1, 3, 6, 10]
gen = (x**2 for x in list1)

print(gen)
print(next(gen))
print(next(gen))
```

# Iteration, iterable and iterator (4)

```
class Ex1 :
 def __init__(self):
 self.x = [11, 12, 13, 14, 15]
 def __iter__(self):
 for xx in self.x:
 yield xx

ex1 = Ex1()

for y1 in ex1:
 print(y1) # 11 12 13 14 15

x1 = iter(ex1)
print(next(x1)) # 11
```

# Iteration, iterable and iterator (5)

```
class Ex2 :
 def __init__(self):
 self.x = [21, 22, 23, 24, 25]
 self.i = 0

 def __iter__(self):
 return self

 def __next__(self):
 if self.i < 5:
 self.i += 1
 return self.x[self.i-1]

 else:
 raise StopIteration
```

# Iteration, iterable and iterator (6)

```
ex2 = Ex2()
for y2 in ex2:
 print(y2) # 21 22 23 24 25
```

```
ex2 = Ex2()
x2 = iter(ex2)
print(next(x2)) # 21
```

## 실습 5- Cocos Invaders 성능 향상

- ◆ 좌/우 벽면에서 이동에 오류를 수정
- ◆ Cannon의 laser가 연속 발사되도록 수정 (충돌 반응 포함)
- ◆ 개별적으로 기능 추가

## ◆ Closure

- Function that remembers values in enclosing scope.
- Closures can avoid the use of global values and provides some form of data hiding.

```
def add_number(num):
 def adder(number):
 print('adder is a closure')
 return num + number
 return adder

a_10 = add_number(10)
a_20 = add_number(20)
a1 = a_10(21) # prints 'adder is ...'
a2 = a_20(21) # prints 'adder is ...'
print(a1) #31
print(a2) #41
```

### #Nested function

```
def print1(msg):
 def printer():
 print(msg)
 printer()
```

### #Closure

```
def print2(msg):
 def printer():
 print(msg)
 return printer
```

```
print1("Hello")
ex = print2("Hello")
ex()
```

## ◆ Decorator

- Callable object that is used to modify a function or a class
- Decorators can new function w/o modifying the internal functions

```
from functools import wraps
def NewDecorator(func):
 @wraps(func) # 함수명 및 설명문 유지, 없으면(NewAdd)
 def NewAdd(*args, **kwargs):
 print('Before call')
 result = func(*args, **kwargs)
 print('After call')
 return result
 return NewAdd

@NewDecorator
def add(a, b):
 print('Add')
 return a + b

add2 = NewDecorator(add)
print(add2(1, 3))

sum = add(1, 3) # 'Before call' 'Add' 'After call' 4
print(sum)
print(add.__name__) # @wraps(func)->add(@wraps 사용) or NewAdd
```

# Closure and decorator (3)

```
class ClassDecorator(object):
 def __init__(self, f):
 print('__init__')
 self.f = f

 def __call__(self, *args, **kwargs):
 print('__call__')
 return self.f(*args, **kwargs)

@ClassDecorator
def Fn(a, b):
 print("inside Fn()")
 return a + b

print(Fn(1, 2)) # __init__ __call__ inside Fn() 3
```

# Property decorator (1)

```
class P1:
 def __init__(self, x):
 self.set_x(x)

 def get_x(self):
 return self.x

 def set_x(self, x):
 if x < 0: self.x = 0
 elif x > 1000: self.x = 1000
 else: self.x = x

p1 = P1(1001); print(p1.get_x()) # 1000
p1 = P1(-1); print(p1.get_x()) # 0
p1 = P1(15); print(p1.get_x()) # 15
p1.x = 1001; print(p1.get_x()) # 1001
```

Built-in decorator

# Property decorator (2)

```
class P2:
 def __init__(self, x):
 self.x = x

 @property
 def x(self):
 return self._x

 @x.setter
 def x(self, x):
 if x < 0:
 self._x = 0
 elif x > 1000:
 self._x = 1000
 else:
 self._x = x
```

```
p2 = P2(1001)
print(p2.x) # 1000

p2.x = -12
print(p2.x) # 0
```

```
class C:
 def __init__(self):
 self._x = None

 @property
 def x(self):
 return self._x

 @x.setter
 def x(self, x):
 self._x = x**2

 @x.deleter
 def x(self):
 del self._x
```

Getter, setter, deleter

# Property decorator (4)

```
class C2:
 def __init__(self):
 self._x = None
 def getx(self):
 return self._x
 def setx(self, value):
 self._x = value
 def delx(self):
 del self._x
x = property(getx, setx, delx)
c2 = C2()
c2.x = 10
print(c2.x)
```

# 3. Building a Tower Defense Game

Kyung Hee University  
Daeho Lee

```
import cocos
import cocos.actions as ac

if __name__ == '__main__':
 cocos.director.director.init(caption='Actions 101')

 layer = cocos.layer.Layer()
 sprite = cocos.sprite.Sprite('tank.png', position=(200, 200))
sprite.do(ac.MoveTo((250, 300), 3))
 layer.add(sprite)

 scene = cocos.scene.Scene(layer)
 cocos.director.director.run(scene)
```

# Interval actions (1)

|                         |                                       |
|-------------------------|---------------------------------------|
| Lerp                    | Linear interpolation                  |
| MoveTo, MoveBy          | Moves to (by) position (offset) x, y  |
| JumpTo, JumpBy          | Simulating a jump movement            |
| Bezier                  | Bezier path                           |
| Blink                   | Blinks                                |
| RotateTo, RotateBy      | Rotates to (by) angle (by, clockwise) |
| ScaleTo, ScaleBy        | Scales to (by) zoom factor            |
| FadeOut, FadeIn, FadeTo | Fades out/in/to alpha value           |
| Delay                   | Delays the action                     |
| RandomDelay             | Delays the action randomly            |

```
#sprite.do(ac.MoveTo((250, 300), 3))
sprite.do(ac.JumpTo((500, 200), 100, 5, 3))
 #position, height, jumps, duration
#sprite.do(ac.RotateTo(90, 2))
#sprite.do(ac.ScaleTo(2, 2))
#sprite.do(ac.FadeIn(3))
#sprite.do(ac.Delay(2)+ac.MoveTo((250, 300), 3))
#sprite.do(ac.Blink(10, 2)) # times, duration
```

|                  |                                                        |
|------------------|--------------------------------------------------------|
| Place            | Places the target in the position (x, y)               |
| CallFunc         | Calls a function                                       |
| CallFuncS        | Calls a function with the target as the first argument |
| Hide             | Hides, visibility to False                             |
| Show             | Shows, visibility to True                              |
| ToggleVisibility | Toggles visibility                                     |

```
def func1():
 print ("hello")
action = CallFunc(func1)
sprite.do(action)
def func2(sprite):
 print ("hello")
action = CallFuncS(func2)
sprite.do(action)
```

# Combining actions

```
ac.MoveBy((80, 0), 3) + ac.Delay(3) + \
ac.CallFunc(sprite.kill)
```

```
ac.MoveBy((80, 0), 3) | ac.RotateBy(90, 2)
```

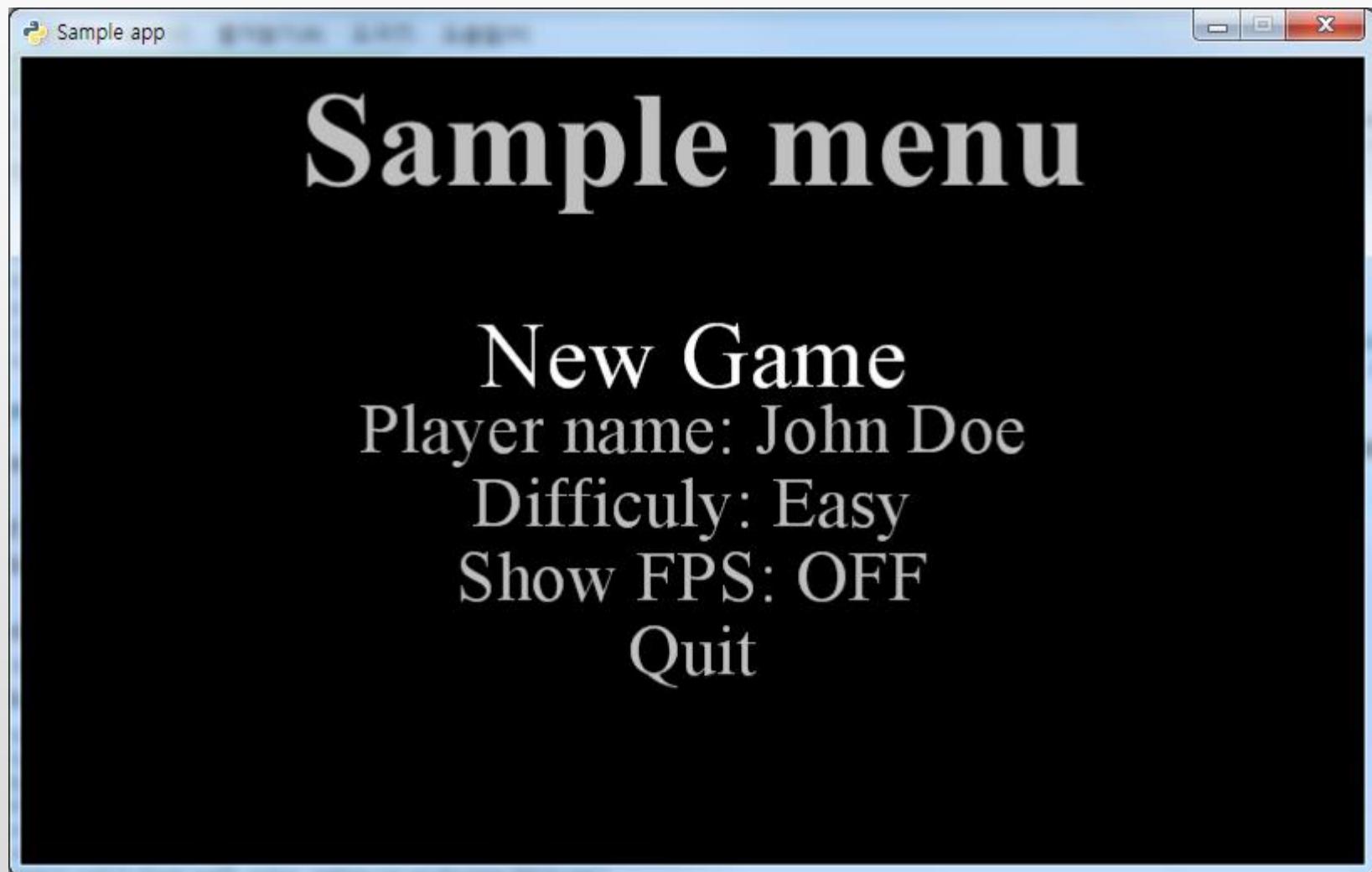
```
ac.MoveBy((80, 0), 3) | ac.RotateBy(90, 2) + \
ac.CallFunc(sprite.kill)
```

```
class Hit(ac.IntervalAction):
 def init(self, duration=0.5):
 self.duration = duration

 def update(self, t):
 self.target.color = (255, 255 * t, 255 * t)

 # t range (0, 1)
 # start(), stop()
```

# Adding a main menu (1)



## ◆ Cocos.menu

- MenuItem(label, callback, \*args, \*\*kwargs)
- EntryMenuItem(label, callback\_func, value, max\_length=0)
- ToggleMenuItem(label, callback\_func, value=False)
- MultipleMenuItem(label, callback\_func, items, default\_item=0)
  - ✓ items = ['Mute','10','20','30','40','50','60','70','80','90','100']
- ImageMenuItem(image, callback\_func, \*args, \*\*kwargs)
- ColorMenuItem(label, callback\_func, items, default\_item=0)
  - ✓ items = [(255, 255, 255), (100, 200, 100), (200, 50, 50)]

# Adding a main menu (3)

```
import cocos
from cocos.menu import *
import pyglet.app

class MainMenu(Menu):
 def __init__(self):
 super(MainMenu, self).__init__('Sample menu')
 self.font_title['font_name'] = 'Times New Roman'
 self.font_title['font_size'] = 60
 self.font_title['bold'] = True
 self.font_item['font_name'] = 'Times New Roman'
 self.font_item_selected['font_name'] = 'Times New Roman'
```

# Adding a main menu (4)

```
self.difficulty = ['Easy', 'Normal', 'Hard']
m1 = MenuItem('New Game', self.start_game)
m2 = EntryMenuItem('Player name:', self.set_player_name,
 'John Doe', max_length=10)
m3 = MultipleMenuItem('Difficulty: ', self.set_difficulty,
 self.difficulty)
m4 = ToggleMenuItem('Show FPS: ', self.show_fps, False)
m5 = MenuItem('Quit', pyglet.app.exit)
self.create_menu([m1, m2, m3, m4, m5],
 shake(), shake_back())
selected effect, unselected effect
zoom_in(), zoom_out()
```

# Adding a main menu (5)

```
def start_game(self):
 print('Starting a new game!')

def set_player_name(self, name):
 print('Player name:', name)

def set_difficulty(self, index):
 print('Difficulty set to', self.difficulty[index])

def show_fps(self, val):
 cocos.director.director.show_FPS = val

if __name__ == '__main__':
 cocos.director.director.init(caption='Sample app', width=800)
 scene = cocos.scene.Scene(MainMenu())
 cocos.director.director.run(scene)
```

- ◆ <https://www.zapsplat.com/sound-effect-category/game-sounds/>
- ◆ <https://freesound.org/browse/tags/game/>
- ◆ <http://soundbible.com/free-sound-effects-1.html>

```
import cocos.audio

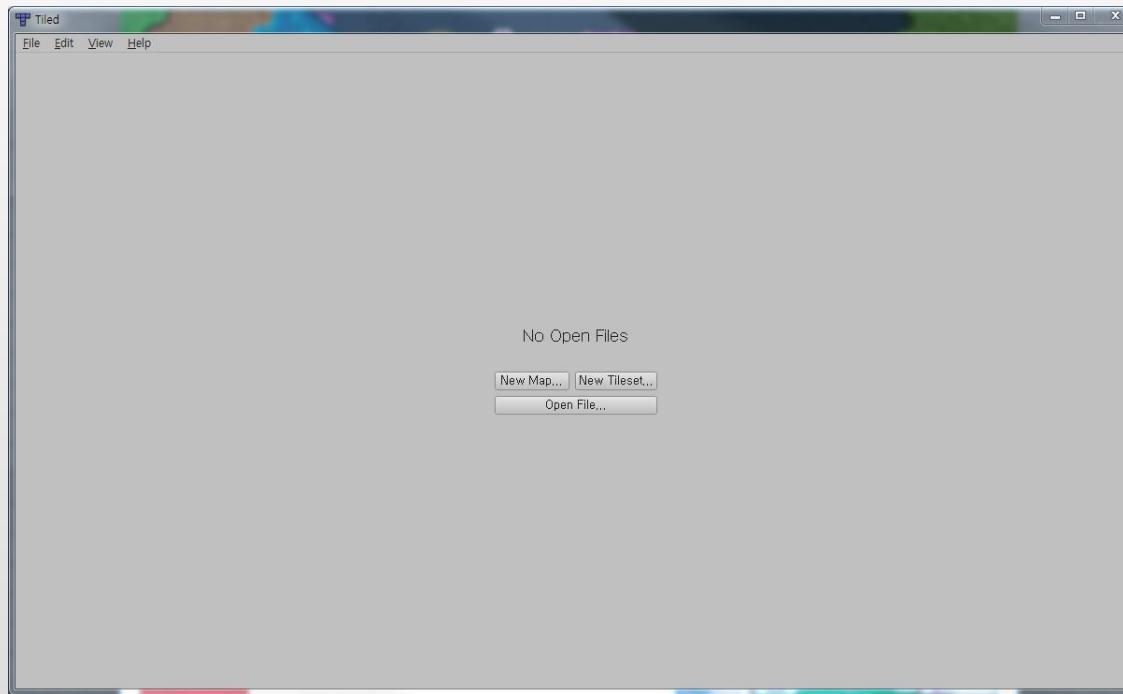
import cocos.audio.pygame #pip install pygame
import cocos.audio.pygame.mixer

cocos.audio.pygame.mixer.init()
sound = cocos.audio.pygame.mixer.Sound('ex.ogg')
channel = sound.play()

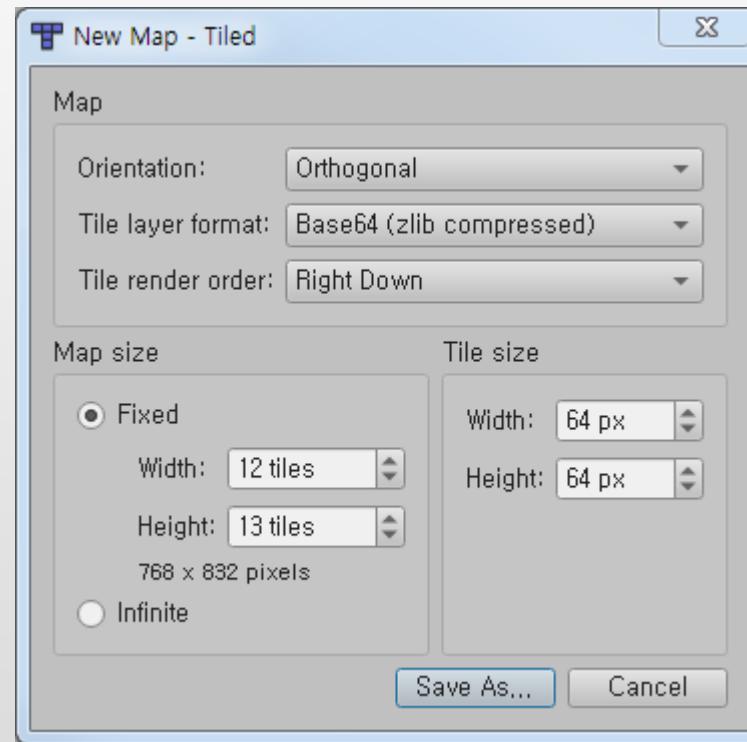
 pygame.init()
 self.fallSound =
 pygame.mixer.Sound('resources/fall.ogg')
 pygame.mixer.Sound.play(self.fallSound)
```

# Tiled Map Editor (1)

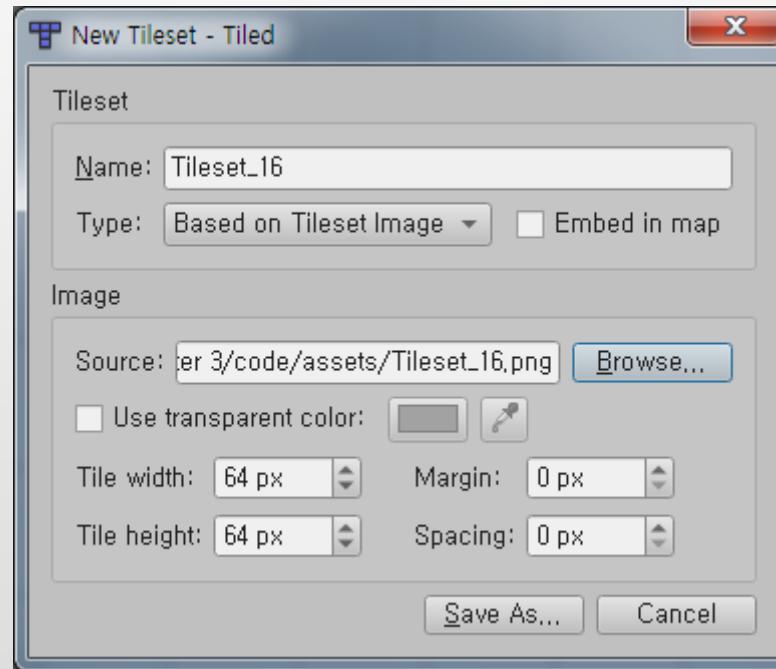
◆ <http://www.mapeditor.org/>



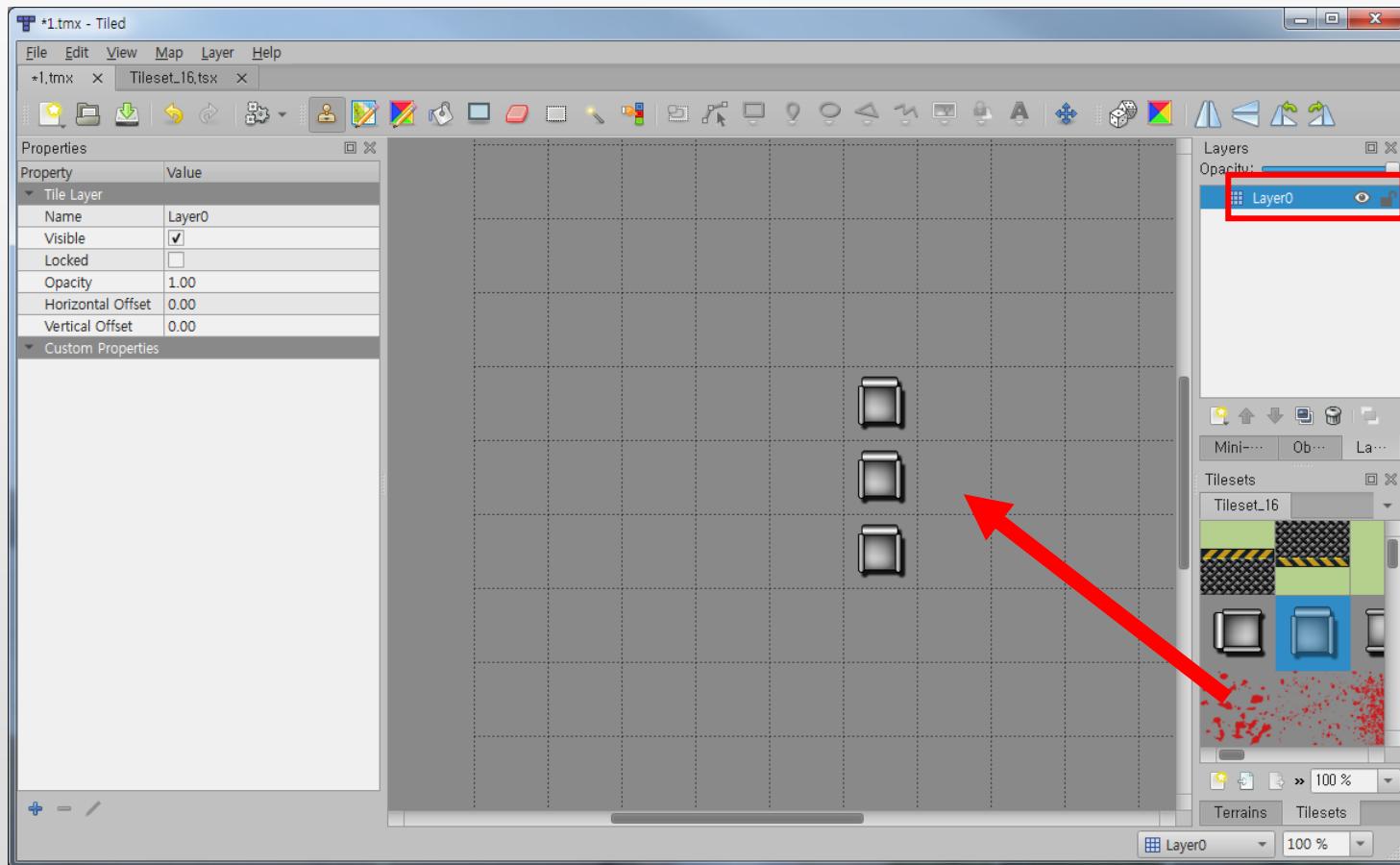
## ◆ [File]-[New Map...]



## ❖ [File]-[New Tileset...]



# Tiled Map Editor (4)

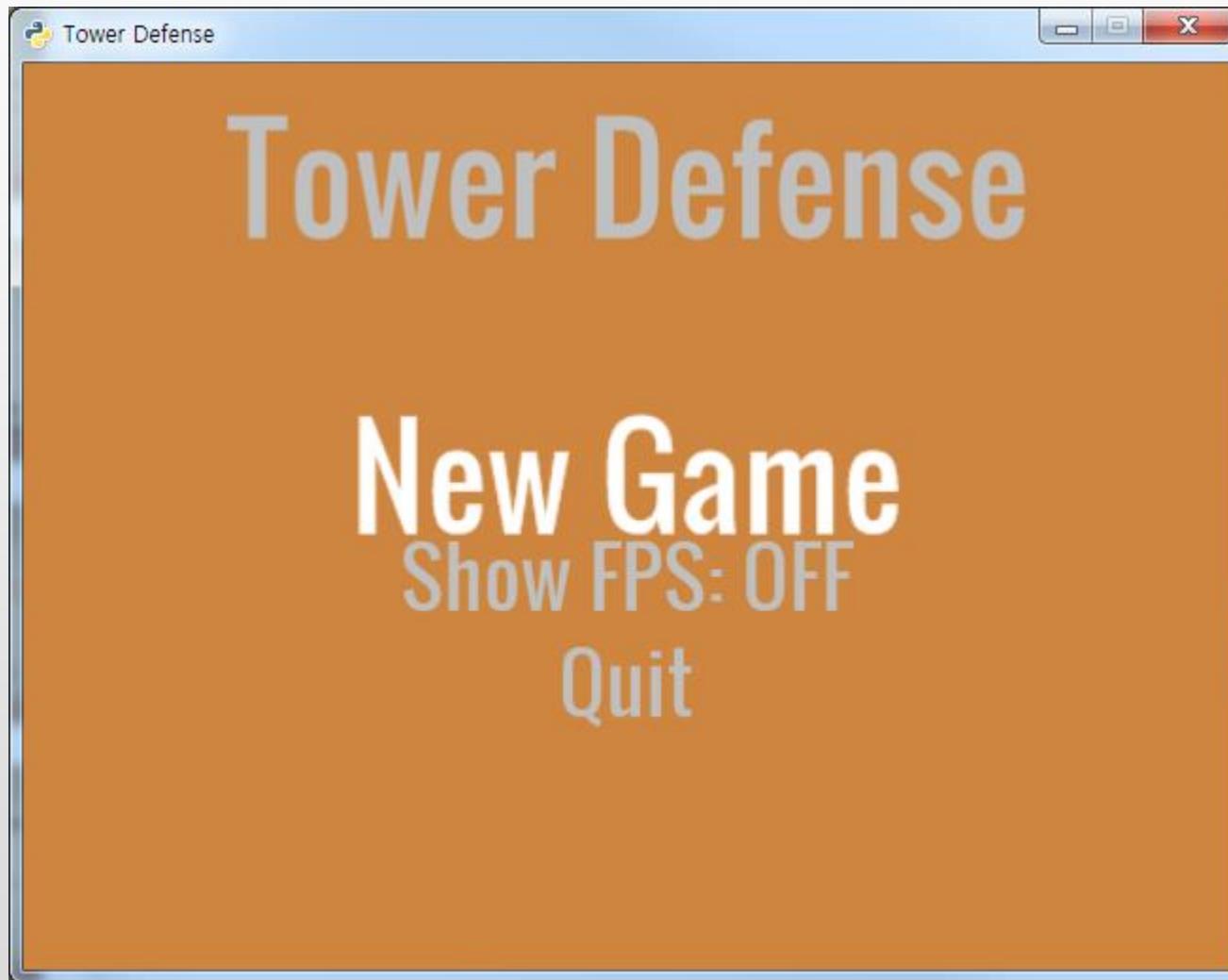


# Loading and display tiles

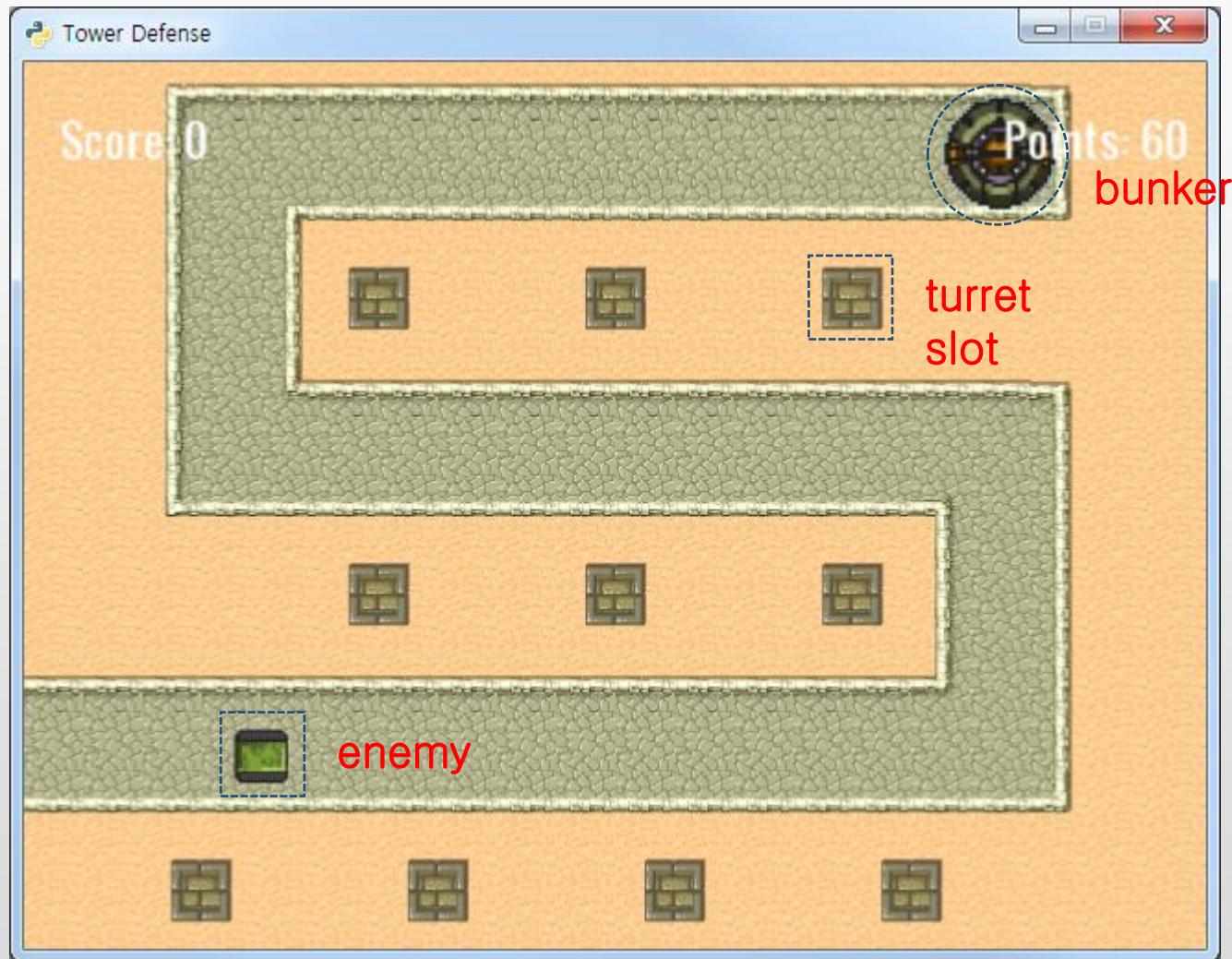
```
main_scene = cocos.scene.Scene()

background_layer = cocos.layer.ColorLayer(100, 120, 95, 255)
main_scene.add(background_layer, z = 0)

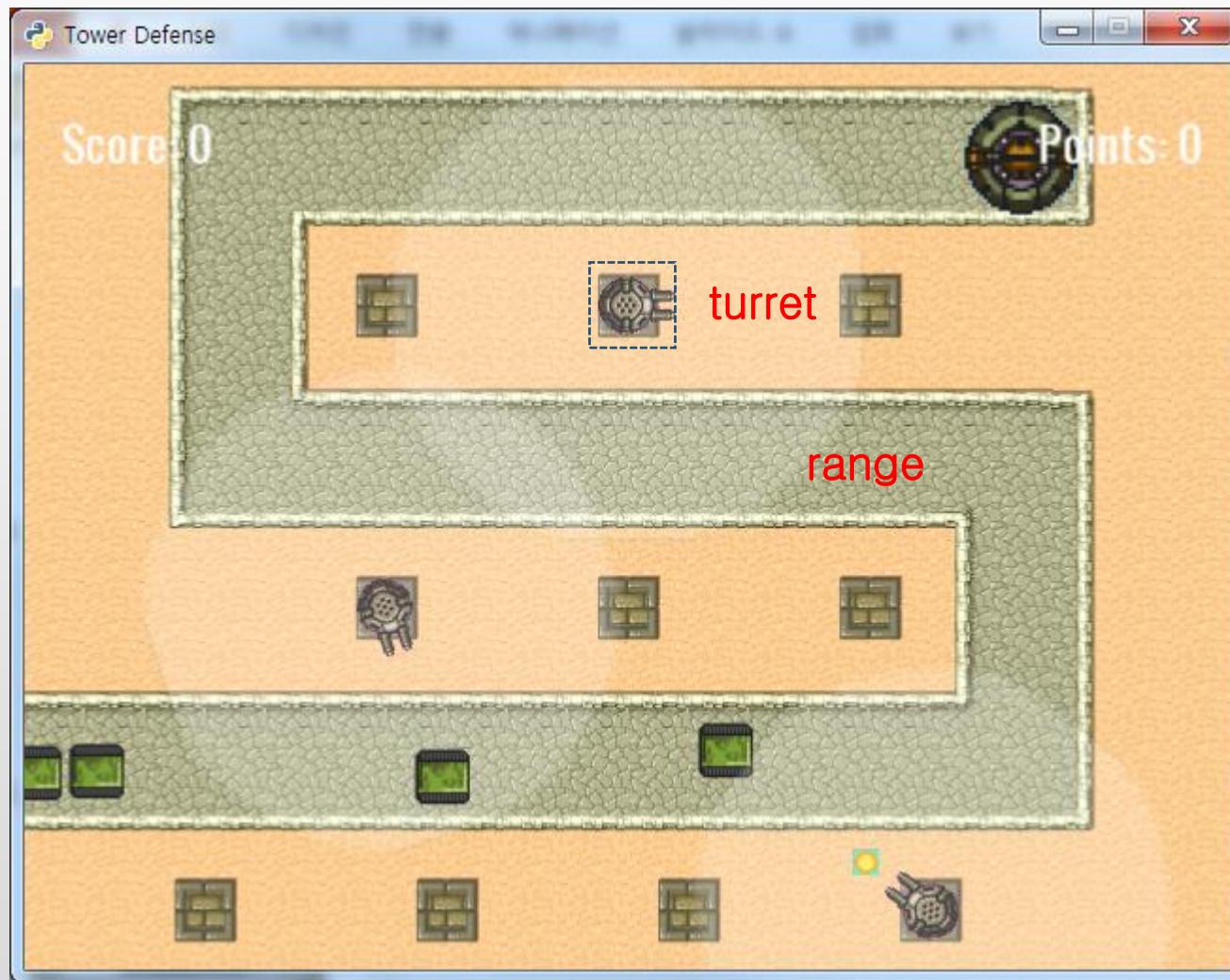
tmx_map = cocos.tiles.load('assets/ArcadeMap.tmx')
bg0 = tmx_map['map0'] #layer
bg1 = tmx_map['map1'] #layer
bg0.set_view(0, 0, bg0.px_width, bg0.px_height)
bg1.set_view(0, 0, bg1.px_width, bg1.px_height)
main_scene.add(bg0, z = 1)
main_scene.add(bg1, z = 1)
```



# Tower defense (2)



# Tower defense (3)



```
scenario.py, 배경 가져오기, actors 설정, action 설정
```

```
import cocos.tiles
```

```
import cocos.actions as ac
```

```
RIGHT = ac.RotateBy(90, 1) # action
```

```
LEFT = ac.RotateBy(-90, 1) # action
```

```
def move(x, y): # action
```

```
 dur = abs(x+y) / 100.0
```

```
 return ac.MoveBy((x, y), duration=dur)
```

# Game scenario (2)

```
class Scenario(object): # object defined in new_game function
 # by get_scenario function
 def __init__(self, tmx_map, turrets, bunker, enemy_start):
 self.tmx_map = tmx_map #string for layer, 'map0'
 self.turret_slots = turrets
 self.bunker_position = bunker
 self.enemy_start = enemy_start
 self._actions = None

 def get_background(self): # create background
 # called in new_game function
 tmx_map = cocos.tiles.load('assets/tower_defense.tmx')
 bg = tmx_map[self.tmx_map]
 bg.set_view(0, 0, bg.px_width, bg.px_height)
 return bg
```

## @property

```
def actions(self):
 return self._actions
```

## @actions.setter

```
def actions(self, actions):
 self._actions = ac.RotateBy(90, 0.5) # 초기 동작 추가
 for step in actions:
 self._actions += step
```

```
def get_scenario(): # create scenario
 turret_slots = [(192, 352), (320, 352), (448, 352),
 (192, 192), (320, 192), (448, 192),
 (96, 32), (224, 32), (352, 32), (480, 32)]
 bunker_position = (528, 430)
 enemy_start = (-80, 110)
 sc = Scenario('map0', turret_slots,
 bunker_position, enemy_start)
 sc.actions = [move(610, 0), LEFT, move(0, 160),
 LEFT, move(-415, 0), RIGHT,
 move(0, 160), RIGHT, move(420, 0)]
 return sc # Scenario class instance
```

```
import cocos.menu
import cocos.scene
import cocos.layer
import cocos.actions as ac
from cocos.director import director
from cocos.scenes.transitions import FadeTRTransition

import pyglet.app

from gamelayer import new_game # new_game function
```

```
class MainMenu(cocos.menu.Menu):
 def __init__(self):
 super(MainMenu, self).__init__('Tower Defense')

 self.font_title['font_name'] = 'Oswald'
 self.font_item['font_name'] = 'Oswald'
 self.font_item_selected['font_name'] = 'Oswald'

 self.menu_anchor_y = 'center'
 self.menu_anchor_x = 'center'
```

```
items = list()
items.append(cocos.menu.MenuItem('New Game',
 self.on_new_game))
items.append(cocos.menu.ToggleMenuItem('Show FPS: ',
 self.show_fps, director.show_FPS))
items.append(cocos.menu.MenuItem('Quit', pyglet.app.exit))

self.create_menu(items, ac.ScaleTo(1.25, duration=0.25),
 ac.ScaleTo(1.0, duration=0.25))

def on_new_game(self):
 # fade to top-right (TR), push: running scene is pushed
 director.push(FadeTRTransition(new_game(), duration=2))
def show_fps(self, val):
 director.show_FPS = val
```

```
def new_menu(): # create menu
 scene = cocos.scene.Scene()
 color_layer = cocos.layer.ColorLayer(205, 133, 63, 255)
 scene.add(MainMenu(), z=1)
 scene.add(color_layer, z=0)
 return scene
```

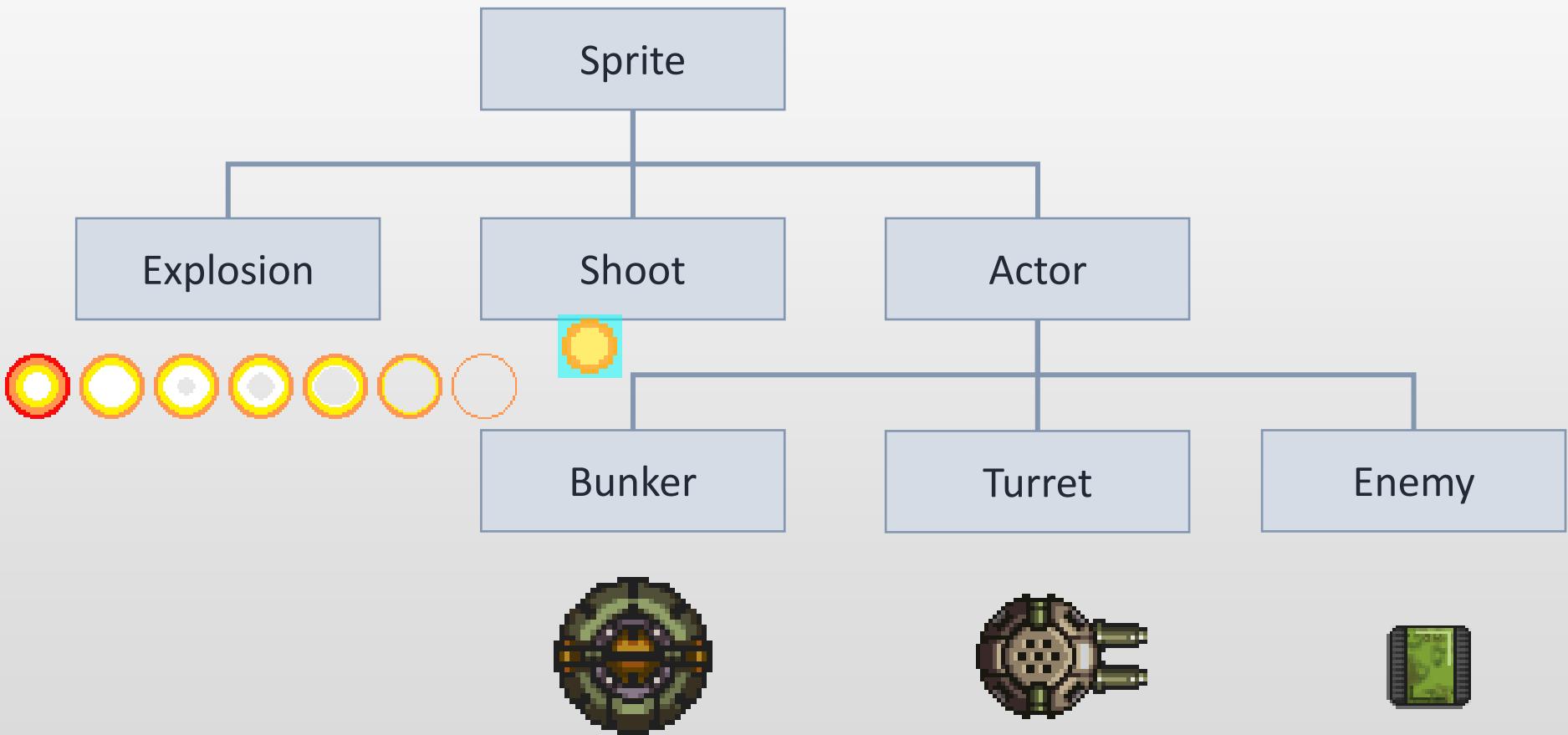
# new\_game() and game\_over() (1)

```
def new_game(): # gamelayer.py
 scenario = get_scenario()
 background = scenario.get_background()
 hud = HUD()
 game_layer = GameLayer(hud, scenario)
 return cocos.scene.Scene(background, game_layer, hud)
 # cocos.scene.Scene(*children)
```

# new\_game() and game\_over() (2)

```
def game_over(): # gamelayer.py, called GameLayer
 # director.replace(SplitColsTransition(game_over()))
w, h = director.get_window_size()
layer = cocos.layer.Layer()
text = cocos.text.Label('Game Over', position=(w*0.5, h*0.5),
 font_name='Oswald', font_size=72,
 anchor_x='center', anchor_y='center')
layer.add(text)
scene = cocos.scene.Scene(layer)
new_scene = FadeTransition(mainmenu.new_menu())
replace: replaces running scene with new scene
func = lambda: director.replace(new_scene)
scene.do(ac.Delay(3) + ac.CallFunc(func))
return scene
```

# Actors (1)



```
import math

import cocos.sprite
import cocos.audio
import cocos.actions as ac
import cocos.euclid as eu
import cocos.collision_model as cm

import pyglet.image
from pyglet.image import Animation

raw = pyglet.image.load('assets/explosion.png')
seq = pyglet.image.ImageGrid(raw, 1, 8)
explosion_img = Animation.from_image_sequence(seq, 0.07, False)
```

```
class Explosion(cocos.sprite.Sprite): # Explosion animation
 def __init__(self, pos):
 super(Explosion, self).__init__(explosion_img, pos)
 # animation : delay + kill
 self.do(ac.Delay(1) + ac.CallFunc(self.kill))
```

```
class Shoot(cocos.sprite.Sprite):# shoot to designated target
 def __init__(self, pos, offset, target):
 super(Shoot, self).__init__('shoot.png',
 position=pos)
 self.do(ac.MoveBy(offset, 0.1) +
 ac.CallFunc(self.kill) +
 ac.CallFunc(target.hit))
```

```
class Hit(ac.IntervalAction): # hit action
 def init(self, duration=0.5):
 self.duration = duration

 def update(self, t):
 self.target.color = (255, 255 * t, 255 * t)

class TurretSlot(object):
 def __init__(self, pos, side):
 # for mouse event
 # detect slots including mouse position
 self.cshape = cm.AARectShape(pos, side*0.5, side*0.5)
 #(eu.Vector2(*pos), side*0.5, side*0.5) # original
```

```
class Actor(cocos.sprite.Sprite):
 def __init__(self, img, x, y):
 super(Actor, self).__init__(img, position=(x, y))
 self._cshape = cm.CircleShape(self.position,
 self.width * 0.5)

 @property #getter
 def cshape(self):
 self._cshape.center = eu.Vector2(self.x, self.y)
 return self._cshape
```

```
class Turret(Actor):

 def __init__(self, x, y):
 super(Turret, self).__init__('turret.png', x, y)
 self.add(cocos.sprite.Sprite('range.png', opacity=50,
 scale=5)) # attack range
 self.cshape.r = 125.0 # attack range
 self.target = None # target within range
 self.period = 2.0 # reload period
 self.reload = 0.0 # elapse reload
 self.schedule(self._shoot)
```

```
def _shoot(self, dt): # by self.schedule
 if self.reload < self.period:
 self.reload += dt # elapse reload
 elif self.target is not None:
 self.reload -= self.period
 offset = eu.Vector2(self.target.x - self.x,
 self.target.y - self.y)
 # pos = self.cshape.center #①
 pos = self.cshape.center + offset.normalized() * 20 #②
 # Layer add
 self.parent.add(Shoot(pos, offset, self.target))
```



```
def collide(self, other):
 self.target = other
 if self.target is not None:
 # angle for rotation, radian → degree
 # rotation by sprite.rotation
 x, y = other.x - self.x, other.y - self.y
 angle = -math.atan2(y, x) # CCW
 # CW(self.x - other.x, self.y - other.y)
 self.rotation = math.degrees(angle) # CW
```

```
class Enemy(Actor):
 def __init__(self, x, y, actions):
 super(Enemy, self).__init__('tank.png', x, y)
 self.health = 100
 self.score = 20
 self.destroyed = False
 self.do(actions)

 def hit(self):
 self.health -= 25
 self.do(Hit())
 # is_running: whether or not the object is running
 if self.health <= 0 and self.is_running:
 self.destroyed = True
 self.explode()
```

```
def explode(self):
 self.parent.add(Explosion(self.position))
 self.kill()

class Bunker(Actor):
 def __init__(self, x, y):
 super(Bunker, self).__init__('bunker.png', x, y)
 self.hp = 100 # health points
 def collide(self, other):
 if isinstance(other, Enemy):
 self.hp -= 10
 other.explode()
 if self.hp <= 0:
 self.kill()
```

```
import random

from cocos.director import director
from cocos.scenes.transitions import SplitColsTransition,
FadeTransition
import cocos.layer
import cocos.scene
import cocos.text
import cocos.actions as ac
import cocos.collision_model as cm

import actors
import mainmenu
from scenario import get_scenario
```

```
class GameLayer(cocos.layer.Layer):
 is_event_handler = True

 def __init__(self, hud, scenario):
 super(GameLayer, self).__init__()
 self.hud = hud
 self.scenario = scenario
 self.score = self._score = 0
 self.points = self._points = 60
 self.turrets = []
```

```
w, h = director.get_window_size()
cell_size = 32
self.coll_man = cm.CollisionManagerGrid(0, w, 0, h,
 cell_size, cell_size)
self.coll_man_slots = cm.CollisionManagerGrid(0, w,
 0, h, cell_size, cell_size) # for mouse click
for slot in scenario.turret_slots:
 self.coll_man_slots.add(actors.TurretSlot(slot,
 cell_size))

self.bunker = actors.Bunker(*scenario.bunker_position)
self.add(self.bunker)
self.schedule(self.game_loop)
```

```
@property
def points(self):
 return self._points
@points.setter
def points(self, val):
 self._points = val
 self.hud.update_points(val) # point 변경시 hud 변경

@property
def score(self):
 return self._score
@score.setter
def score(self, val):
 self._score = val
 self.hud.update_score(val)
```

```
def game_loop(self, _): # by self.schedule
 self.coll_man.clear()
 for obj in self.get_children():
 if isinstance(obj, actors.Enemy):
 self.coll_man.add(obj)
 for turret in self.turrets:
 obj = \ # next(iterator[, default])
 # turret당 1개의 enemy
 next(self.coll_man.iter_colliding(turret), None)
 turret.collide(obj)
 for obj in self.coll_man.iter_colliding(self.bunker):
 self.bunker.collide(obj)
 if random.random() < 0.005:
 self.create_enemy()
```

```
def create_enemy(self):
 enemy_start = self.scenario.enemy_start
 x = enemy_start[0] + random.uniform(-10, 10)
 y = enemy_start[1] + random.uniform(-10, 10)
 self.add(actors.Enemy(x, y, self.scenario.actions))

#def on_mouse_press(self, x, y, buttons, mod):
def on_mouse_release(self, x, y, buttons, mod):
 print(x,y)
 slots = self.coll_man_slots objs_touching_point(x, y)
 if len(slots) and self.points >= 20: # slots는 element가 1개인 set
 self.points -= 20
 slot = next(iter(slots)) # slot = list(slots)[0]
 turret = actors.Turret(*slot.cshape.center)
 self.turrets.append(turret)
 self.add(turret)
```

```
def remove(self, obj): # This function is called
 # by remove(obj) or obj.kill()

 if obj is self.bunker:
 director.replace(SplitColsTransition(game_over()))
 elif isinstance(obj, actors.Enemy) and obj.destroyed:
 self.score += obj.score
 self.points += 5
 super(GameLayer, self).remove(obj)
```

```
class HUD(cocos.layer.Layer):
 def __init__(self):
 super(HUD, self).__init__()
 w, h = director.get_window_size()
 self.score_text = self._create_text(60, h-40)
 self.score_points = self._create_text(w-60, h-40)

 def _create_text(self, x, y):
 text = cocos.text.Label(font_size=18,
 font_name='Oswald', anchor_x='center',
 anchor_y='center')
 text.position = (x, y)
 self.add(text)
 return text
```

```
def update_score(self, score):
 self.score_text.element.text = 'Score: %s' % score

def update_points(self, points):
 self.score_points.element.text = 'Points: %s' % points
```

```
from cocos.director import director
import pyglet.font
import pyglet.resource
from mainmenu import new_menu

if __name__ == '__main__':
 pyglet.resource.path.append('assets')
 pyglet.resource.reindex()
 pyglet.font.add_file('assets/Oswald-Regular.ttf')

 director.init(caption='Tower Defense')
 director.run(new_menu())
```

◆ <https://opengameart.org/>

 **OPENGAMEART.ORG**

Home Browse Submit Art Collect Forums FAQ Leaderboards

Follow us:    

**CHAT WITH US!**

IRC/Webchat Rules  
IRC: #OpenGameArt on irc.freenode.net

**Chat from your browser!**

**ACTIVE FORUM TOPICS - (VIEW MORE)** 

- Bug? Posts Cannot Handle Raw Unicode Characters *6 hours 56 min ago* by AntumDeluge
- Where do I find the cat figurine? *8 hours 8 min ago* by Flarefan
- Spooky (due October 29) *9 hours 53 min ago* by Cougarmint
- Music browsing *15 hours 4 min ago* by Invisible Man
- Unable to Upload *1 day 2 hours ago* by AntumDeluge
- TurboSquid merrily helping to sell stolen content... again *1 day 2 hours ago* by fina17
- Pixelart Scaler (ScaleNx and EagleNx) for GIMP *1 day 8 hours ago* by AntumDeluge
- Can't Remove Node from Collection *1 day 9 hours ago* by AntumDeluge

**SUPPORT OPENGAMEART.ORG ON PATREON!**

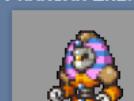
Read about the funding drive or go right to Patreon.

Final goal: **\$2000/month**  
Current goal: **Better Content Curation (\$1000/month)**

\$ 506.27/\$1000 (51%)

Last updated Fri, 06 Apr 2018 04:05:18 -0400. For current progress, see our Patreon page.

**POPULAR THIS WEEK - (VIEW MORE)** 

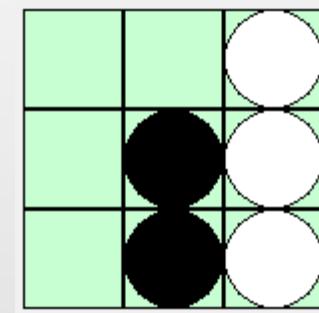
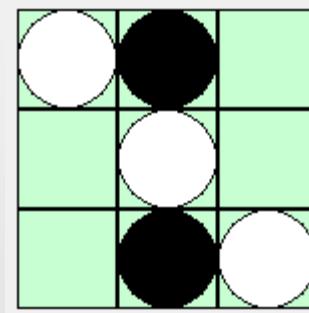
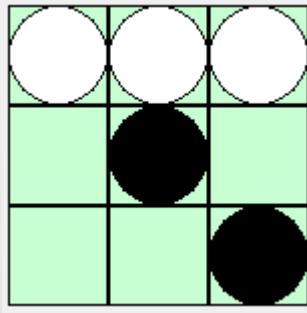
|                                                                                                              |                                                                                                                |                                                                                                               |                                                                                                                |
|--------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| <b>INCA TILESET</b><br>     | <b>LOWPOLY SHIP...</b><br>  | <b>2D GAME-ART...</b><br>  | <b>PHARAOH ENE...</b><br>   |
| <b>TOP-DOWN TIL...</b><br> | <b>GIANT CRAB R...</b><br> | <b>LOWPOLY MOD...</b><br> | <b>RPG - THE SE...</b><br> |

# 8. Strategy Board Game by AI (Othello, Reversi)

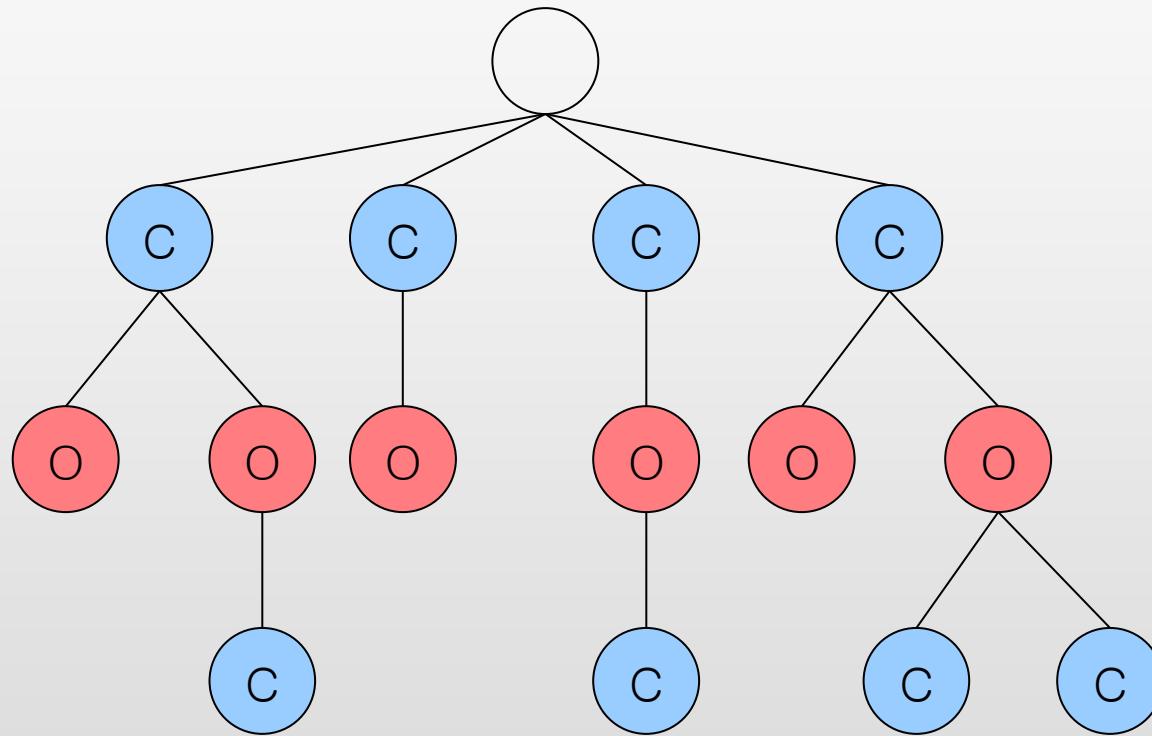
- 실습 6

Kyung Hee University  
Daeho Lee

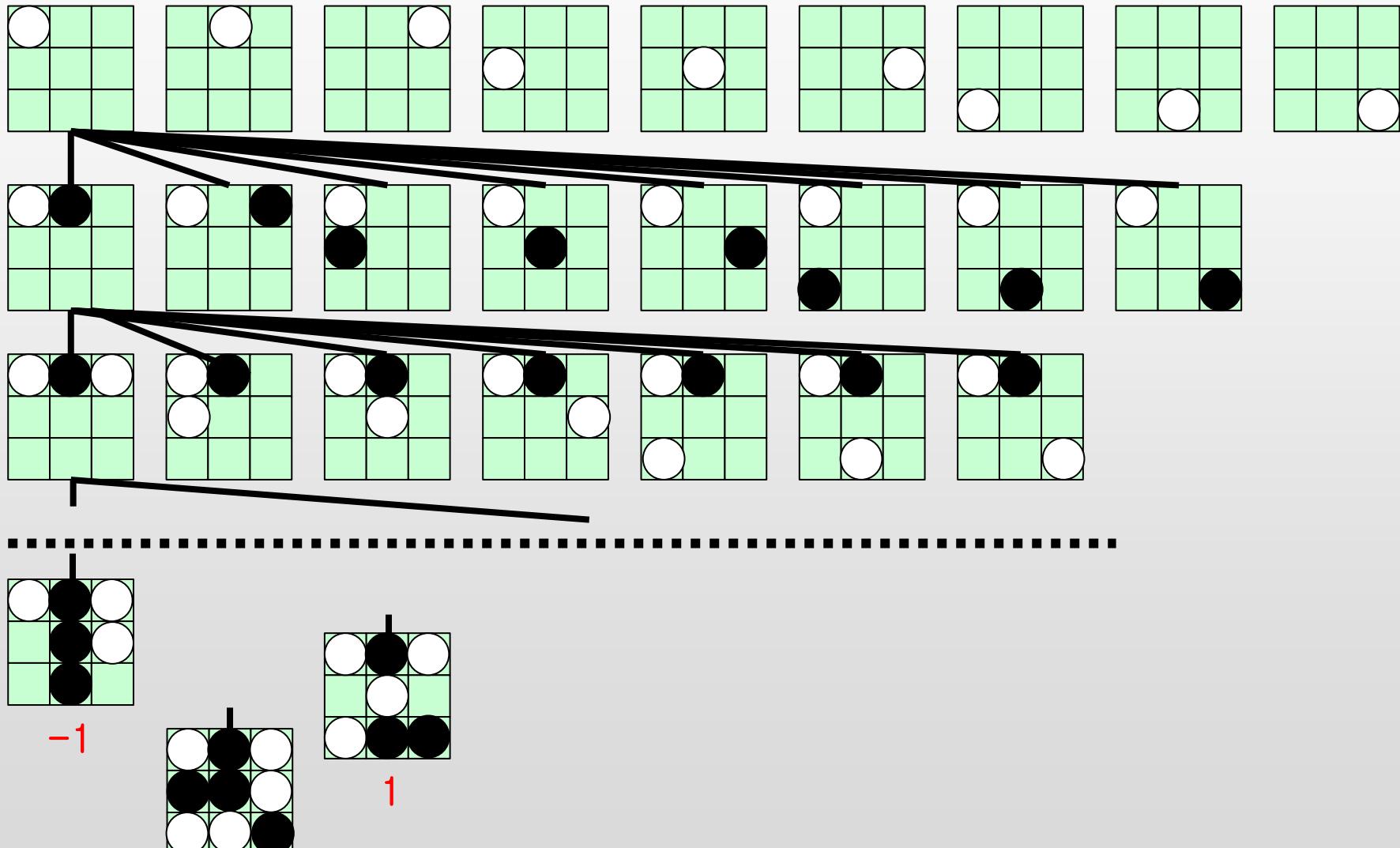
## ◆ Tic-Tac-Toe



## 2. Game tree

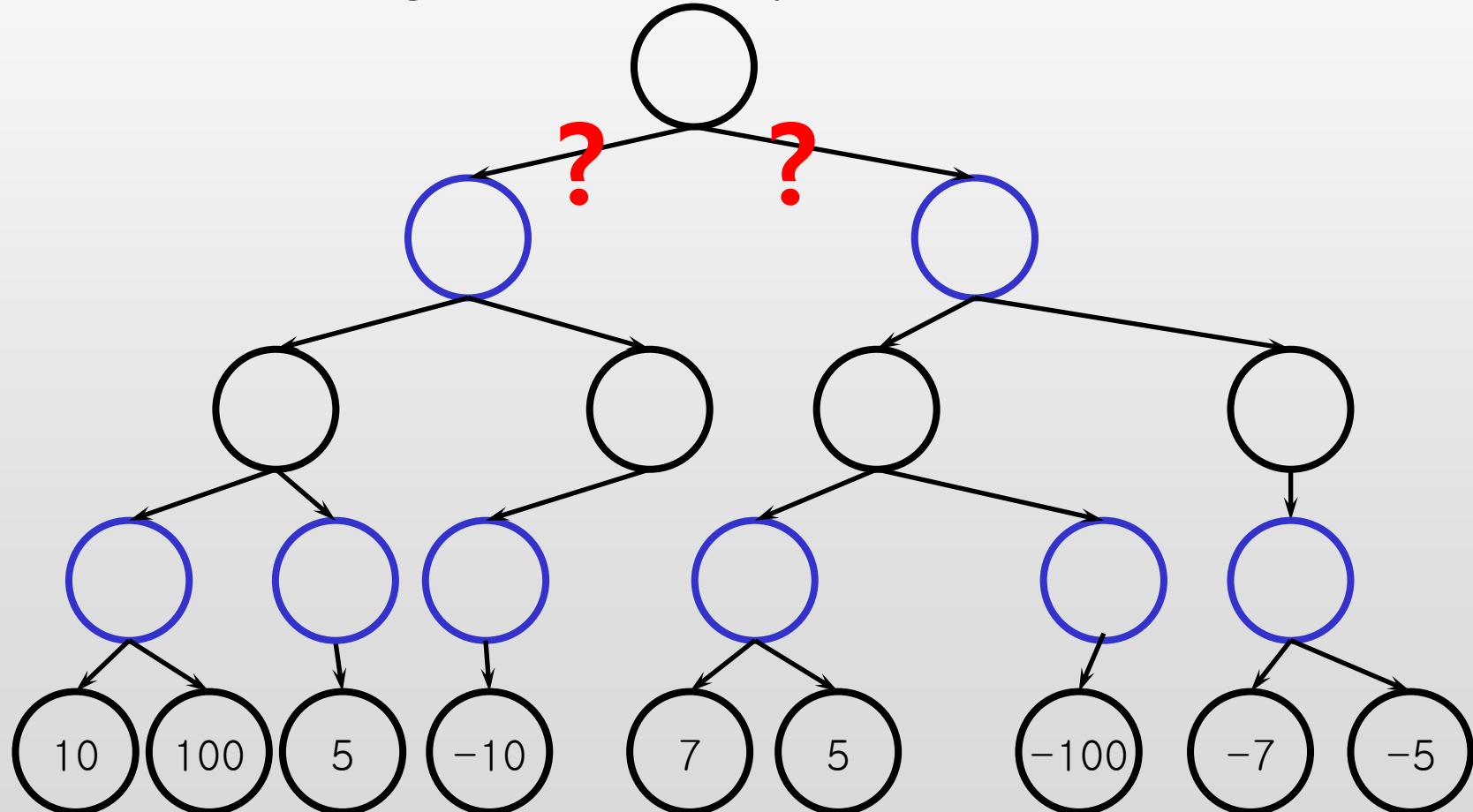


# Tic-Tac-Toe (3)



## ◆ Minimax

- Minimizing the maximum possible loss



- Expand the game tree as far as possible
- Assign state evaluations at each open node
- Propagate upwards the minimax choices
  - ✓ If the parent is a Min node (opponent), propagate up the minimum value of the children
  - ✓ If the parent is a Max node (computer), propagate up the maximum value of the children

## 2. Complexity of Minimax

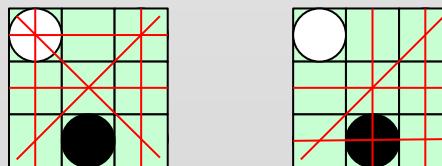
### ➤ Tic-Tac-Toe

- ✓ Computer goes first:  $9! = 362,880$  nodes
- ✓  $(5 \times 5)$ :  $25! = 15,511,210,043,330,985,984,000,000$
- ✓  $(9 \times 9)$ :  $81! \approx 5.8 \times 10^{120}$

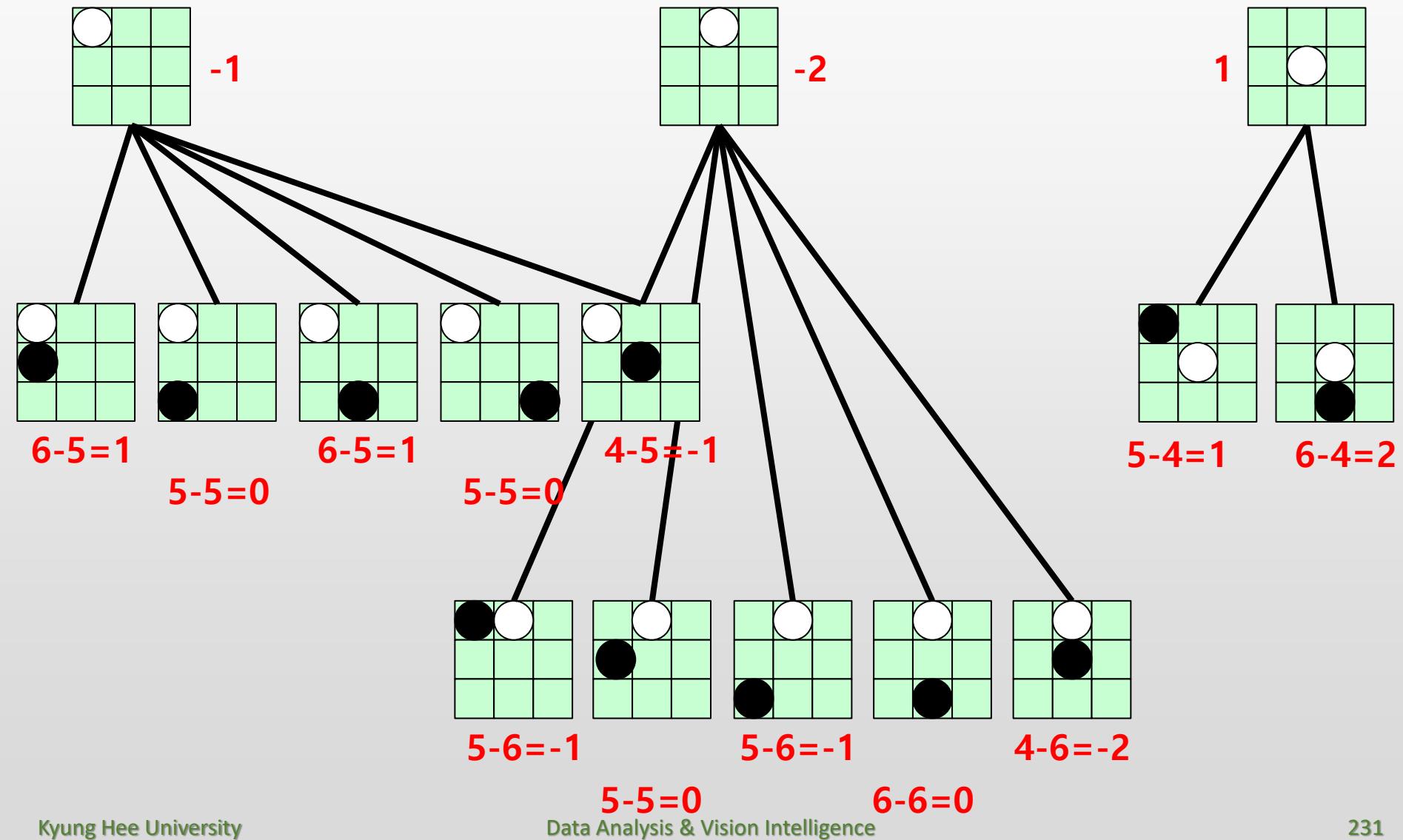
### ➤ Static (heuristic) evaluation functions (휴리스틱 평가 함수)

#### ✓ Tic-Tac-Toe

- Computer's win paths:  $C(n)$
- Opponent's win paths:  $O(n)$
- Evaluation function:  $E(n) = C(n) - O(n)$

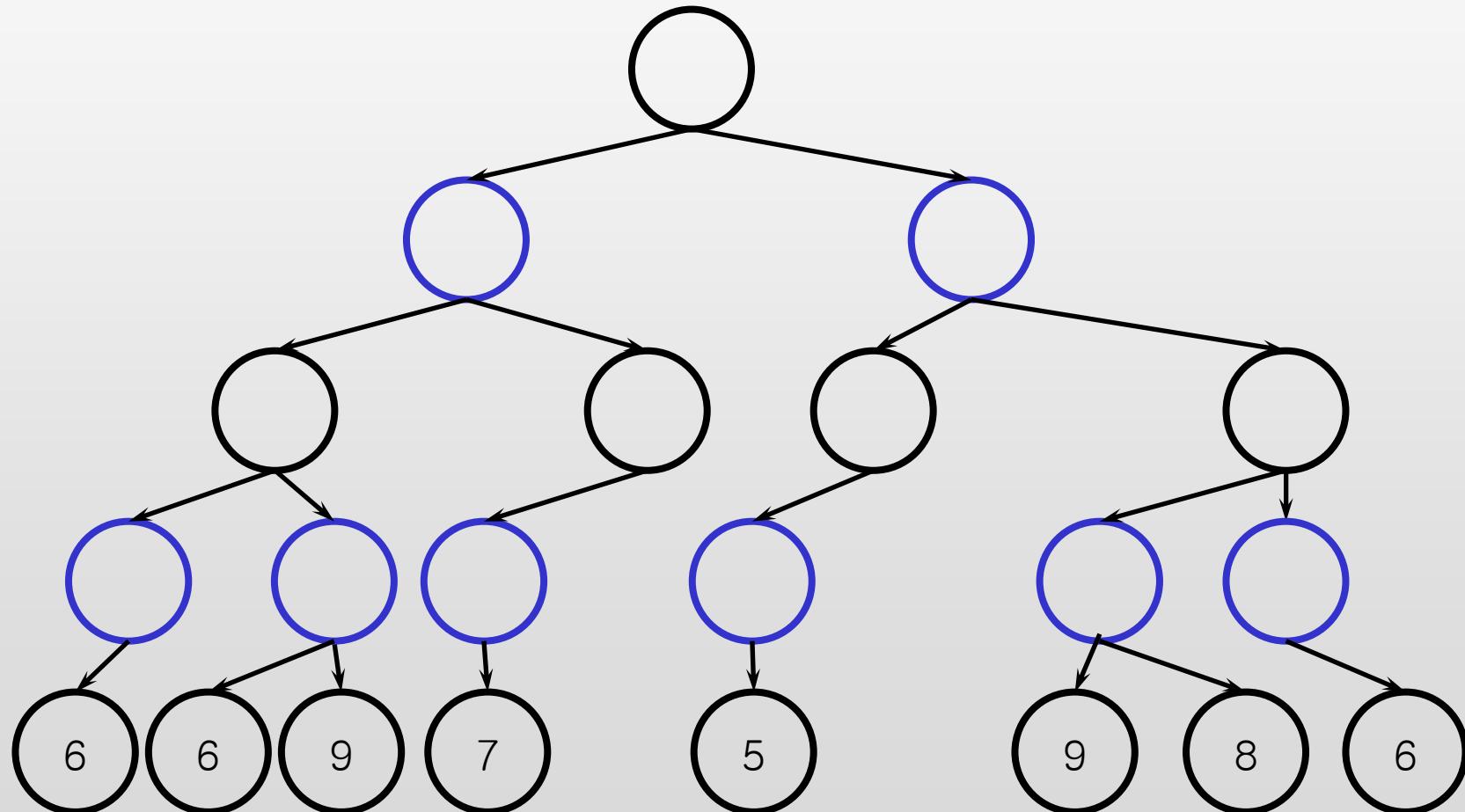


# Minimax (4)



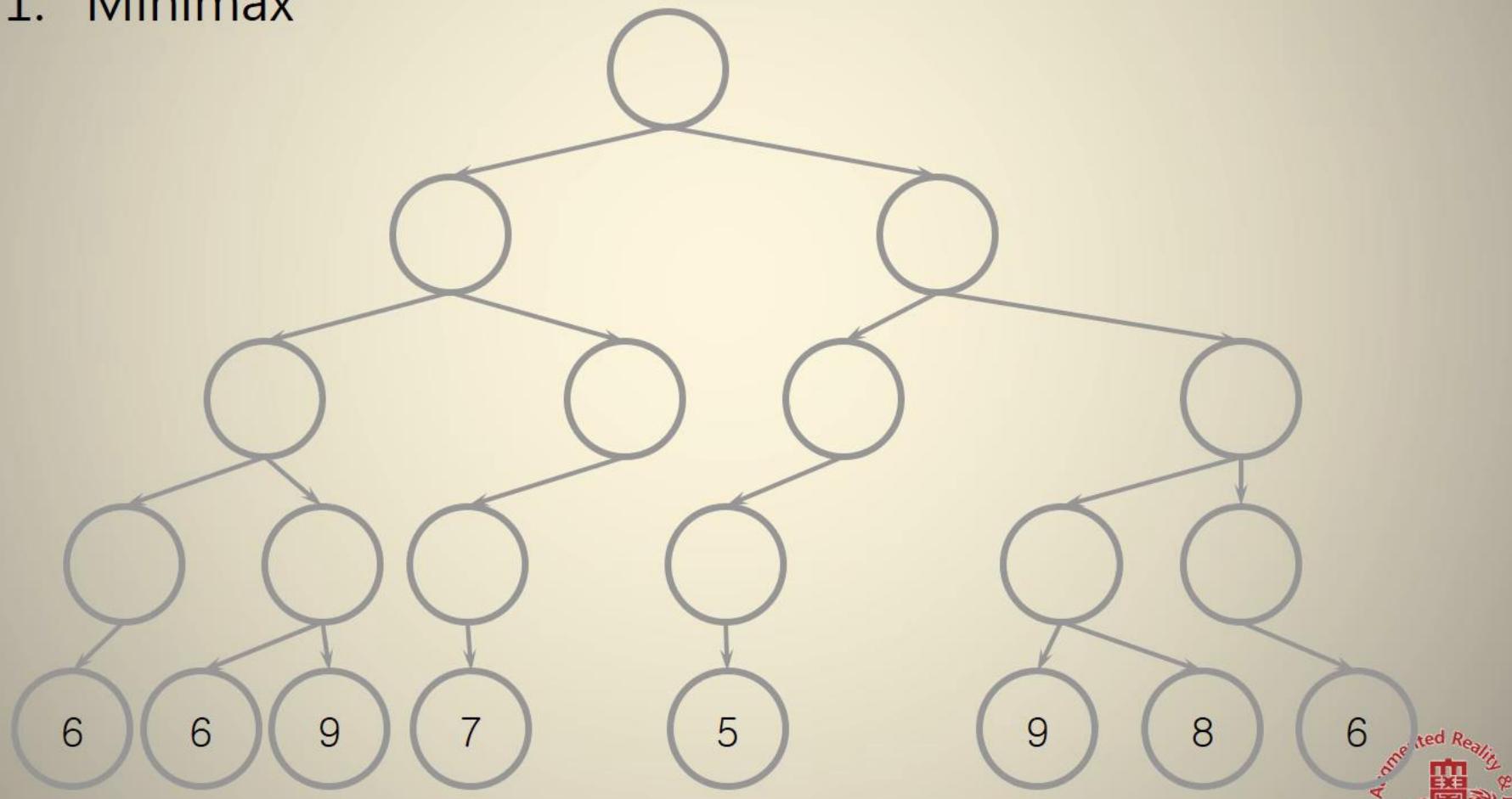
# 알파-베타 가지치기 (1) (Alpha-beta pruning)

◆ Minimax



# 알파-베타 가지치기 (2) (Alpha-beta pruning)

## 1. Minimax

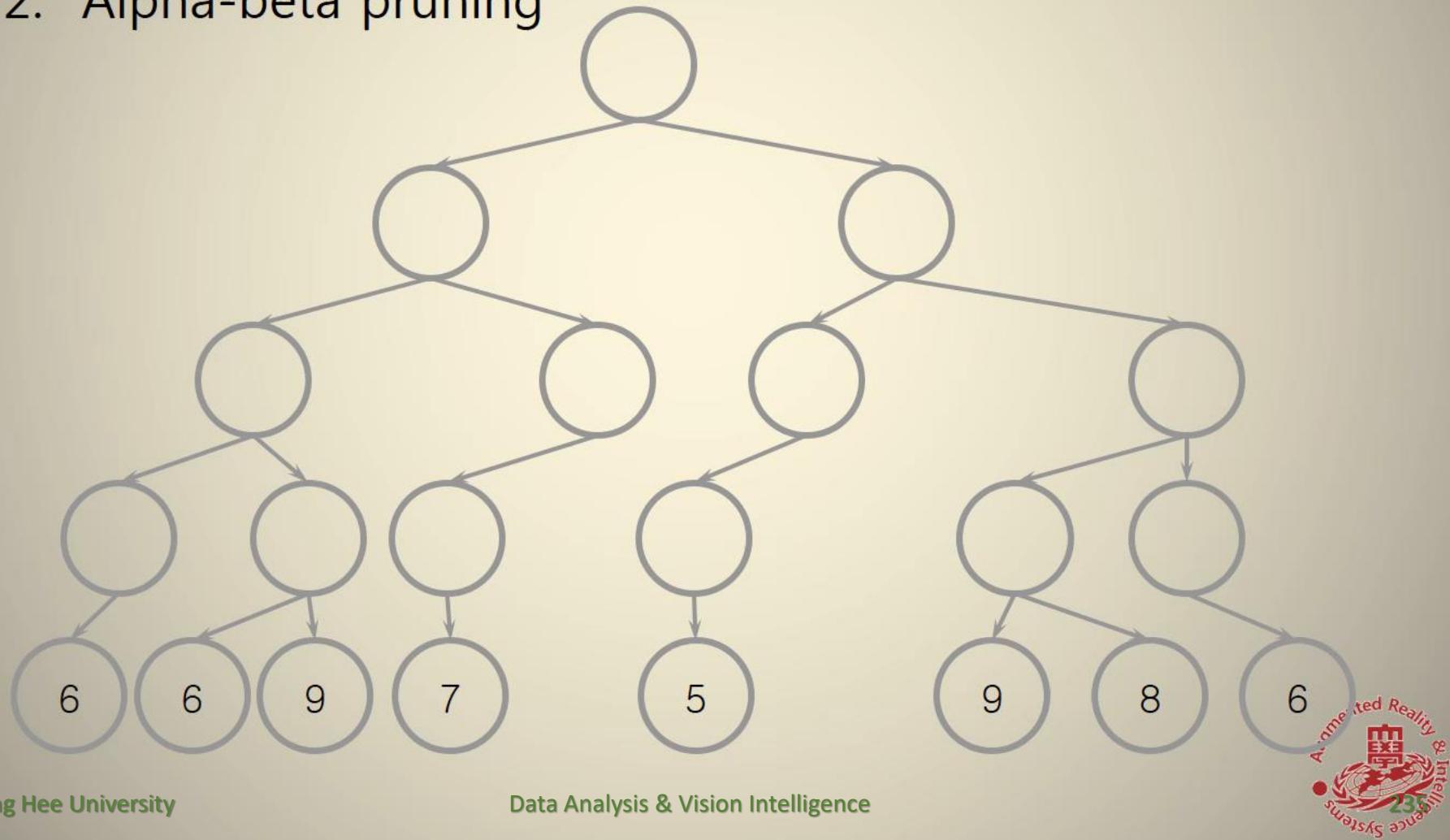


## 2. Alpha-beta pruning

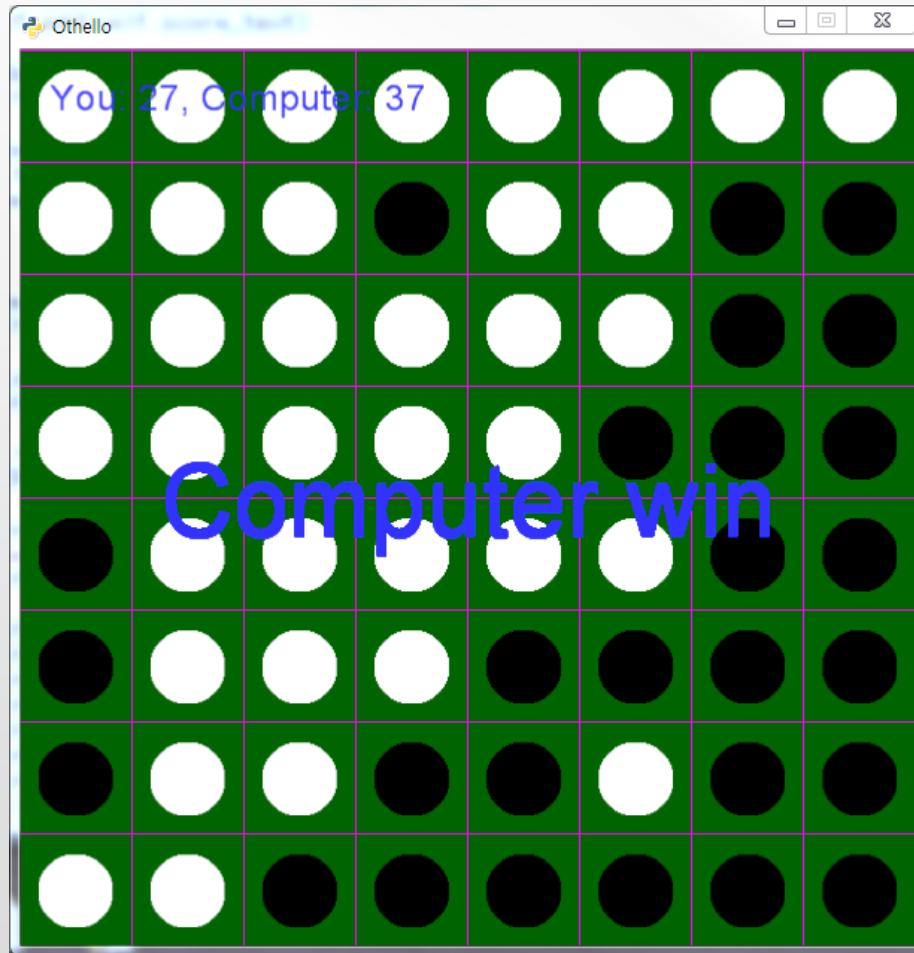
- Alpha : maximizer의 best value
- Beta: minimizer의 best value
- Initialization
  - ✓ Alpha = -infinity
  - ✓ Beta = +infinity
- Update
  - ✓ Alpha = max(alpha, best)
  - ✓ Beta = min(beta, best)
- $\text{beta} \leq \text{alpha}$ : pruning

# 알파-베타 가지치기 (4) (Alpha-beta pruning)

## 2. Alpha-beta pruning



# 실습 5 - Reversi



zip takes n number of iterables and returns list of tuples. ith element of the tuple is created using the ith element from each of the iterables.

```
xList = [1, 2, 3]
yList = [11, 12, 13, 14]
for xx, yy in zip(xList, yList):
 print(xx, yy) # 1 11 2 12 3 13

l = list(zip(xList, yList))
print(l) # [(1, 11), (2, 12), (3, 13)]
```

```
import cocos
from cocos.menu import *
import numpy as np
import cocos.euclid as eu

class HUD(cocos.layer.Layer):
 def __init__(self):
 super(HUD, self).__init__()
 w, h = cocos.director.director.get_window_size()
 self.score_text = cocos.text.Label('', font_size=18,
 color=(50, 50, 255, 255))
 self.score_text.position = (20, h - 40)
 self.add(self.score_text)
```

```
def update_score(self, person, computer):
 self.score_text.element.text =\
 'You: %s, Computer: %s' % (person, computer)

def show_game_over(self, winner):
 w, h = cocos.director.director.get_window_size()
 game_over = cocos.text.Label(winner, font_size=50,
 anchor_x='center',
 anchor_y='center',
 color=(50, 50, 255, 255))
 game_over.position = w * 0.5, h * 0.5
 self.add(game_over)
```

# GameLayer (1): \_\_init\_\_

```
class GameLayer(cocos.layer.Layer):
 is_event_handler = True
 PERSON = -1 # Black
 COMPUTER = 1 # White, empty: 0
def __init__(self, difficulty, hud_layer):
 super(GameLayer, self).__init__()
 self.difficulty = difficulty
 self.levelDepth = self.difficulty*2+2
 self.hud = hud_layer
 self.square = 75
 self.row = 8#4
 self.column = 8#4
 self.height = self.row*self.square
 self.width = self.column*self.square
```

## GameLayer (2) : \_\_init\_\_

```
 self.table = \
np.arange(self.row*self.column).reshape(self.row, self.column)

 self.weight \
= np.array(\
[[50, -10, 15, 15, 15, 15, -10, 50],
[-10, -20, -10, -10, -10, -10, -20, -10],
[15, -10, 1, 1, 1, 1, -10, 15],
[15, -10, 1, 1, 1, 1, -10, 15],
[15, -10, 1, 1, 1, 1, -10, 15],
[15, -10, 1, 1, 1, 1, -10, 15],
[-10, -20, -10, -10, -10, -10, -20, -10],
[50, -10, 15, 15, 15, 15, -10, 50]])
```

## GameLayer (3) : \_\_init\_\_

```
for x in range (0, self.column+1) :
 line = cocos.draw.Line((x*self.square, 0),
 (x*self.square, self.height), (255, 0, 255, 255))
 self.add(line)
for y in range (0, self.row+1) :
 line = cocos.draw.Line((0, y*self.square),
 (self.width, y*self.square), (255, 0, 255, 255))
 self.add(line)

self.disk = [[None for i in range(self.column)] \
 for j in range(self.row)] # sprite
```

## GameLayer (4) : \_\_init\_\_

```
for y in range (0, self.row) :
 for x in range (0, self.column) :
 centerPt = eu.Vector2 (
 x*self.square + self.square/2, \
 y*self.square + self.square/2)
 self.disk[y][x] = \
 cocos.sprite.Sprite('ball.png', \
 position = centerPt, \
 color = (255, 255, 255))
 self.add(self.disk[y][x])

self.setup()
self.turn = GameLayer.PERSON
self.count = 0 # delay for person display
self.schedule(self.update)
```

# GameLayer (5) : setup

```
def setup(self):
 for y in range (0, self.row) :
 for x in range (0, self.column) :
 self.table[y] [x] = 0

 self.table[3] [3] = GameLayer.PERSON
 self.table[3] [4] = GameLayer.COMPUTER
 self.table[4] [3] = GameLayer.COMPUTER
 self.table[4] [4] = GameLayer.PERSON
```

# GameLayer (6): update

```
def update(self, dt):
 computer = 0
 person = 0
 self.count += 1
 for y in range(0, self.row) :
 for x in range(0, self.column) :
 if self.table[y][x] == GameLayer.COMPUTER:
 self.disk[y][x].color = (255, 255, 255)
 self.disk[y][x].visible = True
 computer += 1
 elif self.table[y][x] == GameLayer.PERSON:
 self.disk[y][x].color = (0, 0, 0)
 self.disk[y][x].visible = True
 person += 1
 else:
 self.disk[y][x].visible = False
```

# GameLayer (7) : update

```
self.hud.update_score(person, computer)
moves1 = self.getMoves(GameLayer.PERSON, self.table)
moves2 = self.getMoves(GameLayer.COMPUTER, self.table)
if self.turn == GameLayer.PERSON and len(moves1) == 0:
 self.turn *= -1
if computer+person == self.row*self.column or \
 (len(moves1) == 0 and len(moves2) == 0): # 모두 둘 곳 없음
 if computer > person:
 self.hud.show_game_over('Computer win')
 elif computer < person:
 self.hud.show_game_over('You win')
 else:
 self.hud.show_game_over('Draw')
if self.turn == GameLayer.COMPUTER and self.count > 100:
 self.computer()
```

# GameLayer (8): isPossible

```
def isPossible(self, x, y, turn, board): # 8방향으로 확인
 rtnList = list() # reverse되는 좌표 list 반환

 if board[y][x] != 0: return rtnList

 for dirX in range(-1, 2):
 for dirY in range(-1, 2):
 if dirX == 0 and dirY == 0: continue # 가운데
 if (x+dirX < 0 or x+dirX >= self.column): # 경계
 continue
 if (y+dirY < 0 or y+dirY >= self.row): # 경계
 continue
 xList = list()
 yList = list()
```

# GameLayer (9): isPossible

```
방향으로 진행하고 xList, yList에 좌표 저장
if dirX == 0:
 for yy in range(y+dirY*2, \
 self.row*dirY, dirY):
 if(yy < 0 or yy >= self.row): break
 xList.append(x); yList.append(yy)
elif dirY == 0:
 for xx in range(x+dirX*2, \
 self.column*dirX, dirX):
 if(xx < 0 or xx >= self.column): break
 xList.append(xx); yList.append(y)
else:
 for xx, yy in zip(range(x+dirX*2, \
 self.column*dirX, dirX), range(y+dirY*2, self.row*dirY, dirY)):
 if(xx < 0 or xx >= self.column): break
 if(yy < 0 or yy >= self.row): break
 xList.append(xx); yList.append(yy)
```

# GameLayer (10): isPossible

```
bDetected = False
revList = []
if board[y+dirY] [x+dirX] == turn*-1: # 둘 수 있는 곳
 revList.append((x+dirX, y+dirY)) # reverse될 좌표
 for xx, yy in zip(xList, yList):
 if xx >= self.column or xx < 0 or \
 yy >= self.row or yy < 0:
 break
 if board[yy] [xx] == turn*-1:
 revList.append((xx, yy))
 if board[yy] [xx] == turn:
 bDetected = True; break
 if board[yy] [xx] == 0:
 break
 if(bDetected == False): revList = []
rtnList += revList;
return rtnList
```

# GameLayer (11): getMoves

```
def getMoves(self, turn, board): # 둘 수 있는 곳 찾음
 moves = []
 for y in range(0, self.row):
 for x in range(0, self.column):
 if board[y][x] != 0: continue

 revList = self.isPossible(x, y, turn, board)
 if len(revList) > 0:
 moves.append((x, y, revList))

 return moves
[(4, 2, [(4, 3)]), (5, 3, [(4, 3)]), (2, 4, [(3, 4)])]
[(x, y, [(x, y)]), ...]
[(position, reverse position list), ...]
```

## GameLayer (12): on\_mouse\_release

```
def on_mouse_release(self, x, y, buttons, mod):
 if self.turn != GameLayer.PERSON:
 return

 moves = self.getMoves(GameLayer.PERSON, self.table)

 if len(moves) > 0:
 xx = x//self.square
 yy = y//self.square

 revList = self.isPossible(xx, yy, \
 GameLayer.PERSON, self.table)
```

# GameLayer (13): on\_mouse\_release

```
if len(revList) == 0: return

 self.table[yy][xx] = GameLayer.PERSON
 for revX, revY in revList:
 self.table[revY][revX] = GameLayer.PERSON

self.turn *= -1
self.count = 0
```

# GameLayer (14): computer

```
def computer(self):
 move = self.minimax(GameLayer.COMPUTER)

 if len(move) > 0:
 self.table[move[1]][move[0]] = \
 GameLayer.COMPUTER

 for revX, revY in move[2]:
 self.table[revY][revX] = \
 GameLayer.COMPUTER

 self.turn *= -1
```

# GameLayer (15): minimax

```
def minimax(self, player):
 moves = self.getMoves(player, self.table) # 둘 수 있는 곳
 if len(moves) == 0: return moves
 scores = np.zeros(len(moves))
 alpha = float("-inf")
 beta = float("inf")

 for i, move in enumerate(moves):
 boardCopy = self.getNewBoard(move[0], move[1],
 move[2], GameLayer.COMPUTER, np.copy(self.table))
 if 1 >=self.levelDepth:
 scores[i] = self.boardScore(boardCopy)
 else:
 scores[i] = self.minMove(boardCopy, 2, alpha, beta)
 # maxMove(board, depth, alpha, beta)
maxIndex = np.argmax(scores)
return moves[maxIndex]
```

# GameLayer (16): getNewBoard

```
def getNewBoard(self, x, y, revList, player, table):
 # 한 수를 진행한 새로운 보드
 table[y][x] = player

 for (x,y) in revList:
 table[y][x] = player

 return table
```

# GameLayer (17): maxMove

```
def maxMove(self, board, depth, alpha, beta):
 moves = self.getMoves(GameLayer.COMPUTER, board)
 scores = np.zeros(len(moves))

 if len(moves)==0: # 둘 곳 없음
 if depth<=self.levelDepth: # 다음 턴으로(탐색 level 내)
 return self.minMove(board, depth+1, \
 alpha, beta)
 else:
 return self.boardScore(board)
```

# GameLayer (18): maxMove

```
for i, move in enumerate(moves):
 boardCopy = self.getNewBoard(move[0], move[1], \
 move[2], GameLayer.COMPUTER, np.copy(board))
 if depth>=self.levelDepth:
 scores[i] = self.boardScore(boardCopy)
 else:
 scores[i] = self.minMove(boardCopy, depth+1, \
 alpha, beta)
 if scores[i] > alpha:
 alpha = scores[i]
 if beta <= alpha:
 return scores[i]
return max(scores)
```

# GameLayer (19): minMove

```
def minMove(self, board, depth, alpha, beta):
 moves = self.getMoves(GameLayer.PERSON, board)
 scores = np.zeros(len(moves))

 if len(moves)==0:
 if depth<=self.levelDepth:
 return self.maxMove(board, depth+1, \
 alpha, beta)
 else:
 return self.boardScore(board)
```

# GameLayer (20): minMove

```
for i, move in enumerate(moves):
 boardCopy = self.getNewBoard(move[0], move[1], \
 move[2], GameLayer.PERSON, np.copy(board))
 if depth>=self.levelDepth:
 scores[i] = self.boardScore(boardCopy)
 else:
 scores[i] = self.maxMove(boardCopy, depth+1, \
 alpha, beta)
 if beta > scores[i]:
 beta = scores[i]
 if beta <= alpha:
 return scores[i]
return min(scores)
```

# GameLayer (21): boardScore

```
def boardScore(self, board):
 computerWeightSum = 0
 personWeightSum = 0

 for y in range(0, self.row):
 for x in range(0, self.column):
 if board[y][x] == GameLayer.COMPUTER:
 computerWeightSum += self.weight[y][x]

 if board[y][x] == GameLayer.PERSON:
 personWeightSum += self.weight[y][x]

 return computerWeightSum - personWeightSum;
```

```
class MainMenu(Menu):
 def __init__(self):
 super(MainMenu, self).__init__('Menu')
 self.font_title['font_name'] = 'Times New Roman'
 self.font_title['font_size'] = 60
 self.font_title['bold'] = True
 self.font_item['font_name'] = 'Times New Roman'
 self.font_item_selected['font_name'] = 'Times New Roman'

 self.selDifficulty = 0
 self.difficulty = ['Easy', 'Normal', 'Hard']
```

```
items = list()
items.append(MenuItem('New Game', self.start_game))
items.append(MultiplayerMenuItem('Difficulty: ', \
 self.set_difficulty, self.difficulty, 0))
items.append(MenuItem('Quit', exit))
self.create_menu(items, shake(), shake_back())
def start_game(self):
 scene = cocos.scene.Scene()
 color_layer = cocos.layer.ColorLayer(0, 100, 0, 255)
 hud_layer = HUD()
 scene.add(hud_layer, z=2)
 scene.add(GameLayer(self.selDifficulty, hud_layer), z=1)
 scene.add(color_layer, z=0)
 cocos.director.director.push(scene)
def set_difficulty(self, index):
 self.selDifficulty = index
```

```
if __name__ == '__main__':
 cocos.director.director.init(caption='Othello', \
 width = 75*8, height = 75*8)

 scene = cocos.scene.Scene()
 scene.add(MainMenu())

cocos.director.director.run(scene)
```

# 4. Steering Behaviors

Kyung Hee University  
Daeho Lee

## ◆ cocos.particle\_systems

- class ParticleSystem(fallback=None, texture=None) → base
- class Fireworks(fallback=None, texture=None)
- class Spiral(fallback=None, texture=None)
- class Meteor(fallback=None, texture=None)
- class Sun(fallback=None, texture=None)
- class Galaxy(fallback=None, texture=None)
- class Flower(fallback=None, texture=None)
- class Explosion(fallback=None, texture=None)
- class Smoke(fallback=None, texture=None)

# Particle System class (2)

```
import cocos
import cocos.particle_systems as ps

class MainLayer(cocos.layer.Layer):
 def __init__(self):
 super(MainLayer, self).__init__()
 particles = ps.Fireworks(fallback = False)
 #particles.angle = 180
 #particles.size = 3
 particles.position = (320, 240)
 self.add(particles)
```

fallback: Defaults to None.  
False: use point sprites, faster, not always available  
True: use quads, slower but always available  
None: auto-detect, use the fastest available

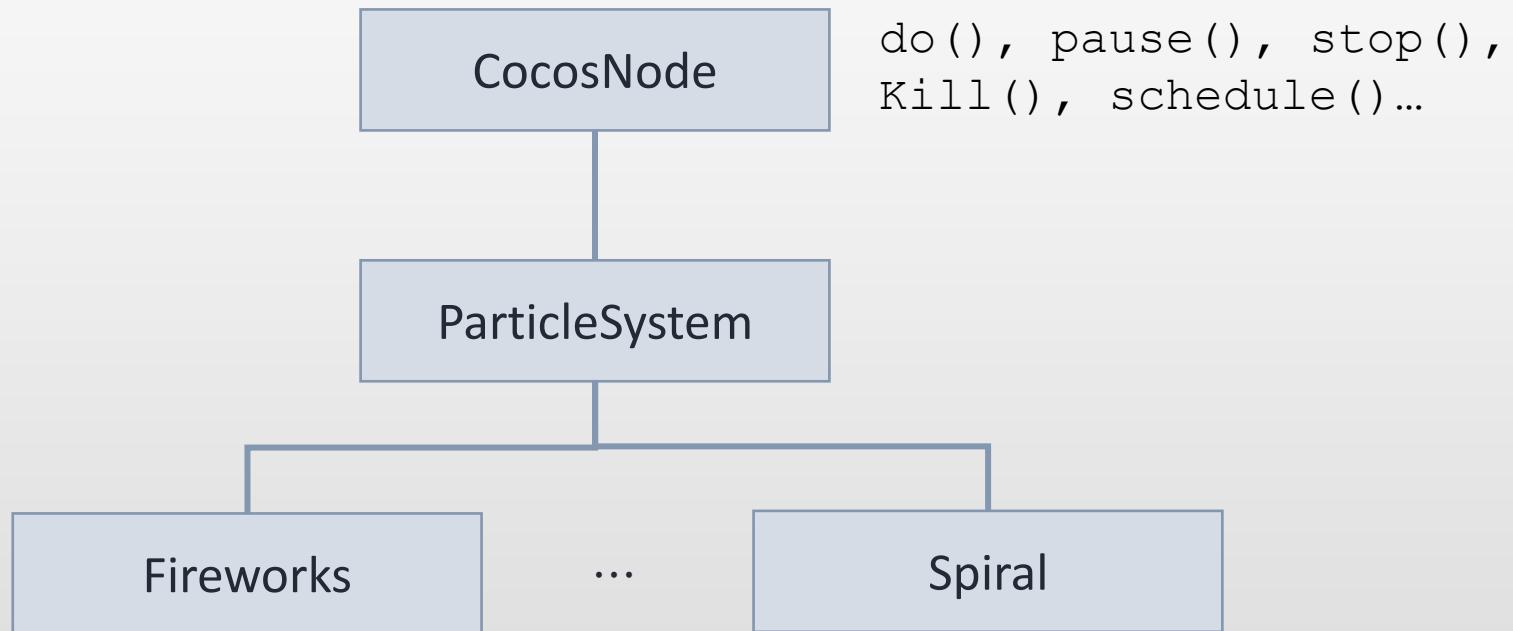
# Particle System class (3)

```
if __name__ == '__main__':
 cocos.director.director.init(caption='Particle example')
 scene = cocos.scene.Scene(MainLayer())
 cocos.director.director.run(scene)
```

# Particle System class (4)

|                                           |                                                 |
|-------------------------------------------|-------------------------------------------------|
| active                                    | True/False                                      |
| duration                                  | Duration, in seconds, -1: infinite duration     |
| gravity                                   | Gravity, Point2(0.00, 0.00)                     |
| angle, angle_var                          | Angular direction (in degrees), variance (_var) |
| speed, speed_var                          | Speed                                           |
| tangential_accel,<br>tangential_accel_val | Tangential acceleration                         |
| radial_accel, radial_accel_val            | Radial acceleration                             |
| size, size_val                            | Size                                            |
| life, life_val                            | Life time in seconds                            |
| start_color, start_color_val              | Start color                                     |
| end_color, end_color_val                  | End color                                       |
| totoal_particles                          | Maximum number of particles                     |

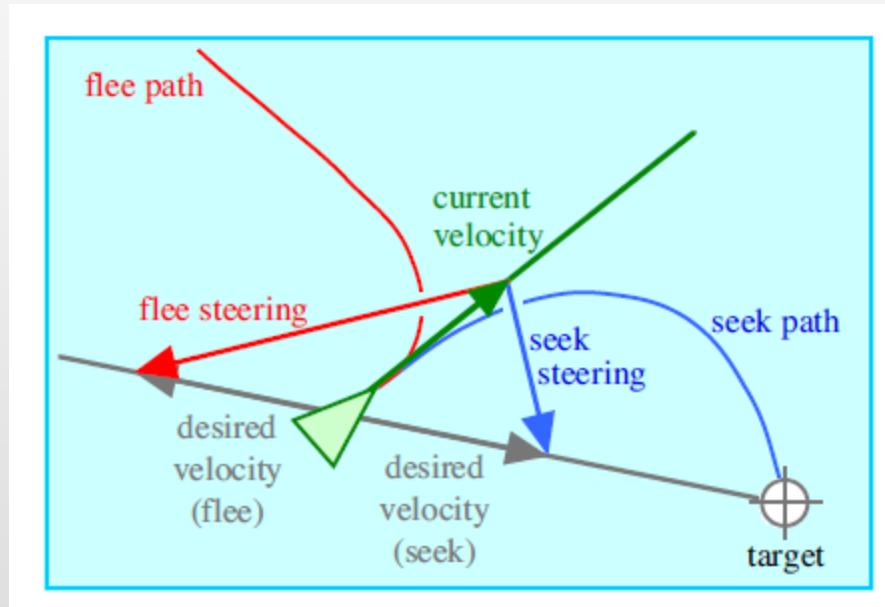
# Particle System class (5)



# Implementing steering behaviors

- ◆ C.W. Reynolds, “Steering behaviors for autonomous characters,” in Proc. Game Developer’s Conference, 1999.<https://www.red3d.com/cwr/papers/1999/gdc99steer.pdf>
- ◆ Seek and flee : pursuit/evade of static target
- ◆ Arrival
- ◆ Pursuit and evade: seek/flee for another moving character
- ◆ Wander
- ◆ Obstacle avoidance

## ◆ Seek (pursuit of static target) <-> Flee



C.W. Reynolds, “Steering behaviors for autonomous characters,” in Proc. Game Developer’s Conference, 1999.

```
import cocos
import cocos.euclid as eu
import cocos.particle_systems as ps

def truncate(vector, m): # limits vector
 magnitude = abs(vector)
 if magnitude > m:
 vector *= m / magnitude # 크기를 m으로
 return vector
```

```
class Actor(cocos.cocosnode.CocosNode):
 def __init__(self, x, y):
 super(Actor, self).__init__()
 self.position = (x, y)
 self.velocity = eu.Vector2(0, 0)
 self.speed = 2 # particle speed
 self.max_force = 5
 self.max_velocity = 200
 self.target = None
 self.add(ps.Sun())
 self.schedule(self.update)
```

```
def update(self, dt):
 if self.target is None:
 return

 distance = self.target - eu.Vector2(self.x, self.y)
 steering = distance * self.speed - self.velocity
 steering = truncate(steering, self.max_force)
 self.velocity = truncate(self.velocity + steering,
 self.max_velocity)

 self.position += self.velocity * dt
```



```
class MainLayer(cocos.layer.Layer):
 is_event_handler = True
 def __init__(self):
 super(MainLayer, self).__init__()
 self.actor = Actor(320, 240)
 self.add(self.actor)

 def on_mouse_motion(self, x, y, dx, dy):
 self.actor.target = eu.Vector2(x, y)

if __name__ == '__main__':
 cocos.director.director.init(caption='Steering Behaviors')
 scene = cocos.scene.Scene(MainLayer())
 cocos.director.director.run(scene)
```

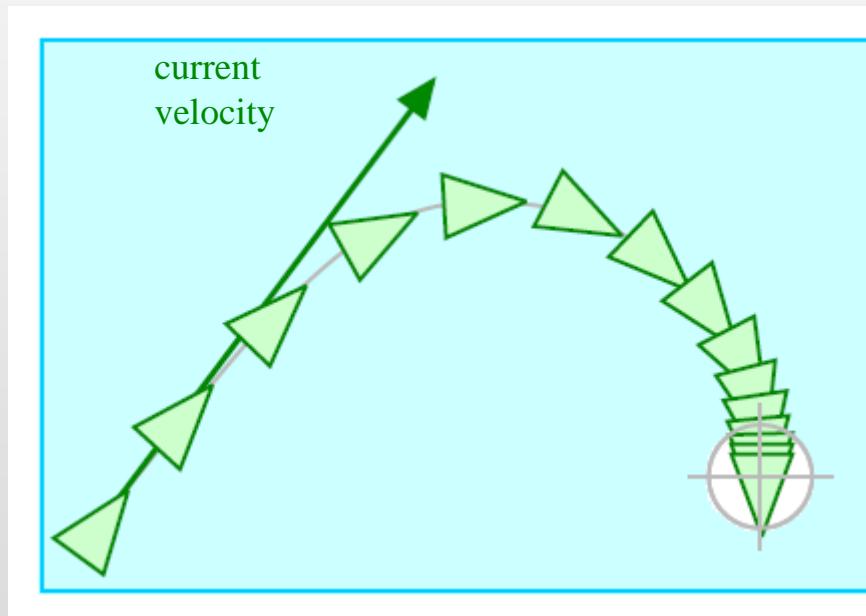
## # For Flee

```
def update(self, dt):
 if self.target is None:
 return

 distance = self.target - eu.Vector2(self.x, self.y)
 steering = distance * self.speed - self.velocity
 steering = truncate(steering, self.max_force)
 self.velocity = truncate(self.velocity + steering,
 self.max_velocity)

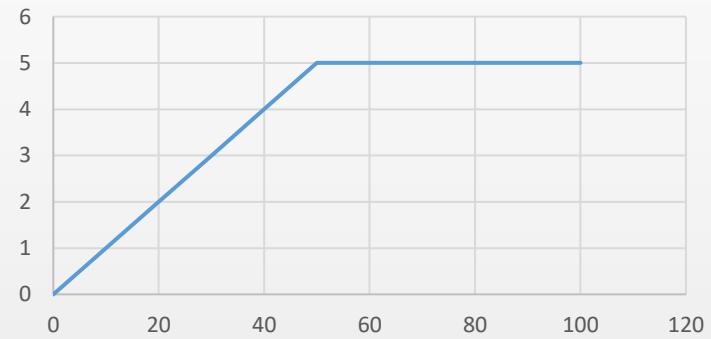
direction = 1 if self.seek else -1 # seek and flee
self.position += self.velocity * dt * direction
```

- ◆ Arrival (seek while the character is far from its target/slow down as it approaches the target)

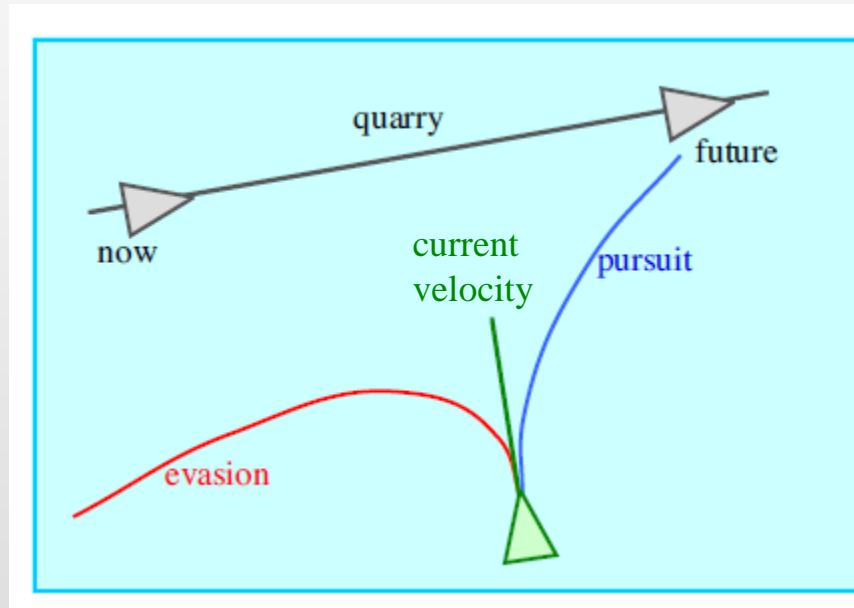


C.W. Reynolds, “Steering behaviors for autonomous characters,” in Proc. Game Developer’s Conference, 1999.

```
class Actor(cocos.cocosnode.CocosNode):
 def __init__(self, x, y):
 ...
 self.slow_radius = 200
 ...
 def update(self, dt):
 ...
 distance = self.target - eu.Vector2(self.x, self.y)
 ramp = min(abs(distance) / self.slow_radius, 1.0)
 steering = distance * self.speed * ramp - \
 self.velocity
 steering = truncate(steering, self.max_force)
 self.velocity = truncate(self.velocity + steering,
 self.max_velocity)
 self.position += self.velocity * dt
```



## ◆ Pursuit (seek for another moving character) <-> evasion



C.W. Reynolds, “Steering behaviors for autonomous characters,” in Proc. Game Developer’s Conference, 1999.

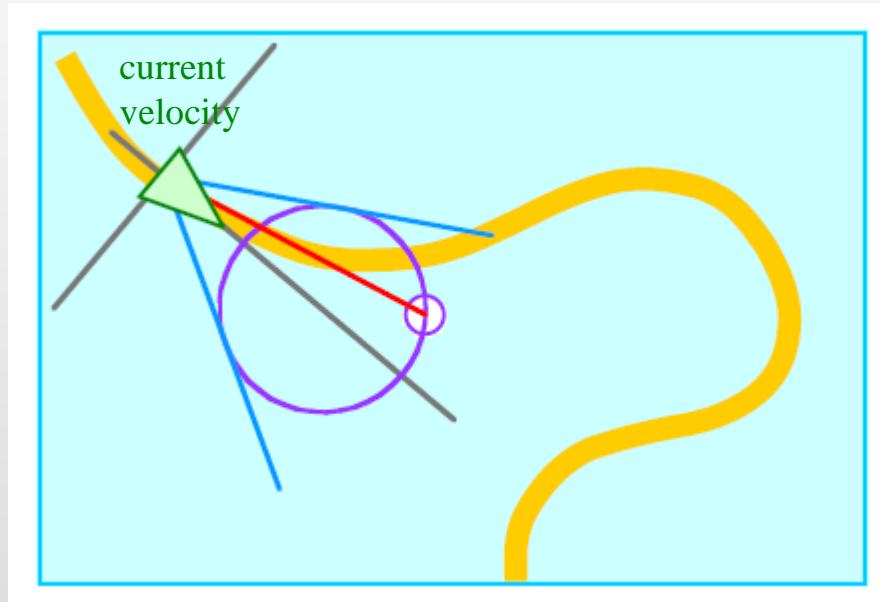
# Pursuit and evade (2)

```
class MainLayer(cocos.layer.Layer):
 def __init__(self):
 super(MainLayer, self).__init__()
 self.target = ps.Sun()
 self.target.position = (40, 40)
 self.target.start_color = ps.Color(0.2, 0.7, 0.7, 1.0)
 #self.target.velocity = eu.Vector2(50, 0)
 self.target.velocity = eu.Vector2(150, 0)
 self.add(self.target)
 self.actor = Actor(320, 240)
 self.actor.target = self.target
 self.add(self.actor)
 self.schedule(self.update)
 def update(self, dt):
 self.target.position += self.target.velocity * dt
```

# Pursuit and evade (3)

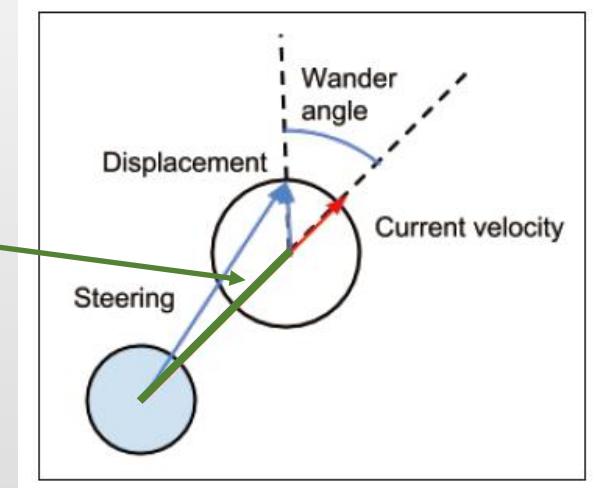
```
class Actor(cocos.cocosnode.CocosNode) :
 ...
 def update(self, dt) :
 ...
 pos = self.target.position
 #future_pos = pos + self.target.velocity * 1
 future_pos = pos + self.target.velocity * dt
 distance = future_pos - eu.Vector2(self.x, self.y)
 steering = distance * self.speed - self.velocity
 steering = truncate(steering, self.max_force)
 self.velocity = truncate(self.velocity + steering,
 self.max_velocity)
 self.position += self.velocity * dt
```

## ◆ Wander (a type of random steering)



C.W. Reynolds, “Steering behaviors for autonomous characters,” in Proc. Game Developer’s Conference, 1999.

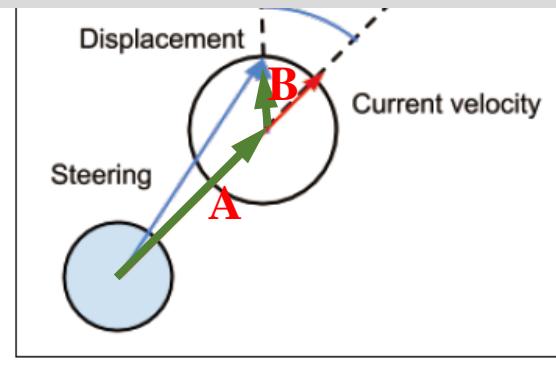
```
class Actor(cocos.cocosnode.CocosNode):
 def __init__(self, x, y):
 ...
 self.position = (x, y)
 self.velocity = eu.Vector2(0, 0)
 self.wander_angle = 0
 self.circle_distance = 50
 self.circle_radius = 10
 self.angle_change = math.pi / 4
 self.max_velocity = 50
 self.add(ps.Sun())
 self.schedule(self.update)
```



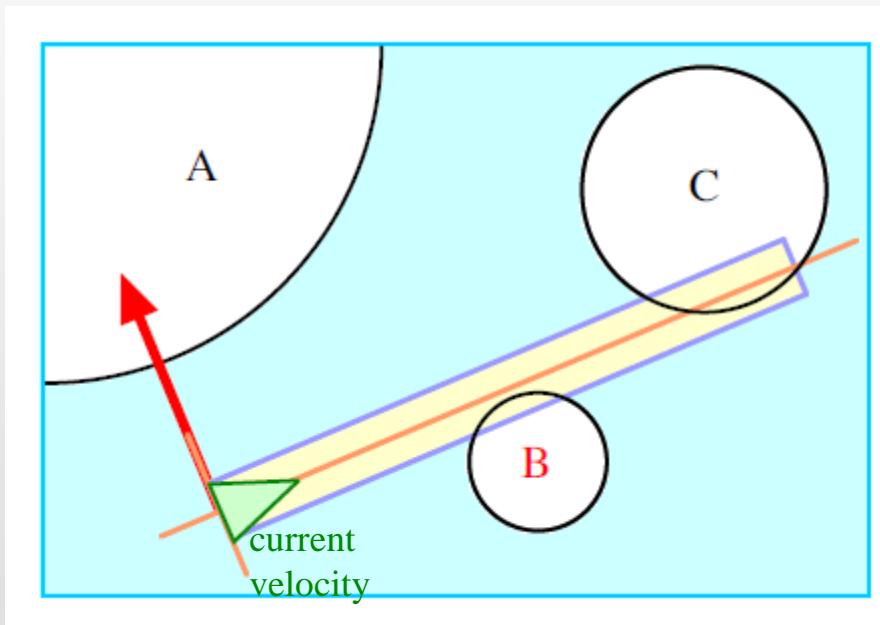
```

def update(self, dt):
 circle_center = self.velocity.normalized() * \
 self.circle_distance # A, placing circle
 dx = math.cos(self.wander_angle)
 dy = math.sin(self.wander_angle)
 displacement = eu.Vector2(dx, dy) * \
 self.circle_radius # B
 self.wander_angle += (random.random() - 0.5) * \
 self.angle_change
 self.velocity += circle_center + displacement # A + B
 self.velocity = truncate(self.velocity,
 self.max_velocity)
 self.position += self.velocity * dt
 self.position = (self.x % 640, self.y % 480)

```



# Obstacle avoidance (1)



C.W. Reynolds, “Steering behaviors for autonomous characters,” in Proc. Game Developer’s Conference, 1999.

# Obstacle avoidance (2)

```
class Obstacle(cocos.cocosnode.CocosNode):
 instances = []

 def __init__(self, x, y, r):
 super(Obstacle, self).__init__()
 self.position = (x, y)
 self.radius = r
 particles = ps.Sun()
 particles.size = r * 2
 particles.start_color = ps.Color(0.0, 0.7, 0.0, 1.0)
 self.add(particles)
 self.instances.append(self)
```

# Obstacle avoidance (3)

```
class Actor(cocos.cocosnode.CocosNode):
 def __init__(self, x, y):
 super(Actor, self).__init__()
 self.position = (x, y)
 self.velocity = eu.Vector2(0, 0)
 self.speed = 2
 self.max_velocity = 300
 self.max_force = 10
 self.target = None
 self.max_ahead = 200
 self.max_avoid_force = 300

 ...
```

# Obstacle avoidance (4)

```
def update(self, dt):
 if self.target is None:
 return

 distance = self.target - eu.Vector2(self.x, self.y)
 steering = distance * self.speed - self.velocity # seek
 steering += self.avoid_force()
 steering = truncate(steering, self.max_force)
 self.velocity = truncate(self.velocity + steering,
 self.max_velocity)
 self.position += self.velocity * dt
```

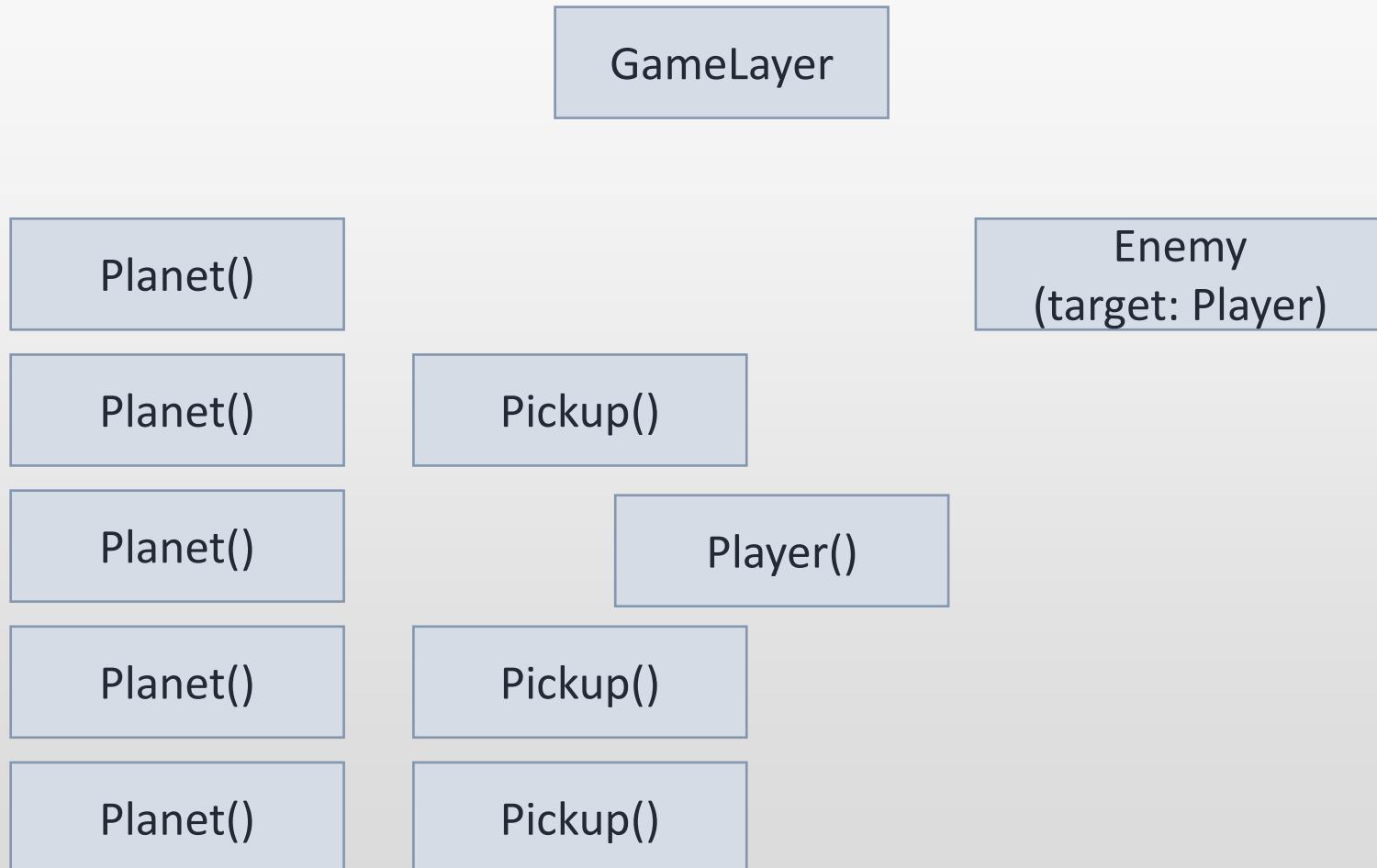
# Obstacle avoidance (5)

```
def avoid_force(self):
 avoid = eu.Vector2(0, 0)
 ahead = self.velocity * self.max_ahead /
 self.max_velocity
 l = ahead.dot(ahead)
 if l == 0: # ahead is zero
 return avoid
```

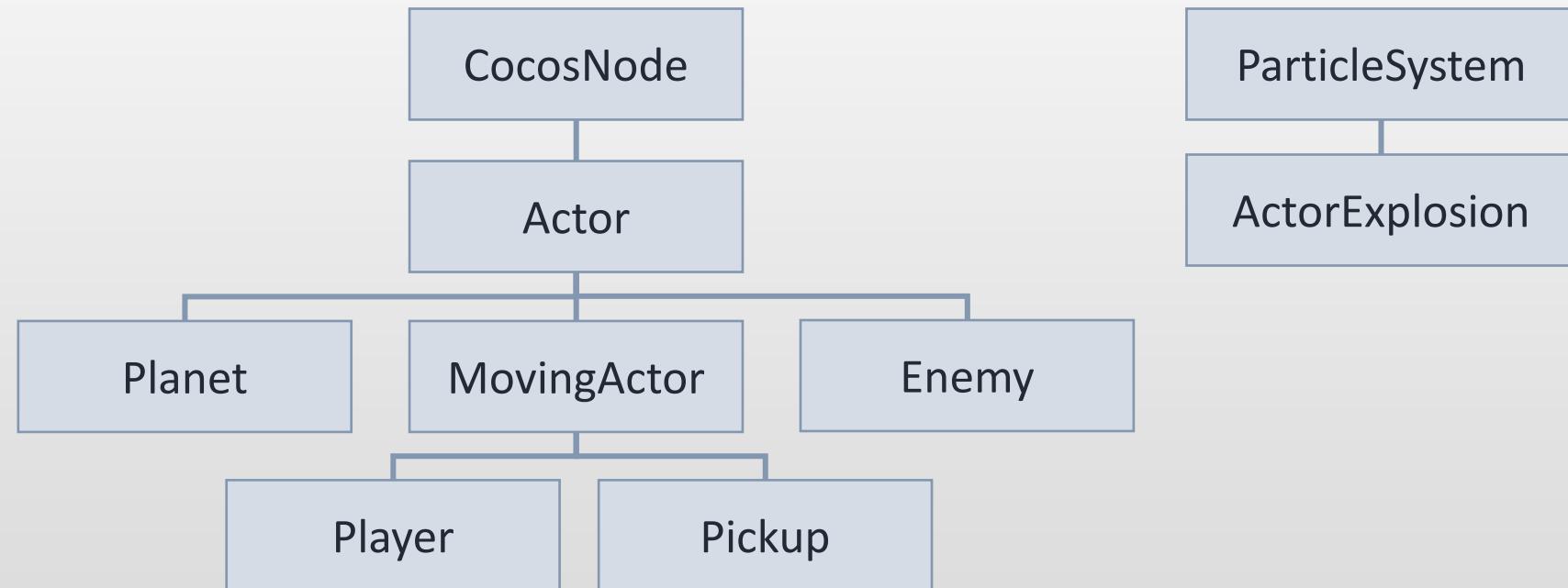
# Obstacle avoidance (6)

```
closest, closest_dist = None, None
for obj in Obstacle.instances:
 w = eu.Vector2(obj.x - self.x, obj.y - self.y)
 t = ahead.dot(w)
 if 0 < t < 1: # 0 < t < L(ahead.dot(ahead))
 proj = self.position + ahead * t / 1 # (L)
 dist = abs(obj.position - proj)
 if dist < obj.radius and \
 (closest is None or dist < closest_dist):
 closest, closest_dist = obj.position, dist
if closest is not None:
 avoid = self.position + ahead - closest
 avoid = avoid.normalized() * self.max_avoid_force
return avoid
```

# Gravitation game (1)



# Gravitation game (2)



MovingActor: Planet을 공전

Player: 공전하는 Planet이 없는 경우도 있음  
Planet없는 경우의 동작은 Player 클래스에서 정의

```
import math

from cocos.cocosnode import CocosNode
from cocos.director import director

import cocos.layer
import cocos.scene
import cocos.euclid as eu
import cocos.collision_model as cm
import cocos.particle_systems as ps

from pyglet.window import key
```

# Gravitation game (4)

```
class ActorExplosion(ps.ParticleSystem):
 total_particles = 400
 duration = 0.1
 gravity = eu.Point2(0, 0)
 angle = 90.0; angle_var = 360.0
 speed = 40.0; speed_var = 20.0
 life = 3.0; life_var = 1.5
 emission_rate = total_particles / duration
 start_color_var = ps.Color(0.0, 0.0, 0.0, 0.2)
 end_color = ps.Color(0.0, 0.0, 0.0, 1.0)
 end_color_var = ps.Color(0.0, 0.0, 0.0, 0.0)
 size = 15.0
 size_var = 10.0
 blend_additive = True
```

# Gravitation game (5)

```
def __init__(self, pos, color):
 super(ActorExplosion, self).__init__()
 self.position = pos
 self.start_color = color

def truncate(vector, m):
 magnitude = abs(vector)
 if magnitude > m:
 vector *= m / magnitude
 return vector
```

```
class Actor(CocosNode):
 def __init__(self, x, y, r):
 super(Actor, self).__init__()
 self.position = (x, y)
 self._cshape = cm.CircleShape(self.position, r)

 @property
 def cshape(self):
 self._cshape.center = eu.Vector2(self.x, self.y)
 return self._cshape
```

```
class MovingActor(Actor):
 def __init__(self, x, y, r):
 super(MovingActor, self).__init__(x, y, r)
 self._planet = None
 self._distance = 0
 self.angle = 0
 self.rotationSpeed = 0.6
 self.schedule(self.update)
```

```
@property
def planet(self):
 return self._planet
```

# Gravitation game (8)

```
@planet.setter
def planet(self, val):
 if val is not None:
 dx, dy = self.x - val.x, self.y - val.y
 self.angle = -math.atan2(dy, dx)
 self._distance = abs(eu.Vector2(dx, dy))
 self._planet = val
def update(self, dt):
 if self.planet is None:
 return
 dist = self._distance
 self.angle = (self.angle + self.rotationSpeed * dt) % \
 (math.pi * 2)
 self.x = self.planet.x + dist * math.cos(self.angle)
 self.y = self.planet.y - dist * math.sin(self.angle) # CW, CCW(+)
```

```
class Planet(Actor):
 instances = []

 def __init__(self, x, y, r=50):
 super(Planet, self).__init__(x, y, r)
 particles = ps.Sun()
 particles.start_color = ps.Color(0.5, 0.5, 0.5, 1.0)
 particles.size = r * 2
 self.add(particles)
 self.instances.append(self)
```

# Gravitation game (10)

```
class Player(MovingActor):
 def __init__(self, x, y, planet):
 super(Player, self).__init__(x, y, 16)
 self.planet = planet
 self.rotationSpeed = 1
 self.linearSpeed = 80
 self.direction = eu.Vector2(0, 0)
 self.particles = ps.Meteor()
 self.particles.size = 50
 self.add(self.particles)
```

# Gravitation game (11)

```
def update(self, dt):
 if self.planet is not None:
 super(Player, self).update(dt)
 gx = 20 * math.cos(self.angle)
 gy = 20 * math.sin(self.angle)
 self.particles.gravity = eu.Point2(gx, -gy)
 else:
 self.position += self.direction * dt
def switch(self): # planet이 존재하다가 없어진 설정
 new_dir = eu.Vector2(self.y - self.planet.y,
 self.planet.x - self.x)
 self.direction = new_dir.normalized() * self.linearSpeed
 self.planet = None
 self.particles.gravity = eu.Point2(-self.direction.x,
 -self.direction.y)
```

# Gravitation game (12)

```
class PickupParticles(ps.Sun):
 size = 20
 start_color = ps.Color(0.7, 0.7, 0.2, 1.0)

class Pickup(MovingActor):
 def __init__(self, x, y, planet):
 super(Pickup, self).__init__(x, y, 10)
 self.planet = planet
 self.gravity_factor = 50
 self.particles = PickupParticles()
 self.add(self.particles)
```

# Gravitation game (13)

```
class Enemy(Actor):
 def __init__(self, x, y, target):
 super(Enemy, self).__init__(x, y, 40)
 self.velocity = eu.Vector2(0, 0)
 self.speed = 2
 self.max_force = 5
 self.max_velocity = 30
 self.max_ahead = 300
 self.max_avoid_force = 500
 self.target = target
 self.add(ps.Sun())
 self.schedule(self.update)
```

# Gravitation game (14)

```
def update(self, dt):
 if self.target is None:
 return

 distance = self.target.position - \
 eu.Vector2(self.x, self.y)
 steering = distance * self.speed - self.velocity
 steering += self.avoid_force()
 steering = truncate(steering, self.max_force)
 self.velocity = truncate(self.velocity + steering,
 self.max_velocity)
 self.position += self.velocity * dt
```

# Gravitation game (15)

```
def avoid_force(self):
 ...

class GameLayer(cocos.layer.Layer):
 def __init__(self):
 super(GameLayer, self).__init__()
 x, y = director.get_window_size()
 cell_size = 32
 self.coll_man = cm.CollisionManagerGrid(0, x, 0, y,
 cell_size, cell_size)
 self.planet_area = 400
```

# Gravitation game (16)

```
planet1 = self.add_planet(450, 280)
planet2 = self.add_planet(180, 200)
planet3 = self.add_planet(270, 440)
planet4 = self.add_planet(650, 480)
planet5 = self.add_planet(700, 150)
self.add_pickup(250, 250, planet2)
self.add_pickup(740, 480, planet4)
self.add_pickup(700, 60, planet5)
self.player = Player(300, 350, planet3)
self.add(self.player)
self.add(Enemy(600, 100, self.player))
self.schedule(self.game_loop)
```

# Gravitation game (17)

```
def add_pickup(self, x, y, target):
 pickup = Pickup(x, y, target)
 self.add(pickup)
def add_planet(self, x, y):
 planet = Planet(x, y)
 self.add(planet)
 return planet
def game_loop(self, _):
 self.coll_man.clear()
 for node in self.get_children():
 if isinstance(node, Actor):
 self.coll_man.add(node)
 if self.player.is_running:
 self.process_player_collisions()
```

# Gravitation game (18)

```
def process_player_collisions(self):
 player = self.player
 for obj in self.coll_man.iter_colliding(player):
 if isinstance(obj, Pickup):
 self.add(ActorExplosion(obj.position,
 obj.particles.start_color))
 obj.kill()

 else:
 self.add(ActorExplosion(player.position,
 player.particles.start_color))
 player.kill()
```

# Gravitation game (19)

```
is_event_handler = True

def on_key_press(self, k, _):
 if k != key.SPACE:
 return

 if self.player.planet is None:
 self.player.planet = self.find_closest_planet()
 # set planet

 else:
 self.player.switch()
 # reset planet
```

# Gravitation game (20)

```
def find_closest_planet(self):
 ranked = \
 self.coll_man.ranked_objs_near(self.player,
 self.planet_area)
 planet = next(filter(lambda x: isinstance(x[0],
 Planet), ranked))
 return planet[0] if planet is not None else None

ranked_objs_near: returns a list with the (other, distance)
pairs, and the list is ordered in increasing distance
```

# Gravitation game (21)

```
listA = [('A', -1), ('B', 1), ('C', 0)]
listB = list(filter(lambda x: x[1] != 0, listA))
C = next(filter(lambda x: x[1] != 0, listA))
print(listB) # [('A', -1), ('B', 1)]
print(C) # ('A', -1)
print(C[0]) #A
print(listA[0] if listA is not None else None) # ('A', -1)
listA = None
print(listA[0] if listA is not None else None) # None
```

# Gravitation game (22)

```
if __name__ == '__main__':
 director.init(width=850, height=600,
 caption='Gravitation')
 director.run(cocos.scene.Scene(GameLayer())))
```

# 5. Pygame and 3D

Kyung Hee University  
Daeho Lee

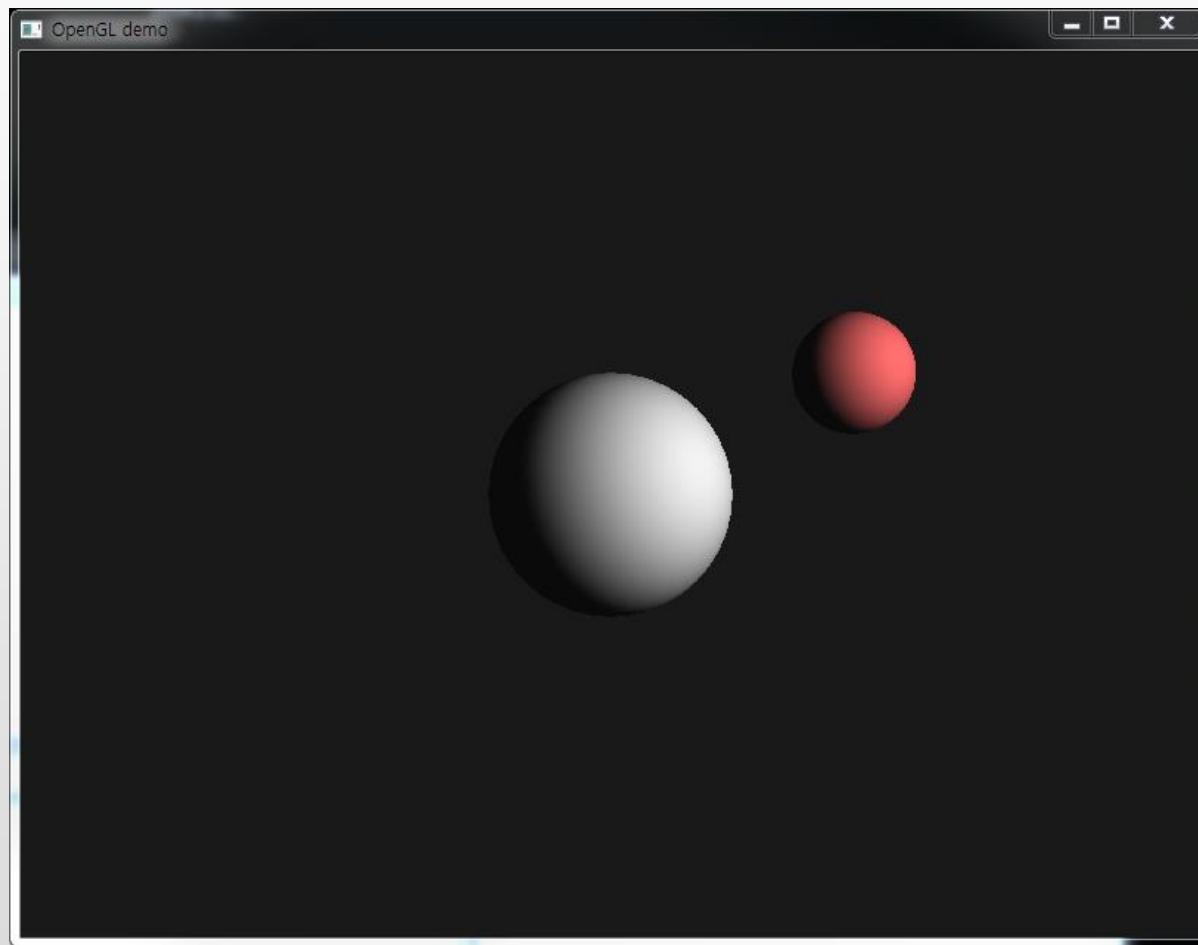
- ◆ > pip install PyOpenGL # GLUT is not included
- ◆ <https://www.lfd.uci.edu/~gohlke/pythonlibs/#pyopengl>

PyOpenGL provides bindings to OpenGL, GLUT, and GLE.

[PyOpenGL-3.1.2-cp27-cp27m-win32.whl](#)  
[PyOpenGL-3.1.2-cp27-cp27m-win\\_amd64.whl](#)  
[PyOpenGL-3.1.2-cp34-cp34m-win32.whl](#)  
[PyOpenGL-3.1.2-cp34-cp34m-win\\_amd64.whl](#)  
[PyOpenGL-3.1.2-cp35-cp35m-win32.whl](#)  
[PyOpenGL-3.1.2-cp35-cp35m-win\\_amd64.whl](#)  
[PyOpenGL-3.1.2-cp36-cp36m-win32.whl](#)  
[PyOpenGL-3.1.2-cp36-cp36m-win\\_amd64.whl](#)  
[PyOpenGL-3.1.2-cp37-cp37m-win32.whl](#)  
[PyOpenGL-3.1.2-cp37-cp37m-win\\_amd64.whl](#)

- ◆ > pip install pygame

# PyOpenGL (2)



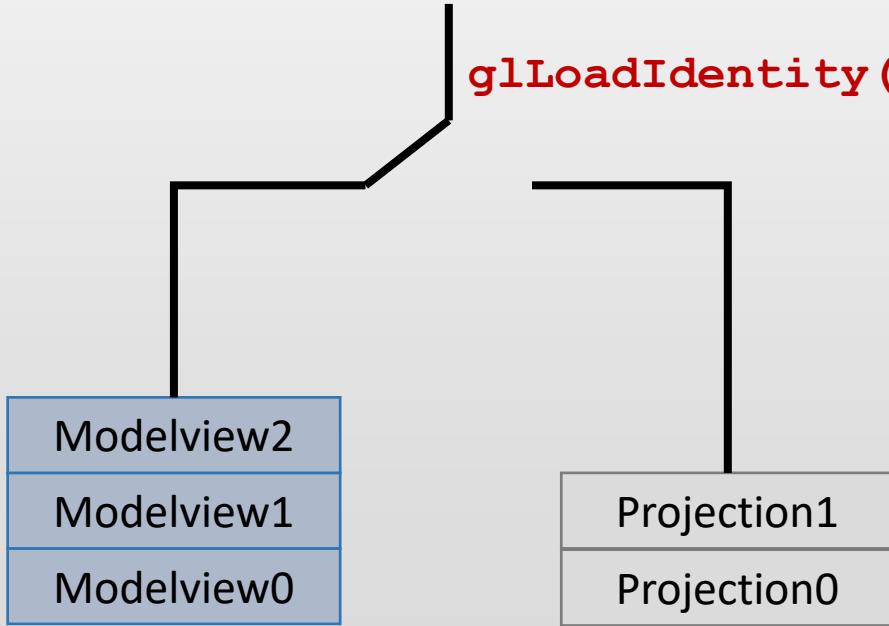
```
chapter5_01.py

import sys
import math
from OpenGL.GL import *
from OpenGL.GLU import * # OpenGL Utility Library
from OpenGL.GLUT import * # OpenGL Utility Toolkit

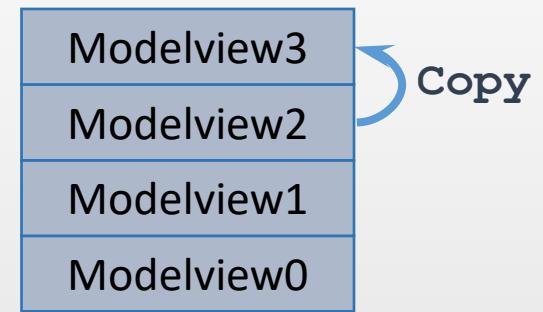
class App(object):

 def __init__(self, width=800, height=600):
 self.title = b'OpenGL demo' # binary string
 self.width = width; self.height = height
 self.angle = 0
 self.distance = 20
```

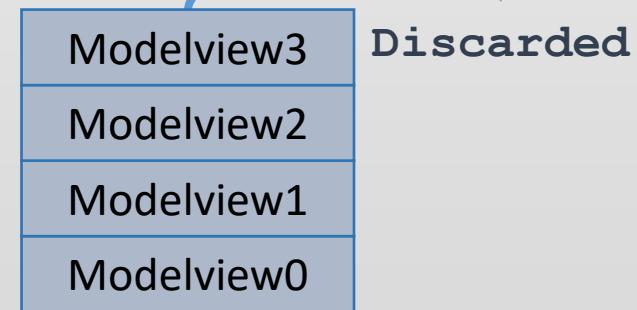
**glMatrixMode(GL\_MODELVIEW)**  
**glLoadIdentity()**, ...



**glPushMatrix**



**glPopMatrix**



```
def start(self): # initialization
 glutInit() Depth buffering ?
 glutInitDisplayMode(GLUT_DOUBLE | GLUT_DEPTH)
 glutInitWindowPosition(50, 50) Double buffering ?
 glutInitWindowSize(self.width, self.height)
 glutCreateWindow(self.title)

 glEnable(GL_DEPTH_TEST)
 glEnable(GL_LIGHTING) # lighting
 glEnable(GL_LIGHT0) # light0 (- light8)
```

```

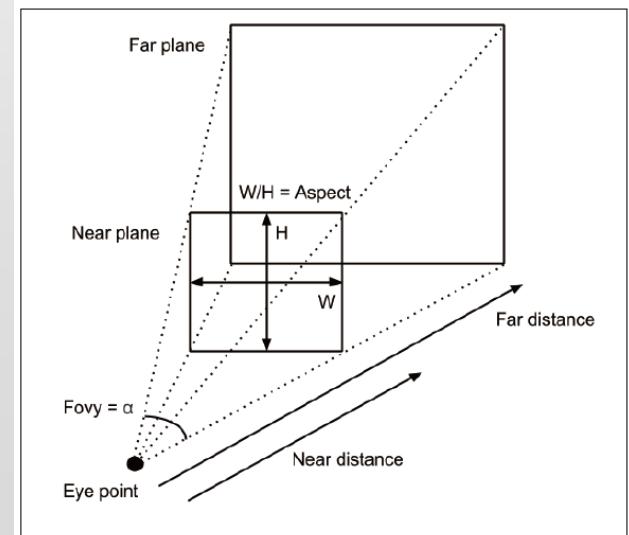
r, g, b, alpha

glClearColor(.1, .1, .1, 1) # color clear (background)
glMatrixMode(GL_PROJECTION)
aspect = self.width / self.height
gluPerspective(40., aspect, 1., 40.)
params: fovy (field of view y), aspect, zNear, zFar
glMatrixMode(GL_MODELVIEW)

glutDisplayFunc(self.display)
glutSpecialFunc(self.keyboard)
glutMainLoop()

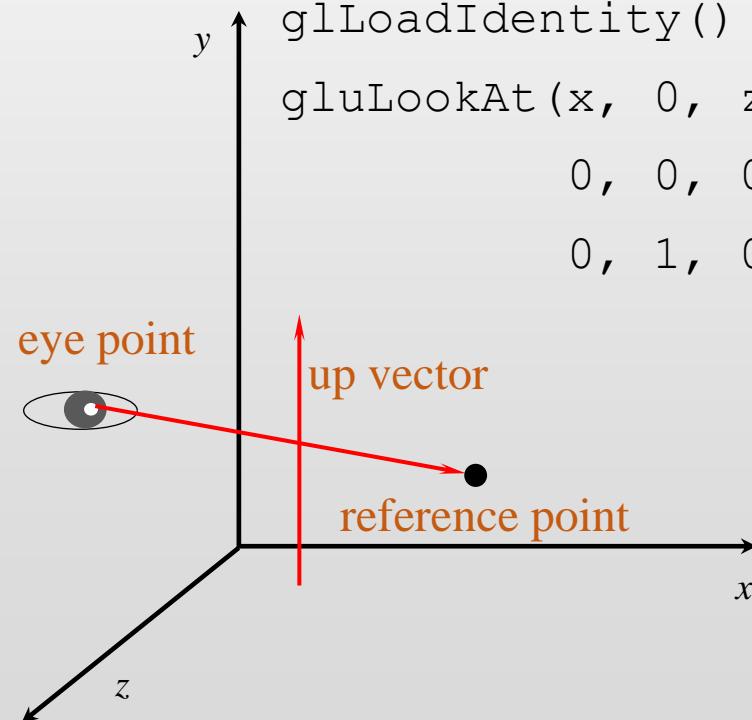
def keyboard(self, key, x, y):
 pass

```



```
def display(self):
 x = math.sin(self.angle) * self.distance # 0*20
 z = math.cos(self.angle) * self.distance # 1*20
clear buffers

 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
 glLoadIdentity() # 4*4 identity matrix
 gluLookAt(x, 0, z, # x, y, z for eye point
 0, 0, 0, # x, y, z for reference point
 0, 1, 0) # direction of the up vector
```



```

setting light (light position, Diffuse RGBA intensity, (점광원, 평행광원, ...))
constant attenuation factor, linear attenuation factor
 glLightfv(GL_LIGHT0, GL_POSITION, [15, 5, 15, 1]) # x, y, z, w
 glLightfv(GL_LIGHT0, GL_DIFFUSE, [1., 1., 1., 1.])
 glLightfv(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 0.1)
 glLightfv(GL_LIGHT0, GL_LINEAR_ATTENUATION, 0.05)
 # GL_QUADRATIC_ATTENUATION
 glPushMatrix() # stack push, create
material, GL_FRONT/GL_BACK/GL_FRONT_AND_BACK
 glMaterialfv(GL_FRONT, GL_DIFFUSE, [1., 1., 1., 1.])
 glutSolidSphere(2, 40, 40) # radius, slices, stacks
 glPopMatrix() # stack pop, delete
slices: The number of subdivisions around the Z axis (similar to lines of
longitude).
stacks: The number of subdivisions along the Z axis (similar to lines of
latitude).

```

$$f_{att}(d) = \frac{1}{a + bd + cd^2}$$

```
glPushMatrix()
glTranslatef(4, 2, 0)
glMaterialfv(GL_FRONT, GL_DIFFUSE, [1., 0.4, 0.4, 1.0])
glutSolidSphere(1, 40, 40)
glPopMatrix()

glutSwapBuffers()

if __name__ == '__main__':
 app = App()
 app.start()
```

# Refactoring Program (1)

```
chapter5_02.py, Light, Sphere class 작성

class App(object):

 def __init__(self, width=800, height=600):
 ...
 self.light = Light(GL_LIGHT0, (15, 5, 15, 1))
 self.sphere1 = Sphere(2, (0, 0, 0), (1, 1, 1, 1))
 self.sphere2 = Sphere(1, (4, 2, 0), (1, 0.4, 0.4, 1))

 def display(self):
 ...
 self.light.render()
 self.sphere1.render()
 self.sphere2.render()
 glutSwapBuffers()
```

# Refactoring Program (2)

```
class App(object):
 ...
def keyboard(self, key, x, y):
 if key == GLUT_KEY_INSERT: sys.exit()
 if key == GLUT_KEY_UP: self.distance -= 0.1
 if key == GLUT_KEY_DOWN: self.distance += 0.1
 if key == GLUT_KEY_LEFT: self.angle -= 0.05
 if key == GLUT_KEY_RIGHT: self.angle += 0.05
 if key == GLUT_KEY_F1: self.light.switch_color()

 self.distance = max(10, min(self.distance, 20))
 self.angle %= math.pi * 2
 glutPostRedisplay()
```

# Refactoring Program (3)

```
class Light(object):
 enabled = False
 colors = [(1.,1.,1.,1.), (1.,0.5,0.5,1.),
 (0.5,1.,0.5,1.), (0.5,0.5,1.,1.)]
 def __init__(self, light_id, position):
 self.light_id = light_id
 self.position = position
 self.current_color = 0
 def render(self):
 light_id = self.light_id
 color = Light.colors[self.current_color]
 glLightfv(light_id, GL_POSITION, self.position)
 glLightfv(light_id, GL_DIFFUSE, color)
 glLightfv(light_id, GL_CONSTANT_ATTENUATION, 0.1)
 glLightfv(light_id, GL_LINEAR_ATTENUATION, 0.05)
```

# Refactoring Program (3)

```
def switch_color(self):
 self.current_color += 1
 self.current_color %= len(Light.colors)

def enable(self):
 if not Light.enabled:
 glEnable(GL_LIGHTING)
 Light.enabled = True
 glEnable(self.light_id)
```

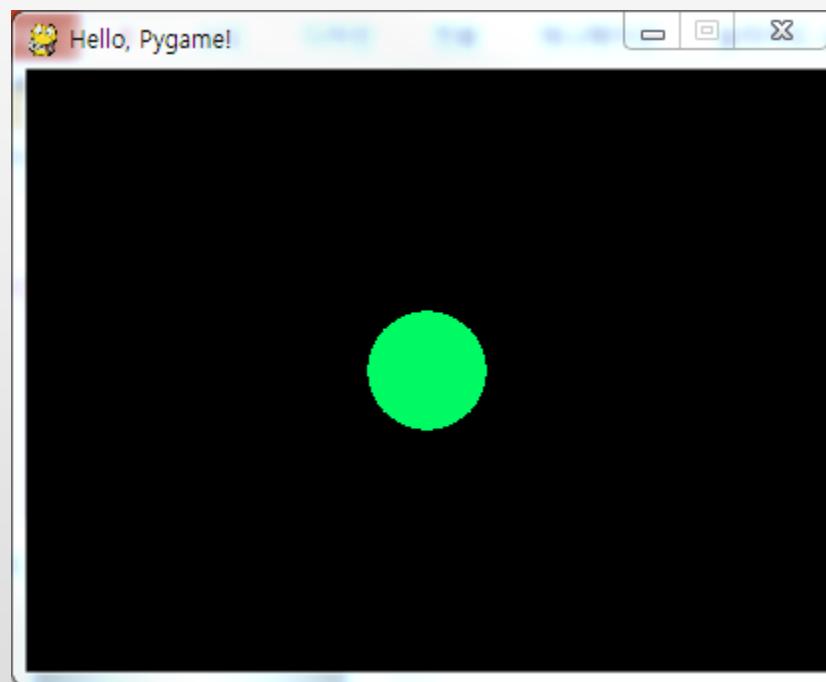
# Refactoring Program (4)

```
class Sphere(object):
 slices = 40
 stacks = 40
 def __init__(self, radius, position, color):
 self.radius = radius
 self.position = position
 self.color = color

 def render(self):
 glPushMatrix()
 glTranslatef(*self.position)
 glMaterialfv(GL_FRONT, GL_DIFFUSE, self.color)
 glutSolidSphere(self.radius, Sphere.slices, Sphere.stacks)
 glPopMatrix()
```

# Adding Pygame (1)

```
chapter5_03.py
```



# Adding Pygame (2)

```
import sys
import pygame
from pygame.locals import *

class App(object):
 def __init__(self, width=400, height=300):
 self.title = 'Hello, Pygame!'
 self.fps = 100 # for Pygame clock, fps
 self.width = width
 self.height = height
 self.circle_pos = width/2, height/2
```

# Adding Pygame (3)

```
def start(self):
 pygame.init() # Pygame initializaton
 size = (self.width, self.height)
 screen = pygame.display.set_mode(size, DOUBLEBUF)
 pygame.display.set_caption(self.title)
 clock = pygame.time.Clock()
 while True:
 dt = clock.tick(self.fps) # clock update, ms
 # runtime speed
 for event in pygame.event.get(): # get events
 if event.type == QUIT:
 pygame.quit()
 sys.exit()
 pressed = pygame.key.get_pressed()
 # list with pressed keys
```

# Adding Pygame (4)

```
x, y = self.circle_pos
if pressed[K_UP]: y -= 0.5 * dt
if pressed[K_DOWN]: y += 0.5 * dt
if pressed[K_LEFT]: x -= 0.5 * dt
if pressed[K_RIGHT]: x += 0.5 * dt
self.circle_pos = x, y
screen.fill((0, 0, 0))

draw circle
pygame.draw.circle(screen, (0, 250, 100),
 (int(x), int(y)), 30)

updates screen
pygame.display.flip()
```

# Adding Pygame (5)

```
if __name__ == '__main__':
 app = App()
 app.start()
```

# Pygame integration (1)

```
chapter5_04.py
```

```
import pygame
from pygame.locals import *

class App(object):
 def __init__(self, width=800, height=600):
 self.title = 'OpenGL demo'
 self.fps = 60
 ...
 def start(self):
 pygame.init()
 pygame.display.set_mode((self.width, self.height),
 OPENGL | DOUBLEBUF)
 pygame.display.set_caption(self.title)
```

# Pygame integration (2)

```
class App(object):
 def start(self):
 ...
 clock = pygame.time.Clock()
 while True:
 dt = clock.tick(self.fps)
 self.process_input(dt)
 self.display()
```

# Pygame integration (3)

```
class App(object):
 ...
 def display(self):
 ...
 pygame.display.flip()
 def process_input(self, dt):
 for event in pygame.event.get():
 if event.type == QUIT:
 self.quit()
 if event.type == KEYDOWN:
 if event.key == K_ESCAPE:
 self.quit()
 if event.key == K_F1:
 self.light.switch_color()
```

# Pygame integration (4)

```
class App(object):
 ...

 def process_input(self, dt):
 ...

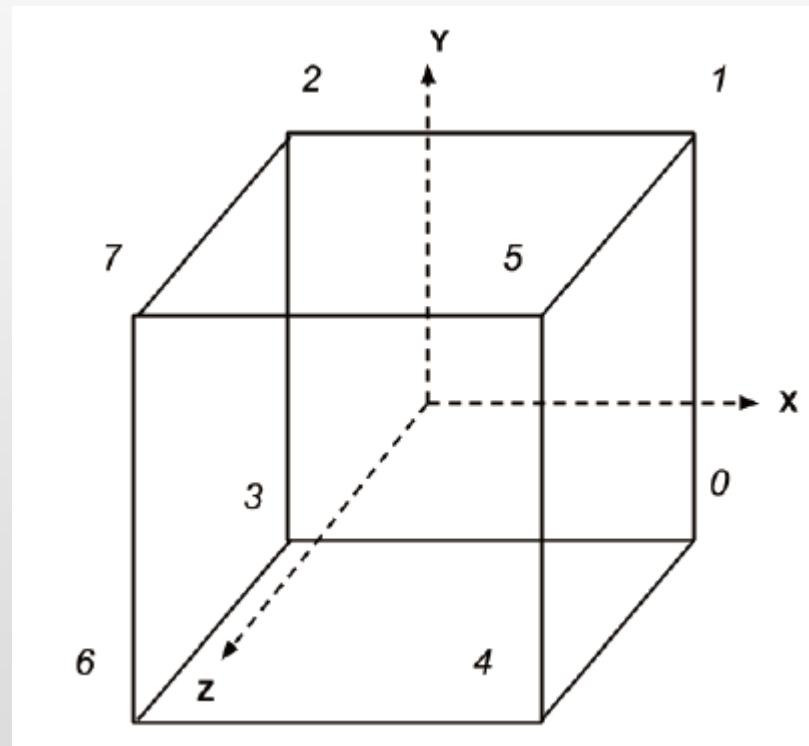
 pressed = pygame.key.get_pressed()
 if pressed[K_UP]: self.distance -= 0.01 * dt
 if pressed[K_DOWN]: self.distance += 0.01 * dt
 if pressed[K_LEFT]: self.angle -= 0.005 * dt
 if pressed[K_RIGHT]: self.angle += 0.005 * dt
 self.distance = max(10, min(self.distance, 20))
 self.angle %= math.pi * 2

 def quit(self):
 pygame.quit()
 sys.exit()
```

# Pygame integration (5)

```
class Sphere(object):
 ...
 def __init__(self, radius, position, color):
 self.radius = radius
 self.position = position
 self.color = color
 self.quadratic = gluNewQuadric() # creates quadrics obj
 def render(self):
 glPushMatrix()
 glTranslatef(*self.position)
 glMaterialfv(GL_FRONT, GL_DIFFUSE, self.color)
 gluSphere(self.quadratic, self.radius,
Sphere.slices, Sphere.stacks) # glutSolidSphere
 glPopMatrix()
```

# Drawing with OpenGL (1)



# Drawing with OpenGL (2)

```
class Cube(object):
 sides = ((0,1,2,3), (3,2,7,6), (6,7,5,4),
 (4,5,1,0), (1,5,7,2), (4,0,3,6))
 normals = ((0., 0., -1.), (-1., 0., 0.), (0., 0., 1.),
 (1., 0., 0.), (0., 1., 0), (0., -1., 0.))

 def __init__(self, position, size, color):
 self.position = position
 self.color = color #Ex) size = (2, 3, 4) // scale(x,y,z)
 x, y, z = map(lambda i: i/2, size)
 self.vertices = ((x, -y, -z), (x, y, -z),
 (-x, y, -z), (-x, -y, -z),
 (x, -y, z), (x, y, z),
 (-x, -y, z), (-x, y, z))
```

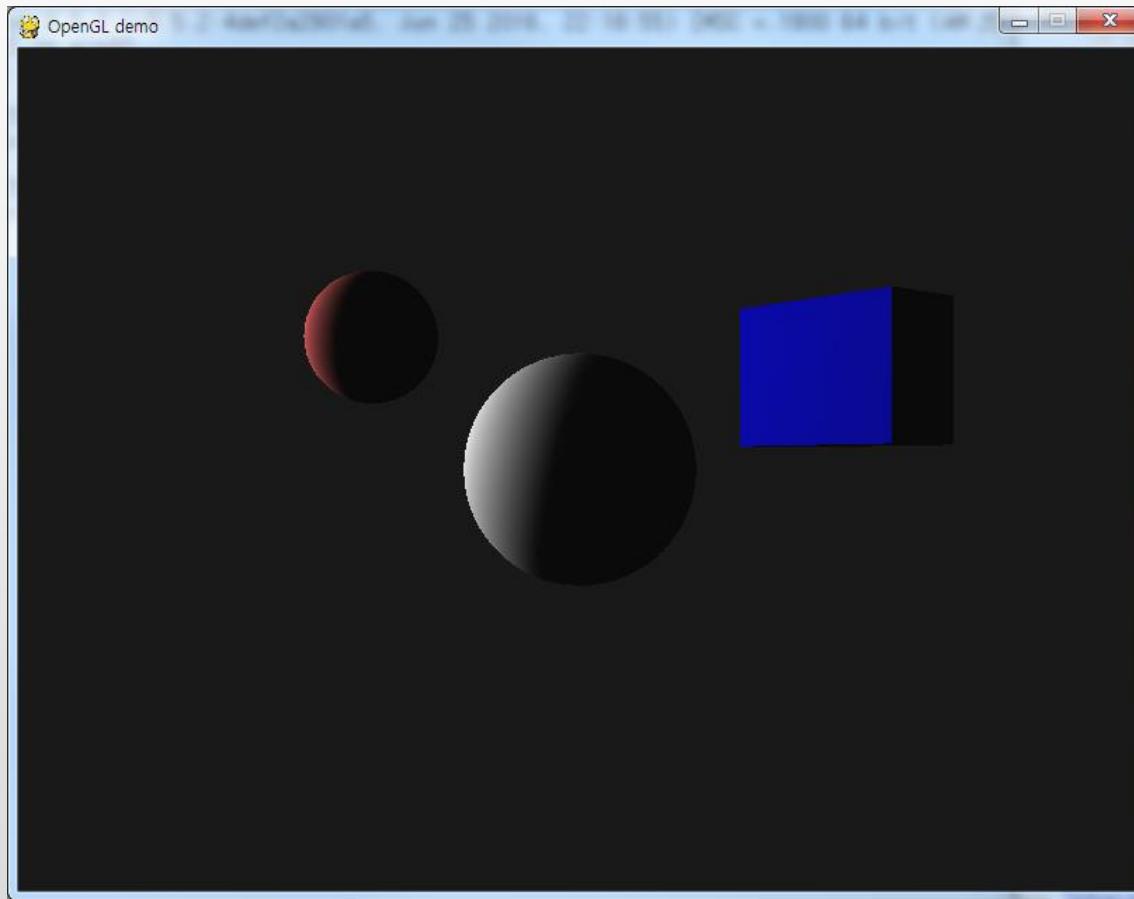
# Drawing with OpenGL (3)

```
def render(self):
 glPushMatrix()
 glTranslatef(*self.position)
 glBegin(GL_QUADS)
 # point, line, triangle, quad, polygon, ...
 glMaterialfv(GL_FRONT, GL_DIFFUSE, self.color)

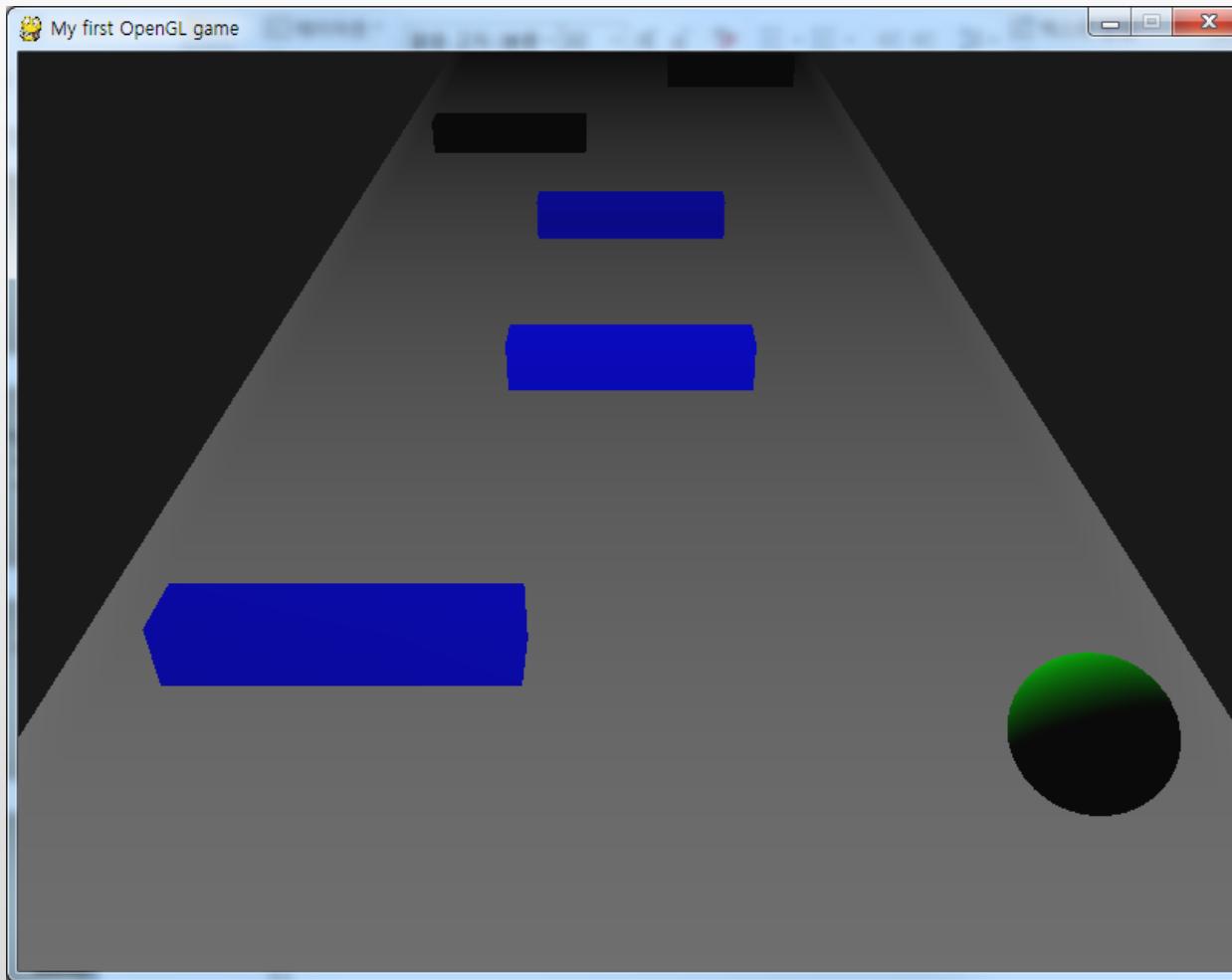
 for index, side in enumerate(Cube.sides):
 for v in side:
 glNormal3fv(self.normals[index]) # inserted
 glVertex3fv(self.vertices[v])

 glEnd()
 glPopMatrix()
```

# Drawing with OpenGL (4)



# Rolling avoid game (1)



# Rolling avoid game (2)

```
import sys
import math
import random

import pygame
from pygame.locals import *

from OpenGL.GL import *
from OpenGL.GLU import *

...
```

# Rolling avoid game (3)

```
class Block(Cube):
 color = (0, 0, 1, 1)
 speed = 0.01

 def __init__(self, position, size):
 super().__init__(position, (size, 1, 1), Block.color)
 self.size = size

 def update(self, dt):
 x, y, z = self.position
 z += Block.speed * dt
 self.position = x, y, z
```

# Rolling avoid game (4)

```
class App(object):
 def __init__(self, width=800, height=600):
 ...
 self.random_dt = 0 # time count for generating blocks
 self.blocks = []
 self.player = Sphere(1, position=(0, 0, 0),
 color=(0, 1, 0, 1))
 self.ground = Cube(position=(0, -1, -20),
 size=(16, 1, 60),
 color=(1, 1, 1, 1))
```

# Rolling avoid game (5)

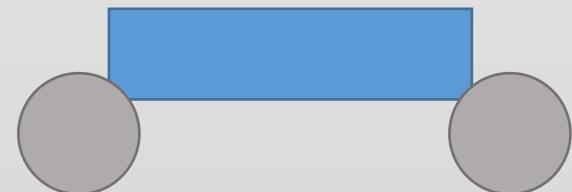
```
def start(self):
 pygame.init()
 pygame.display.set_mode((self.width, self.height),
 OPENGL | DOUBLEBUF)
 pygame.display.set_caption(self.title)
 self.light.enable()
 glEnable(GL_DEPTH_TEST)
 glClearColor(.1, .1, .1, 1)
 glMatrixMode(GL_PROJECTION)
 aspect = self.width / self.height
 gluPerspective(45, aspect, 1, 100)
 glMatrixMode(GL_MODELVIEW)
 glEnable(GL_CULL_FACE)
 self.main_loop()
```

# Rolling avoid game (6)

```
def main_loop(self):
 clock = pygame.time.Clock()
 while True:
 for event in pygame.event.get():
 if event.type == QUIT:
 pygame.quit(); sys.exit()
 if not self.game_over:
 self.display()
 dt = clock.tick(self.fps)
 for block in self.blocks:
 block.update(dt)
 self.clear_past_blocks()
 self.add_random_block(dt)
 self.check_collisions()
 self.process_input(dt)
```

# Rolling avoid game (7)

```
def check_collisions(self):
 blocks = filter(lambda x: 0 < x.position[2] < 1,
 self.blocks) # 0<block.z<1, z collision
 x = self.player.position[0]
 r = self.player.radius
 for block in blocks:
 x1 = block.position[0]
 s = block.size / 2
 if x1-s < x-r < x1+s or x1-s < x+r < x1+s:# x collision
 self.game_over = True
 print("Game over!")
```



# Rolling avoid game (8)

```
def add_random_block(self, dt):
 self.random_dt += dt
 if self.random_dt >= 800:
 r = random.random()
 if r < 0.1:
 self.random_dt = 0
 self.generate_block(r)

def generate_block(self, r):
 size = 7 if r < 0.03 else 5
 offset = random.choice([-4, 0, 4]) # choice from list
 self.blocks.append(Block((offset, 0, -40), size))
```

# Rolling avoid game (9)

```
def clear_past_blocks(self):
 blocks = filter(lambda x: x.position[2] > 5, self.blocks)
 for block in blocks:
 self.blocks.remove(block)
 del block

def display(self):
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
 glLoadIdentity()
 gluLookAt(0, 10, 10, 0, 0, -5, 0, 1, 0)
 self.light.render()
 for block in self.blocks:
 block.render()
 self.player.render()
 self.ground.render()
 pygame.display.flip()
```

# Rolling avoid game (10)

```
def process_input(self, dt):
 pressed = pygame.key.get_pressed()
 x, y, z = self.player.position
 if pressed[K_LEFT]:
 x -= 0.01 * dt
 if pressed[K_RIGHT]:
 x += 0.01 * dt
 x = max(min(x, 7), -7) # limit range
 self.player.position = (x, y, z)

if __name__ == '__main__':
 app = App()
 app.start()
```

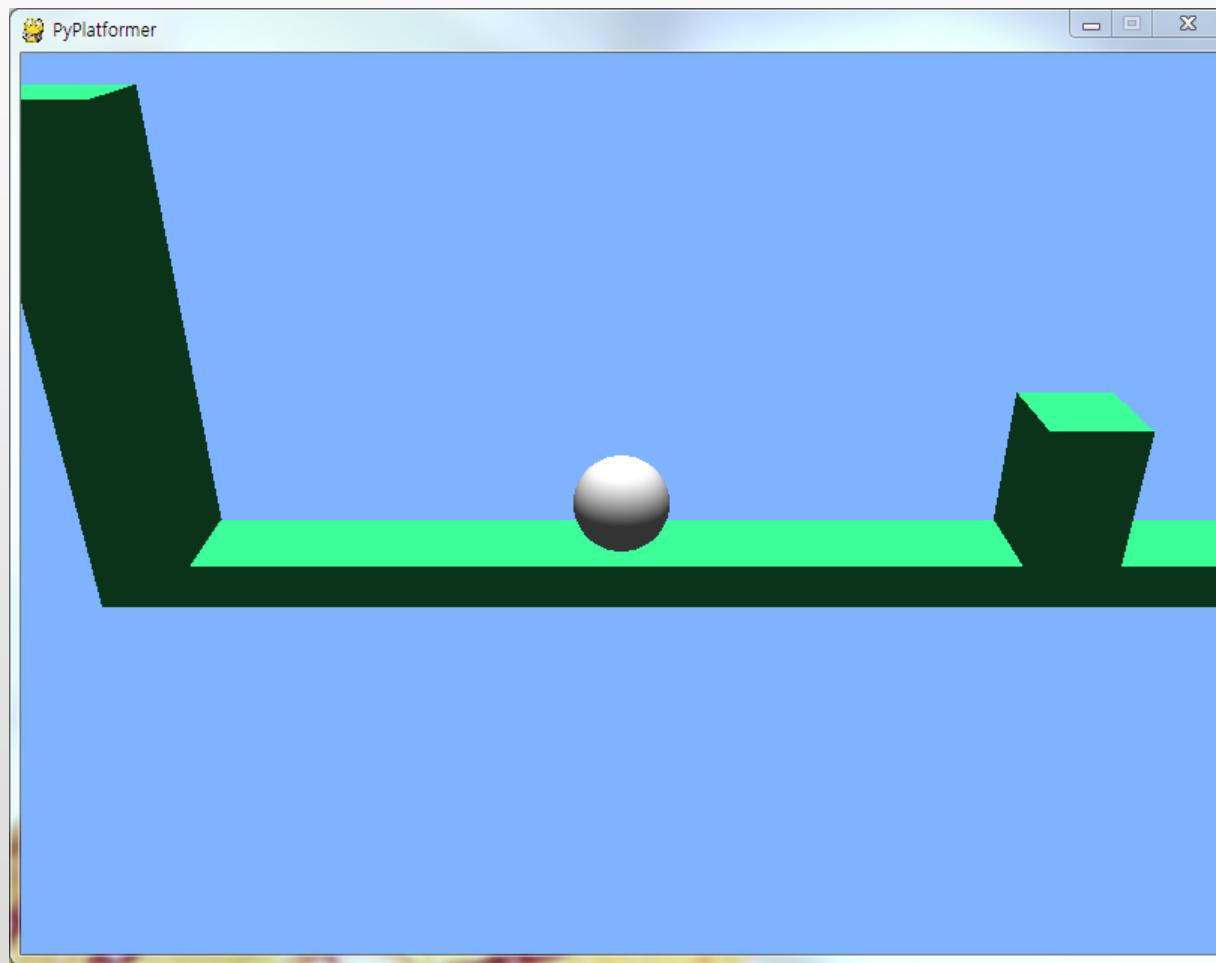
# 6. PyPlatformer

Kyung Hee University  
Daeho Lee

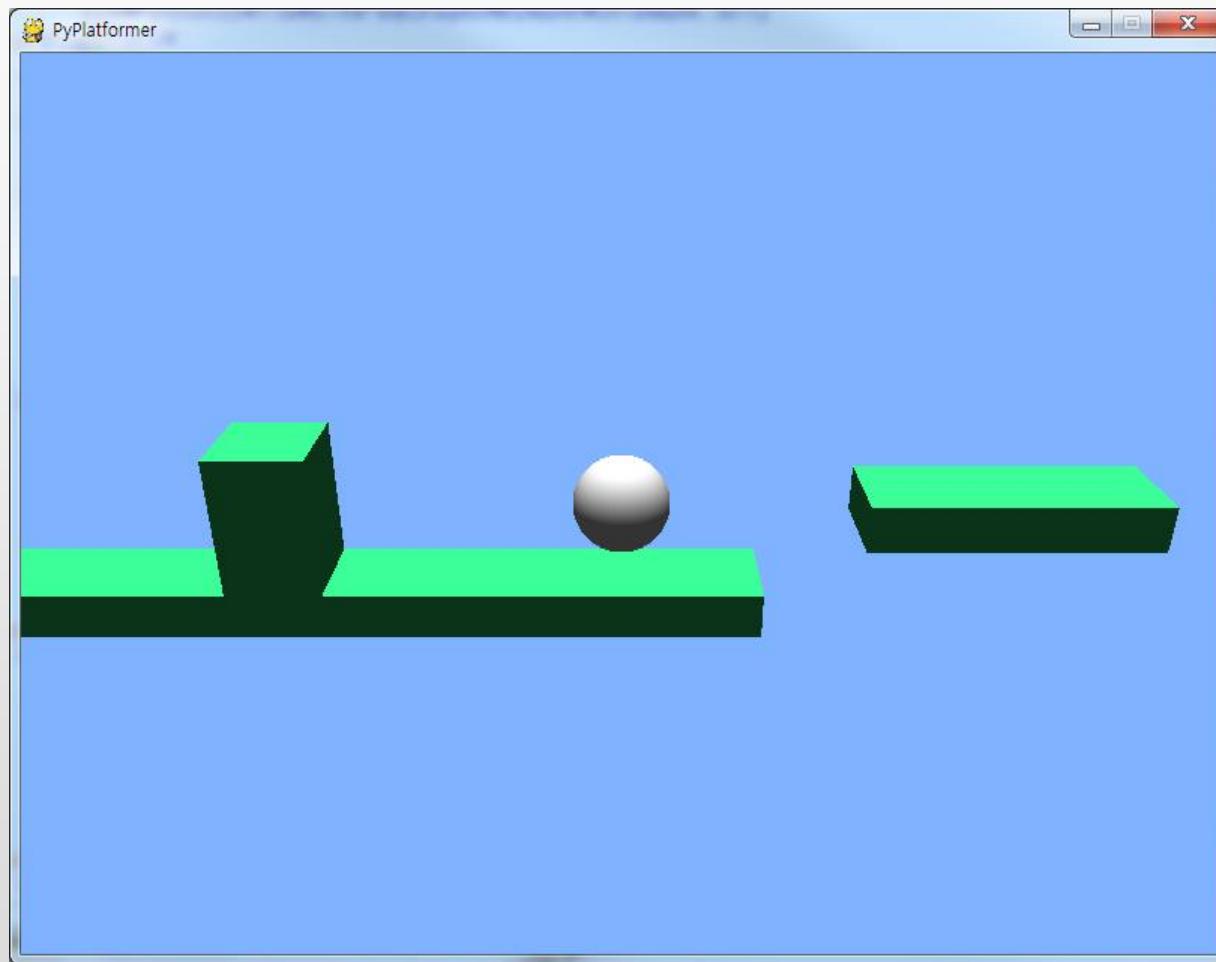
## ◆ Platformer

- Subgenre of action game
- Jumping suspended platforms, avoiding obstacles

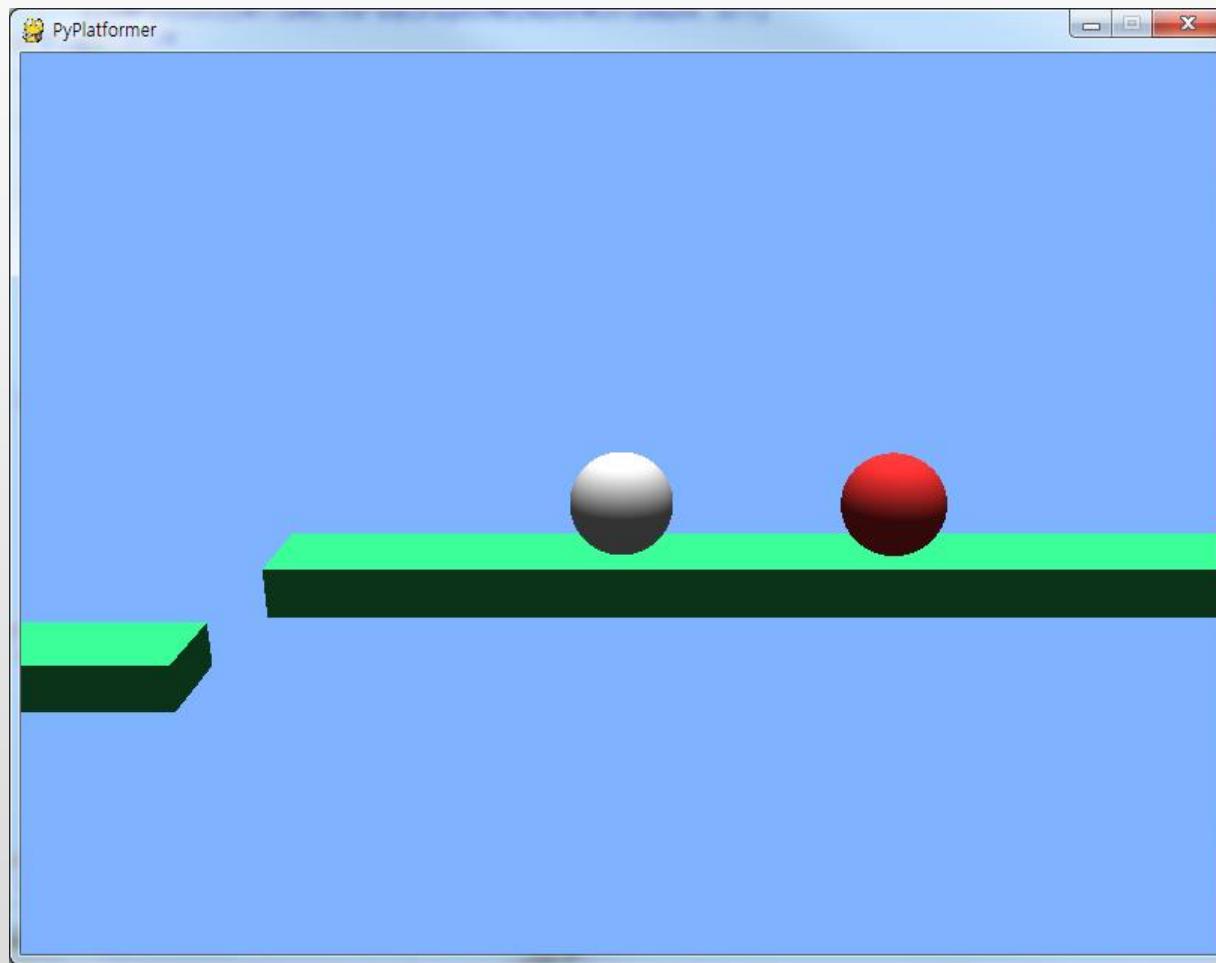
# PyPlatformer (1)



# PyPlatformer (2)



# PyPlatformer (3)



# PyPlatformer (4)



## ◆ module1.py

```
➤ def fn(n):
➤ return n*n
```

## ◆ test.py

```
➤ import module1

➤ print(module1.fn(1))
➤ print(module1.__name__) # module1
➤ newFn = module1.fn
➤ print(newFn(2))
```

## ◆ package1\pk\_module1.py

```
def fn(n):
 return n*n*n
```

## ◆ test2.py

```
import package1.pk_module1
print(package1.pk_module1.fn(3))
```

## ◆ \_\_init\_\_.py # package initialization file

```
list module for "from module import *"
__all__ = ['pk_module1']
```

## ◆ test2.py

```
from package1 import *
print(pk_module1.fn(3))
```

- ◆ @classmethod
- ◆ @staticmethod

```
class A:
 data = 10

 @classmethod
 def fn1(cls, i, j):
 print('A class', cls.data + i + j)
```

```
@staticmethod
def fn2(i, j):
 print('A class', i + j)
```

```
A.fn1(1, 2)
A.fn2(1, 2)
```

## ◆ Instance attributes

- [Default]: dynamic creation, Dict, wastes RAM, `[__slots__]`: fixed

```
class A(object):
 def __init__(self, x, y):
 self.x = x; self.y = y

class B(object):
 __slots__ = ['x', 'y']
 def __init__(self, x, y):
 self.x = x; self.y = y

class C(object):
 __slots__ = ['x', 'y', '__dict__']
 def __init__(self, x, y):
 self.x = x; self.y = y
```

```
a = A(1, 2)
a.z = 3
b = B(1, 2)
#b.z = 3 #impossible
c = C(1, 2)
c.z = 3
print(a.__dict__) # {'z': 3, 'y': 2, 'x': 1}
print(b.__slots__) # ['x', 'y']
print(c.__dict__) # {'z': 3}
print(c.__slots__) # ['x', 'y', '__dict__']
```

## ◆ Pymunk

- 2D physics engine built on top of Chipmunk2D
- > pip install pymunk
- 5.3.2
- Chipmunk2D : 2D physics engine

## ◆ Space

- 2D space, Space(), updated by step()

## ◆ Body

- Rigid body, Body(), position, velocity, force

## ◆ Shape

- All shapes, Circle(), Poly()

## ◆ Arbiter

- Collision pair, used in collision callbacks, contact point set

```
import pymunk
import pygame
import pymunk.pygame_util
import sys
pygame.init()
screen = pygame.display.set_mode((600, 600))
draw_options = pymunk.pygame_util.DrawOptions(screen)
space = pymunk.Space()
space.gravity = 0, -100
body = pymunk.Body(1, 20) # Body(mass, moment, body_type)
body.position = 50, 550 # body_type: DYNAMIC, KINEMATIC, STATIC
shape = pymunk.Circle(body, 10, (0,0)) # Circle(body, r, offset)
space.add(body, shape) # add(*objs)
```

```
clock = pygame.time.Clock()

for i in range(0, 500):
 screen.fill((255,255,255))
 space.debug_draw(draw_options)
 space.step(1/50.0) # update space, step(dt)
 pygame.display.flip()
 clock.tick(50)

pygame.quit()
```

<http://www.pymunk.org/en/latest/tutorials/SlideAndPinJoint.html>

...

```
vs = [(-23,26), (23,26), (0,-26)]
moment = pymunk.moment_for_poly(1, vs)
body2 = pymunk.Body(1, moment)
body2.position = 100, 550
shape2 = pymunk.Poly(body2, vs)
shape2.friction = 0.
shape2.elasticity = 0.5
space.add(body2, shape2)
```

```
body3 = pymunk.Body(body_type = pymunk.Body.STATIC)
body3.position = 300, 100
shape3 = pymunk.Poly(body3,
 [(-300, 0), (300, 0), (300, -2), (-300, -2)])
shape3.friction = 0.
shape3.elasticity = 0.5
space.add(body3, shape3)
body2.apply_impulse_at_local_point((60, 00), (0, 10))
...
...
```

# Error Corrections (1)

```
game.py

class PlayerMovement(Component):

 ...

 def on_collide(self, other, contacts):
 #self.can_jump = any(c.normal.y < 0 for c in contacts)
 if contacts.normal[1] < 0:
 self.can_jump = True
 else:
 self.can_jump = False
```

```
game.py

class Shooter(Component):
 __slots__ = ['ammo']

 def __init__(self):
 self.ammo = 0

 def update(self, dt):
 #if Input.get_key_down(K_SPACE) and self.ammo >= 0:
 if Input.get_key_down(K_SPACE) and self.ammo > 0:
```

```
__init__.py

class GameObject(object):
 ...

 def move(self, x, y):
 #self._body.apply_impulse((x, y))
 self._body.apply_impulse_at_local_point((x, y))

 def apply_force(self, x, y):
 #self._body.apply_force((x, y))
 self._body.apply_force_at_world_point((x, y), (0,0))
```

```
physics.py

#def coll_handler(_, arbiter):
def coll_handler(arbiter, space, _):
 if len(arbiter.shapes) == 2:
 obj1 = arbiter.shapes[0].gameobject
 obj2 = arbiter.shapes[1].gameobject

 # obj1.collide(obj2, arbiter.contacts)
 # obj2.collide(obj1, arbiter.contacts)

 obj1.collide(obj2, arbiter.contact_point_set)
 obj2.collide(obj1, arbiter.contact_point_set)

 return True
```

```
physics.py
space = pymunk.Space()
space.gravity = 0, -10
#space.set_default_collision_handler(coll_handler)
h = space.add_default_collision_handler()
h.begin = coll_handler
```

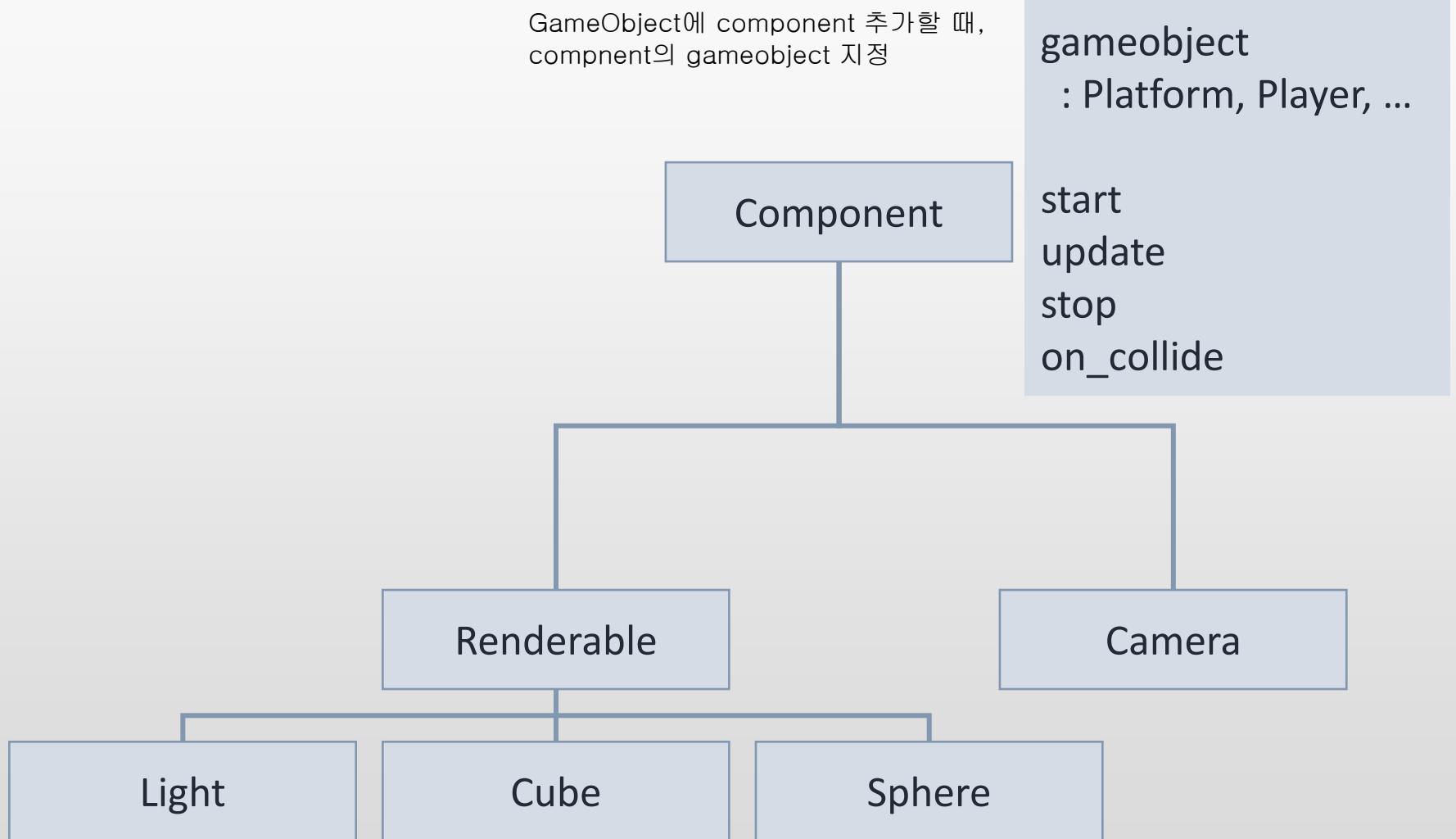
```
physics.py
class Rigidbody(Component):
 ...
 def start(self):
 if not self.is_static:
 ...
#inserted
 else:
 pos = self.gameObject._body.position
 body = pymunk.Body(body_type = pymunk.Body.STATIC)
 body.position = pos
 self.gameObject._body = body
#inserted
```

```
physics.py

class Rigidbody(Component):
 ...

 def add_shape_to_space(self, shape):
 shape.collision_type = 1
 self.gameobject._shape = shape
 shape.gameobject = self.gameobject
 if self.is_static:
 #space.add(shape)
 space.add(self.gameobject._body, shape)
 else:
 space.add(self.gameobject._body, shape)
```

# Component-based Game Engine (1)



# Component-based Game Engine (2)

GameObject

components  
position  
velocity  
move  
render  
remove  
collide  
  
...

Game

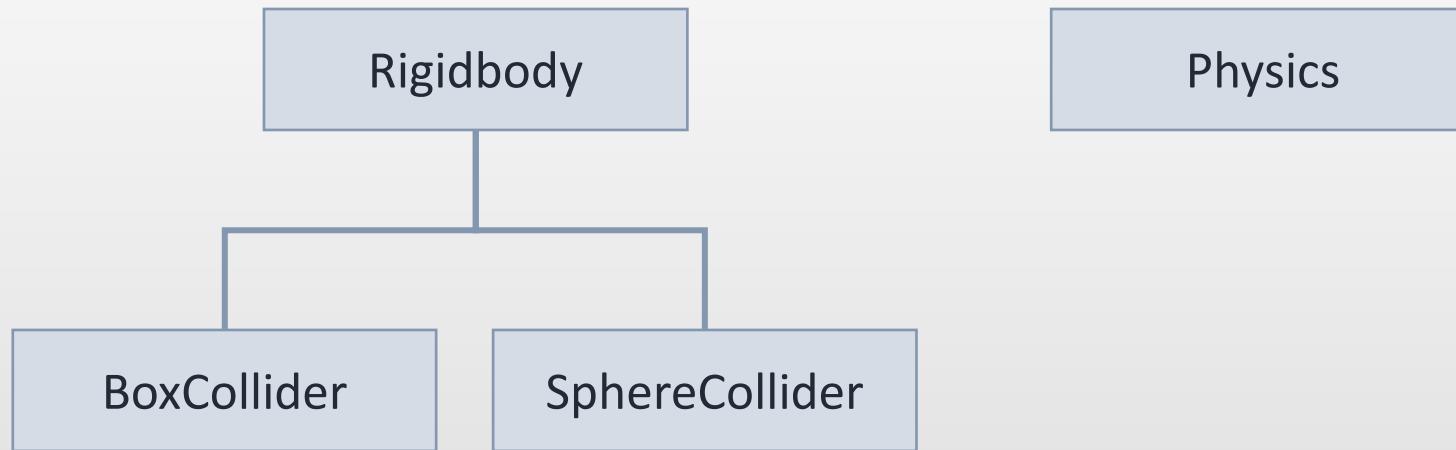
width, height  
mainloop  
setup  
update  
render  
  
...

Input

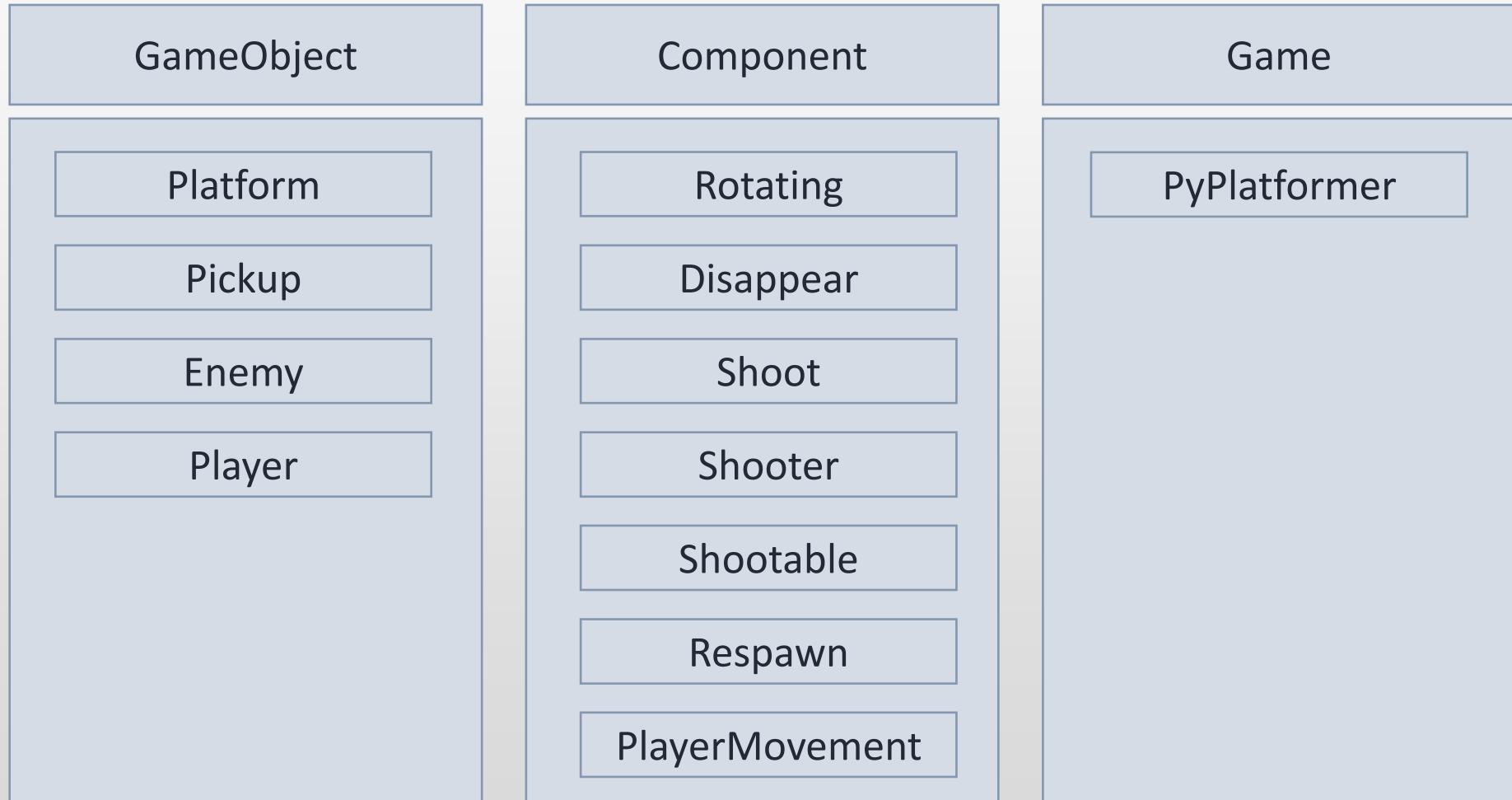
update  
get\_key  
get\_key\_down

# Component-based Game Engine (3)

physics.py



# Component-based Game Engine (4)



# Components.py (1)

```
from OpenGL.GL import *
from OpenGL.GLU import *

__all__ = ['Component', 'Renderable', 'Cube', 'Sphere',
 'Light', 'Camera']

class Component(object):
 __slots__ = ['gameobject'] # used to get the gameobject pos

 def start(self):
 pass

 def update(self, dt):
 pass

 def stop(self):
 pass

 def on_collide(self, other, contacts):
 pass
```

# Components.py (2)

```
class Renderable(Component):
 __slots__ = ['color']
 def __init__(self, color):
 self.color = color
 def render(self):
 pos = self.gameObject.position
 rot = self.gameObject.rotation
 scale = self.gameObject.scale
 glPushMatrix()
 glTranslatef(*pos) # translation
 if rot != (0, 0, 0): # rotation
 glRotatef(rot[0], 1, 0, 0) # rotation angle, vector
 glRotatef(rot[1], 0, 1, 0)
 glRotatef(rot[2], 0, 0, 1)
```

```
if scale != (1, 1, 1): # scaling
 glScalef(*scale)
if self.color is not None: # coloring
 glColor4f(*self.color)
self._render() # call _render
glPopMatrix()

def _render(self):
 pass
```

```
class Light(Renderable):
 def __init__(self, light_id, color=(1, 1, 1, 0),
 constant_att=0.1, linear_att=0.05):
 self.light_id = light_id
 self.enabled = False
 self.color = color
 self.constant_att = constant_att
 self.linear_att = linear_att
```

```
def render(self):
 if not self.enabled:
 self.enabled = True
 glEnable(self.light_id)
 light_id = self.light_id
 position = self.gameObject.position
 glLightfv(light_id, GL_POSITION, position)
 glLightfv(light_id, GL_DIFFUSE, self.color)
 glLightfv(light_id, GL_CONSTANT_ATTENUATION,
 self.constant_att)
 glLightfv(light_id, GL_LINEAR_ATTENUATION,
 self.linear_att)
```

```
class Camera(Component):
 instance = None

 def __init__(self, dy, dz):
 self.dy = dy
 self.dz = dz
 Camera.instance = self

 def render(self):
 pos = self.gameObject.position
 glLoadIdentity()
 gluLookAt(pos[0], self.dy, self.dz, # view
 pos[0], pos[1], pos[2],
 0, 1, 0)
```

```
class Cube(Renderable):
 sides = ((0, 1, 2, 3), (3, 2, 7, 6), (6, 7, 5, 4),
 (4, 5, 1, 0), (1, 5, 7, 2), (4, 0, 3, 6))
 normals = ((0, 0, -1), (-1, 0, 0), (0, 0, 1),
 (1, 0, 0), (0, 1, 0), (0, -1, 0))

 def __init__(self, color, size):
 super(Cube, self).__init__(color)
 x, y, z = map(lambda i: i/2, size)
 self.vertices = (
 (x, -y, -z), (x, y, -z),
 (-x, y, -z), (-x, -y, -z),
 (x, -y, z), (x, y, z),
 (-x, -y, z), (-x, y, z))
```

```
def _render(self):
 glBegin(GL_QUADS)
 for i, side in enumerate(Cube.sides):
 glNormal3fv(Cube.normals[i])
 for v in side:
 glVertex3fv(self.vertices[v])
 glEnd()
```

```
class Sphere(Renderable):
 slices = 40
 stacks = 40

 def __init__(self, radius, color):
 super(Sphere, self).__init__(color)
 self.radius = radius
 self.quadratic = gluNewQuadric()

 def _render(self):
 gluSphere(self.quadratic, self.radius,
 Sphere.slices, Sphere.stacks)
```

# Input\_manager.py (1)

```
from collections import defaultdict
import pygame

class Input(object):
 quit_flag = False
 keys = defaultdict(bool)
 keys_down = defaultdict(bool)
```

## Input\_manager.py (2)

@classmethod

```
def update(cls): # called in Game.update
 cls.keys_down.clear()
 for event in pygame.event.get():
 if event.type == pygame.QUIT:
 cls.quit_flag = True
 if event.type == pygame.KEYUP:
 cls.keys[event.key] = False
 if event.type == pygame.KEYDOWN:
 cls.keys[event.key] = True
 cls.keys_down[event.key] = True
```

```
@classmethod
def get_key(cls, key): # called in PlayerMovement.update
 return cls.keys[key]
```

```
@classmethod
def get_key_down(cls, key): # called in Shooter.update
 return cls.keys_down[key]
```

```
import pymunk
from .components import Component
__all__ = ['BoxCollider', 'SphereCollider', 'Physics',
'Rigidbody']

def coll_handler(arbiter, space, _): # pymunk.Space
 if len(arbiter.shapes) == 2: # collision handler
 obj1 = arbiter.shapes[0].gameobject
 obj2 = arbiter.shapes[1].gameobject

 obj1.collide(obj2, arbiter.contact_point_set)
 obj2.collide(obj1, arbiter.contact_point_set)
 return True
```

```
space = pymunk.Space()
space.gravity = 0, -10
h = space.add_default_collision_handler()
h.begin = coll_handler

class Rigidbody(Component):
 __slots__ = ['mass', 'is_static']

 def __init__(self, mass=1, is_static=True):
 self.mass = mass
 self.is_static = is_static
```

```
def start(self):
 if not self.is_static:
 # Replace the static body
 pos = self.gameobject._body.position
 body = pymunk.Body(self.mass, 1666)
 body.position = pos
 self.gameobject._body = body
#inserted
 else: # for static body
 pos = self.gameobject._body.position
 body = pymunk.Body(body_type = \
 pymunk.Body.STATIC)
 body.position = pos
 self.gameobject._body = body
#inserted
```

```
def add_shape_to_space(self, shape):
 shape.collision_type = 1
 self.gameobject._shape = shape
 shape.gameobject = self.gameobject
 if self.is_static:
 #space.add(shape)
 space.add(self.gameobject._body, shape)
 else:
 space.add(self.gameobject._body, shape)
```

```
class BoxCollider(Rigidbody):
 __slots__ = ['size']

 def __init__(self, width, height, mass=1,
 is_static=True):
 super(BoxCollider, self).__init__(mass, is_static)
 self.size = width, height

 def start(self):
 super(BoxCollider, self).start()
 body = self.gameObject._body
 shape = pymunk.Poly.create_box(body, self.size)
 self.add_shape_to_space(shape) # Rigidbody method
```

```
class SphereCollider(Rigidbody):
 __slots__ = ['radius']

 def __init__(self, radius, mass=1, is_static=True):
 super(SphereCollider, self).__init__(mass,
 is_static)
 self.radius = radius

 def start(self):
 super(SphereCollider, self).start()
 body = self.gameObject._body
 shape = pymunk.Circle(body, self.radius)
 self.add_shape_to_space(shape)
```

```
class Physics(object):
 @classmethod
 def step(cls, dt):
 space.step(dt)

 @classmethod
 def remove(cls, body):
 space.remove(body)
```

```
import sys
import pygame
import pymunk as pm

from OpenGL.GL import *
from OpenGL.GLU import *

from .components import *
from .input_manager import Input
from .physics import BoxCollider, SphereCollider, Physics, \
 Rigidbody
```

```
class GameObject(object):
 instances = []

 def __init__(self, x=0, y=0, z=0, scale=(1, 1, 1)):
 self._body = pm.Body()
 self._body.position = x, y
 self._shape = None
 self._ax = 0
 self._ay = 0
 self._z = z
 self.tag = ''
 self.scale = scale
 self.components = []
 GameObject.instances.append(self)
```

```
@property
def position(self):
 pos = self._body.position
 return pos.x, pos.y, self._z

@position.setter
def position(self, pos):
 self._body.position = pos[0], pos[1]
 self._z = pos[2]

@property
def rotation(self):
 return self._ax, self._ay, self._body.angle
```

```
@rotation.setter
def rotation(self, rot):
 self._ax = rot[0]
 self._ay = rot[1]
 self._body.angle = rot[2]

@property
def velocity(self):
 return self._body.velocity

@velocity.setter
def velocity(self, vel):
 self._body.velocity = vel
```

```
def move(self, x, y):
 self._body.apply_impulse_at_local_point((x, y))

def apply_force(self, x, y):
 self._body.apply_force_at_world_point((x, y), (0, 0))

def add_components(self, *components):
 for component in components:
 self.add_component(component)

def add_component(self, component):
 self.components.append(component)
 component.gameobject = self
 component.start()
```

```
def get_component_by_type(self, cls):
 for component in self.components:
 if isinstance(component, cls):
 return component

def remove_component(self, component):
 component.stop()
 self.components.remove(component)

def render(self):
 for component in self.components:
 if isinstance(component, Renderable):
 component.render()
```

```
def update(self, dt):
 for component in self.components:
 component.update(dt)

def remove(self):
 for component in self.components:
 self.remove_component(component)
 if self._shape is not None:
 Physics.remove(self._shape)
 GameObject.instances.remove(self)

def collide(self, other, contacts): # called in coll_handler
 for component in self.components:
 component.on_collide(other, contacts)
```

```
class Game(object):
 def __init__(self, caption, width=800, height=600):
 self.caption = caption
 self.width = width
 self.height = height
 self.fps = 60
 self.screen = None
```

```
def mainloop(self):
 self.setup()
 clock = pygame.time.Clock()
 while not Input.quit_flag:
 dt = clock.tick(self.fps)
 dt /= 1000
 Physics.step(dt)
 self.update(dt)
 self.render()
 pygame.quit()
 sys.exit()
```

```
def setup(self):
 pygame.init()
 size = self.width, self.height
 self.screen = pygame.display.set_mode(size,
 pygame.OPENGL | pygame.DOUBLEBUF)
 pygame.display.set_caption(self.caption)
 glEnable(GL_LIGHTING)
 glEnable(GL_COLOR_MATERIAL)
 glColorMaterial(GL_FRONT_AND_BACK,
 GL_AMBIENT_AND_DIFFUSE)
 glEnable(GL_DEPTH_TEST)
 glClearColor(0.5, 0.7, 1, 1)
 glMatrixMode(GL_PROJECTION)
 aspect = self.width / self.height
 gluPerspective(45, aspect, 1, 100)
 glMatrixMode(GL_MODELVIEW)
```

```
def update(self, dt):
 Input.update()
 for gameobject in GameObject.instances:
 gameobject.update(dt)

def render(self):
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
 if Camera.instance is not None:
 Camera.instance.render()
 for gameobject in GameObject.instances:
 gameobject.render()
 pygame.display.flip()
```

```
from pygame.locals import *
from engine import *

class Platform(GameObject):
 def __init__(self, x, y, width, height):
 super(Platform, self).__init__(x, y)
 color = (0.2, 1, 0.5, 1)
 self.add_components(Cube(color, size=(width, height, 2)),
 BoxCollider(width, height))
```

```
class Rotating(Component):
 speed = 50
 def update(self, dt):
 ax, ay, az = self.gameobject.rotation
 ay = (ay + self.speed * dt) % 360
 self.gameobject.rotation = ax, ay, az # angles
 # for x, y, z
class Pickup(GameObject):
 def __init__(self, x, y):
 super(Pickup, self).__init__(x, y)
 self.tag = 'pickup'
 color = (1, 1, 0.5, 1)
 self.add_components(Cube(color, size=(1, 1, 1)),
 Rotating(), BoxCollider(1, 1))
```

```
class Disappear(Component):
 def update(self, dt):
 self.gameobject.velocity = 0, 0
 s1, s2, s3 = map(lambda s: s - dt*2,
 self.gameobject.scale)
 self.gameobject.scale = s1, s2, s3
 if s1 <= 0: self.gameobject.remove()
```

```
class Shoot(Component):
 def on_collide(self, other, contacts):
 self.gameobject.remove()
```

```
class Shooter(Component):
 __slots__ = ['ammo']

 def __init__(self):
 self.ammo = 0

 def update(self, dt):
 if Input.get_key_down(K_SPACE) and self.ammo > 0:
 self.ammo -= 1
 direction = 1 \
 if self.gameobject.velocity.x > 0 else -1
 pos = self.gameobject.position
 shoot = GameObject(pos[0] + 1.5 * direction, pos[1])
 shoot.tag = 'shoot'
```

```
shoot.add_components(Sphere(0.3, (1, 1, 0, 1)),
 Shoot(),
 SphereCollider(0.3, mass=0.1, is_static=False))
shoot.apply_force(20 * direction, 0)

def on_collide(self, other, contacts):
 if other.tag == 'pickup':
 self.ammo += 5
 other.add_component(Dissappear())
```

```
class Shootable(Component):
 def on_collide(self, other, contacts):
 if other.tag == 'shoot':
 self.gameObject.add_component(Dissappear())

class Enemy(GameObject):
 def __init__(self, x, y):
 super(Enemy, self).__init__(x, y)
 self.tag = 'enemy'
 color = (1, 0.2, 0.2, 1)
 self.add_components(Sphere(1, color), Shootable(),
 SphereCollider(1, is_static=False))
```

```
class Respawn(Component):
 __slots__ = ['limit', 'spawn_position']

 def __init__(self, limit=-15):
 self.limit = limit
 self.spawn_position = None

 def start(self):
 self.spawn_position = self.gameobject.position

 def update(self, dt):
 if self.gameobject.position[1] < self.limit:
 self.respawn()
```

```
def on_collide(self, other, contacts):
 if other.tag == 'enemy':
 self.respawn()

def respawn(self):
 self.gameobject.velocity = 0, 0
 self.gameobject.position = self.spawn_position
```

```
class PlayerMovement(Component):
 __slots__ = ['can_jump']

 def __init__(self):
 self.can_jump = False

 def update(self, dt):
 d = Input.get_key(K_RIGHT) - Input.get_key(K_LEFT)
 self.gameobject.move(d * 5 * dt, 0)
 if Input.get_key(K_UP) and self.can_jump:
 self.can_jump = False
 self.gameobject.move(0, 8)
```

```
def on_collide(self, other, contacts):
 # self.can_jump = any(c.normal.y < 0 for c in \
 # contacts) # org

 if contacts.normal[1] < 0: # y direction
 self.can_jump = True
 else:
 self.can_jump = False
```

```
class Player(GameObject):
 def __init__(self, x, y):
 super(Player, self).__init__(x, y)
 self.add_components(Sphere(1, (1, 1, 1, 1)),
 PlayerMovement(), Respawn(),
 Shooter(), Camera(10, 20),
 SphereCollider(1, is_static=False))
```

```
class PyPlatformer(Game):
 def __init__(self):
 super(PyPlatformer, self).__init__('PyPlatformer')
 self.player = Player(-2, 0)
 self.light = GameObject(0, 10, 0)
 self.light.add_component(Light(GL_LIGHT0))
 self.ground = [
 # Platform 1
 Platform(3, -2, 30, 1),
 Platform(-11, 3, 2, 9),
 Platform(8, 0, 2, 3),
```

```
Platform 2 & 3
Platform(23, 0, 6, 1),
Platform(40, 2, 24, 1),
Platform 4 & 5
Platform(60, 3, 8, 1),
Platform(84, 4, 26, 1)

]

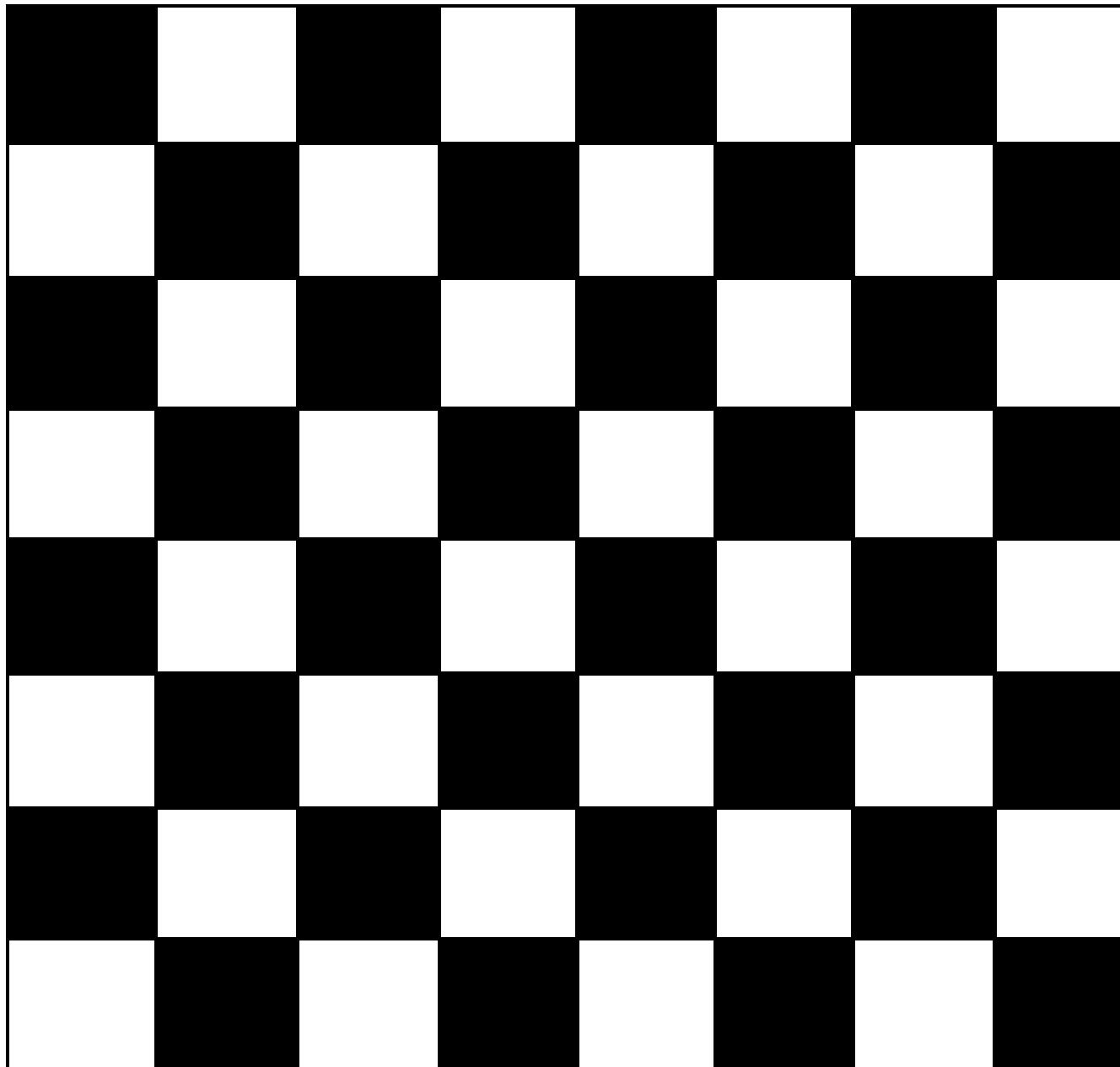
self.pickups = Pickup(60, 5)
self.enemies = [Enemy(40, 4), Enemy(90, 6)]

if __name__ == '__main__':
 game = PyPlatformer()
 game.mainloop()
```

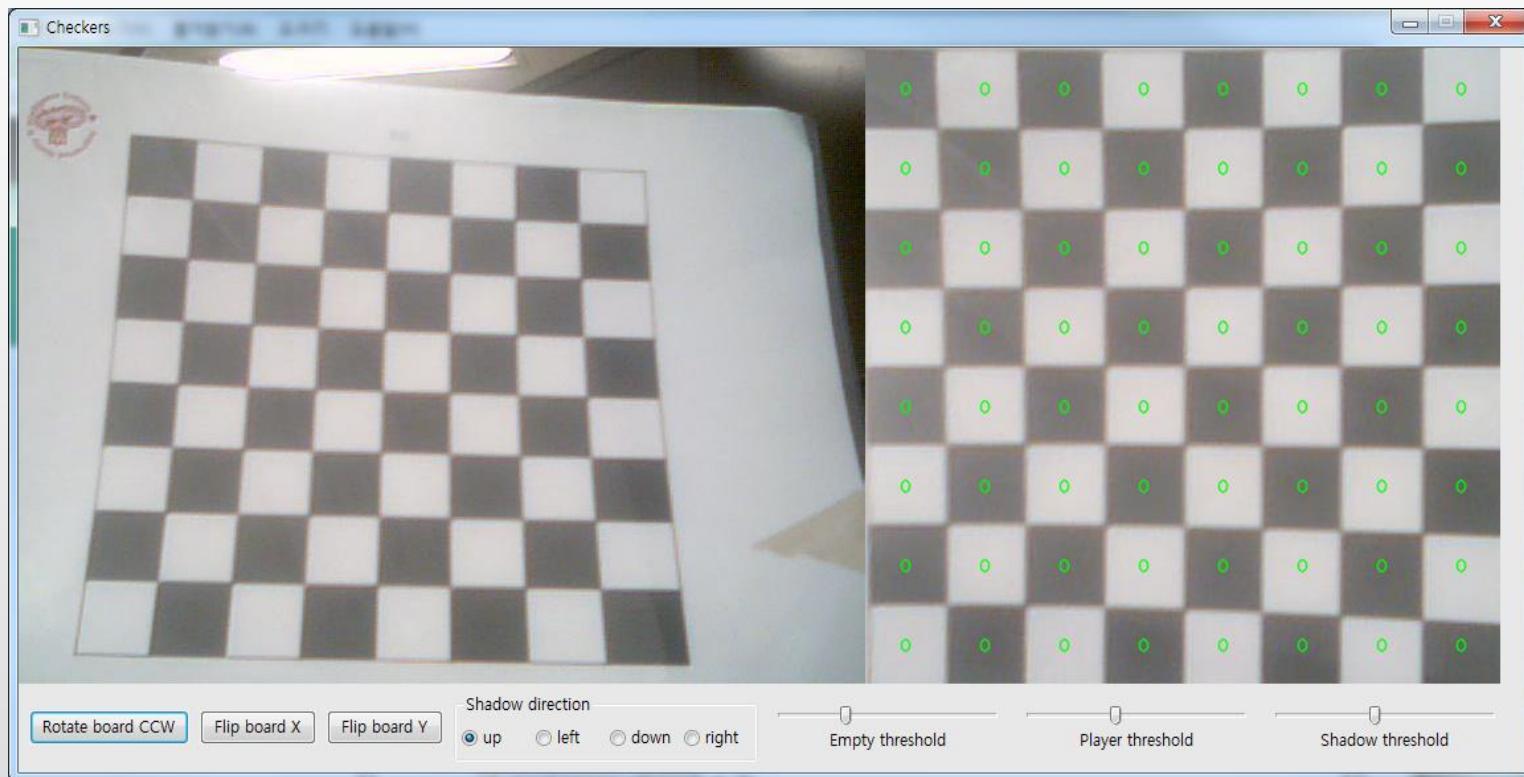
# 7. Augmenting a Board Game with Computer Vision

Kyung Hee University  
Daeho Lee

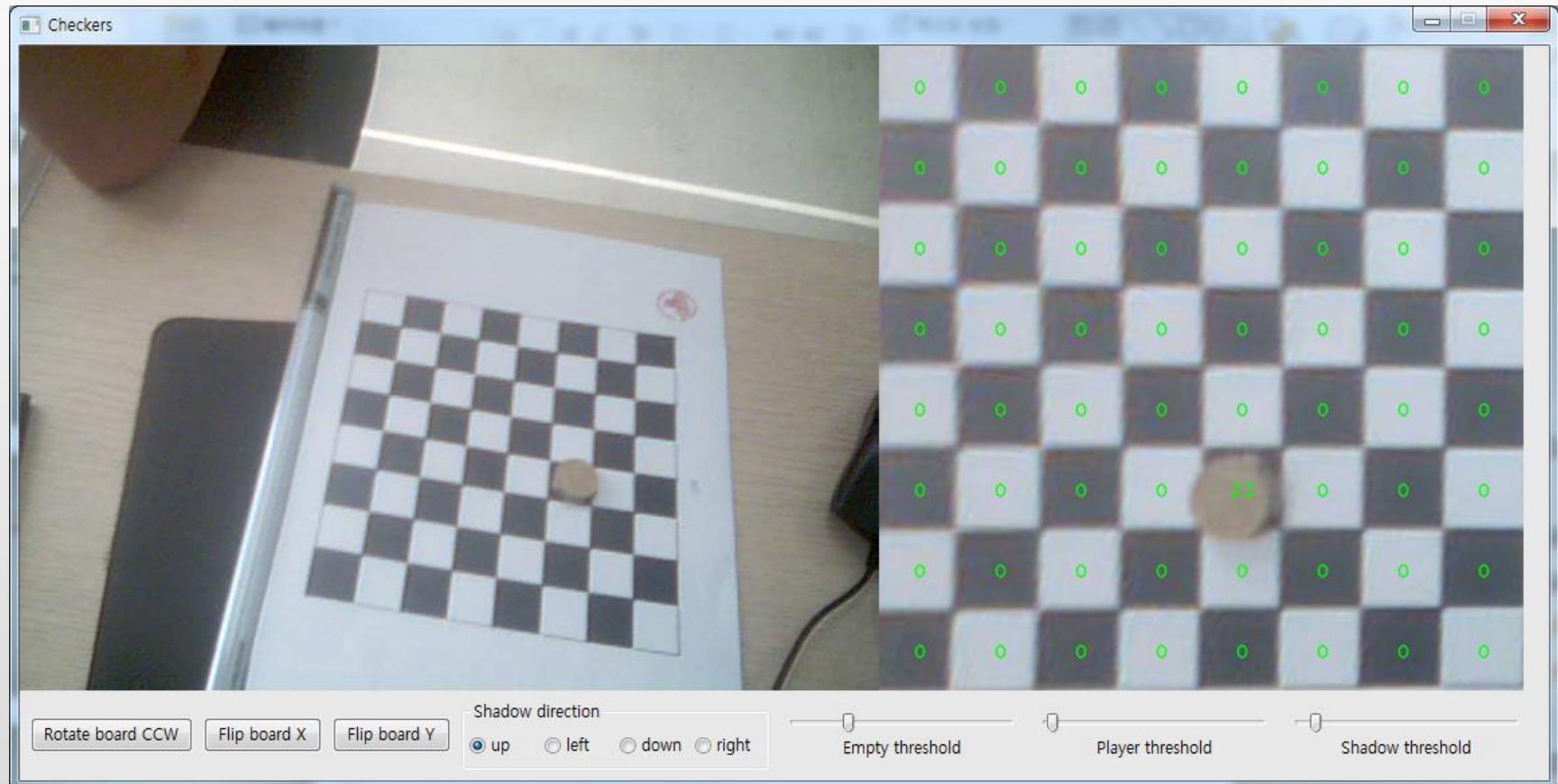
- ◆ > pip install mkl # math kernel library
- ◆ > pip install wxPython # GUI API
- ◆ > pip install sklearn # machine learning library
- ◆ > pip install scipy # scientific computing library
- ◆ > pip install opencv\_python # opencv



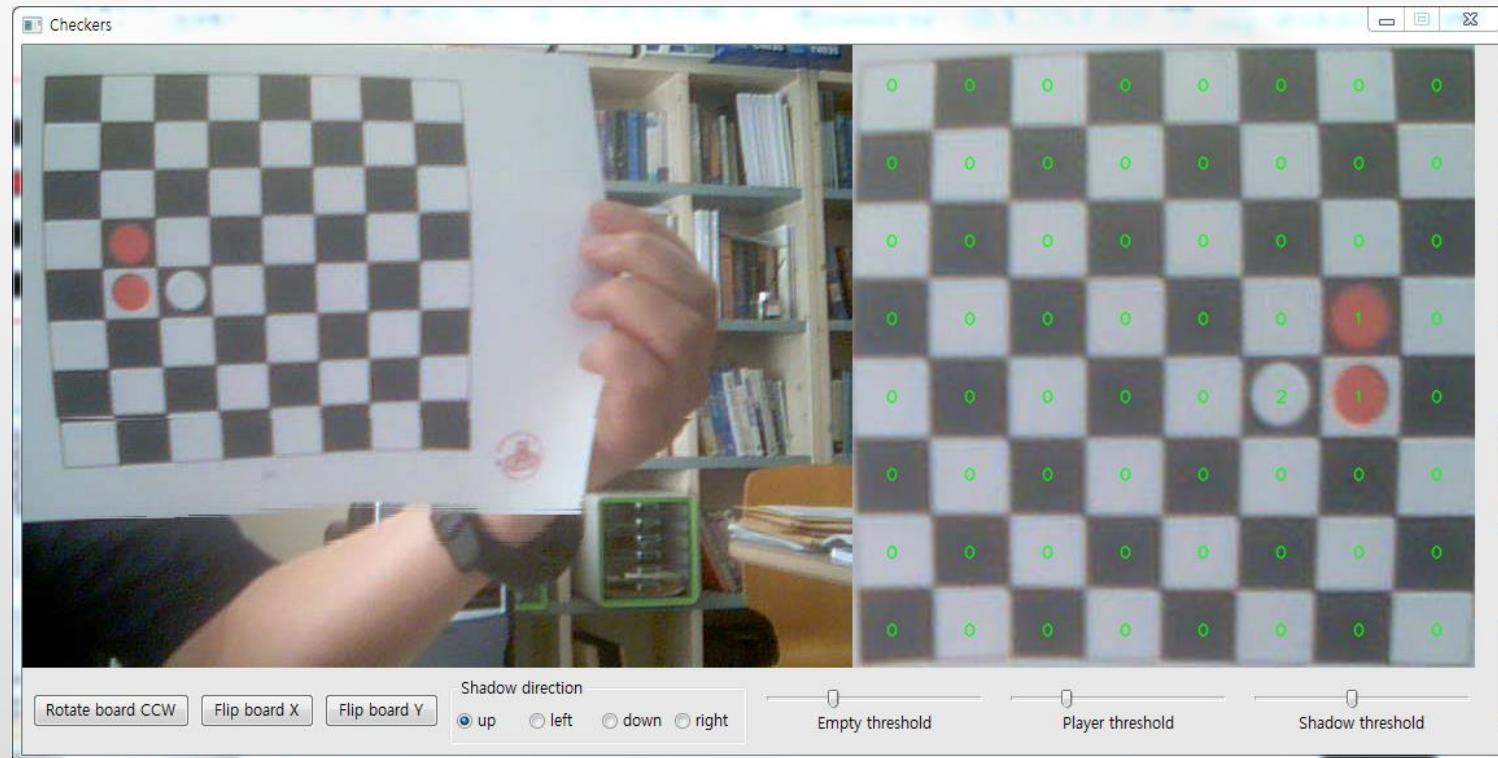
# Checkers (1)



# Checkers (2)



# Checkers (3)



◆ International draughts

➤ 10\*10

◆ Russian, Brazilian, Czech draughts, pool checkers

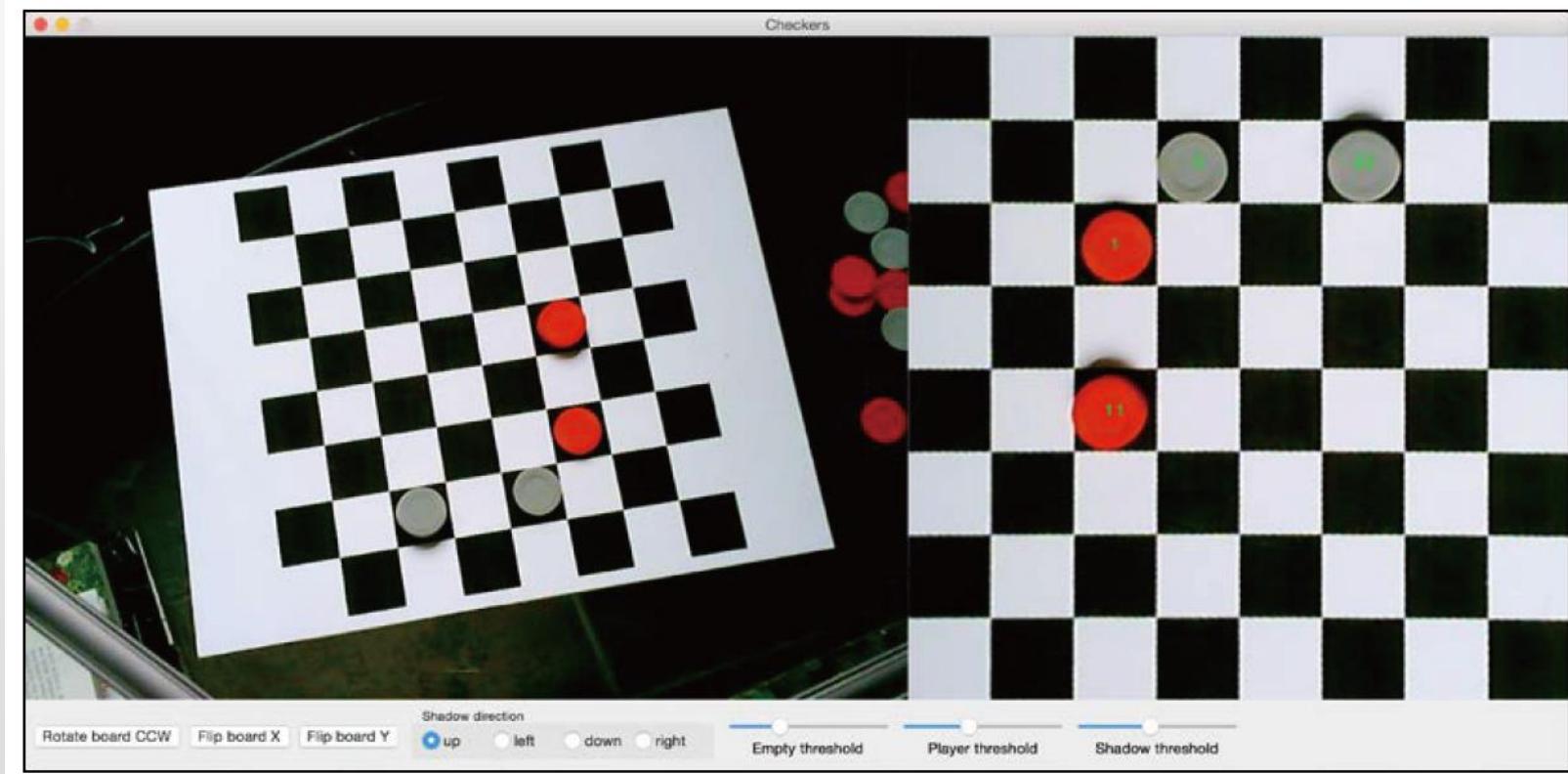
➤ 8\*8

◆ Chips: red and gray

◆ King: 4 chips

◆ Pawn: 2 chips

# Checkers (5)



## # ResizeUtils.py

```
def cvResizeCapture(capture, preferredSize):
 ...
 successW = capture.set(cv2.CAP_PROP_FRAME_WIDTH, w)
 successH = capture.set(cv2.CAP_PROP_FRAME_HEIGHT, h)
 afterW = int(capture.get(cv2.CAP_PROP_FRAME_WIDTH))
 afterH = int(capture.get(cv2.CAP_PROP_FRAME_HEIGHT))
 if afterW == w and afterH == h:#if successW and successH:
 return preferredSize
 ...
 #w = capture.get(cv2.CAP_PROP_FRAME_WIDTH)
 #h = capture.get(cv2.CAP_PROP_FRAME_HEIGHT)
 w = int(capture.get(cv2.CAP_PROP_FRAME_WIDTH))
 h = int(capture.get(cv2.CAP_PROP_FRAME_HEIGHT))
 ...
```

## # Checker.py

```
def cvResizeCapture(capture)
class Checkers(wx.Frame):
 ...
 def _showImage(self, image, staticBitmap, size):
 if image is None:
 #bitmap = wx.EmptyBitmap(size[0], size[1])
 bitmap = wx.Bitmap(size[0], size[1], 3)
 else:
 # Convert the image to bitmap format.
 bitmap = WxUtils.wxBitmapFromCvImage(image)
 # Show the bitmap.
 staticBitmap.SetBitmap(bitmap)
```

## # WxUtils.py

```
if IS_RASPBERRY_PI:
 ...

else:

 def wxBitmapFromCvImage(image):
 if len(image.shape) < 3:
 image = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)
 else:
 image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
 h, w = image.shape[:2]
 #bitmap = wx.BitmapFromBuffer(w, h, image)
 bitmap = wx.Bitmap.FromBuffer(w, h, image)
 return bitmap
```

## # CheckersModel.py

```
class CheckersModel(object):
 ...
 def _drawSquareStatus(self, i, j):

 x0 = i * self._squareWidth
 y0 = j * self._squareWidth

 squareStatus = self._squareStatuses[j, i]
 #if squareStatus > 0:
 if squareStatus >= 0:
```

- ◆ Checker.py
  - GUI using WxPython
- ◆ CheckersModel.py
  - OpenCV, NumPy: capture, analyze, ...
- ◆ WsUtils.py
  - OpenCV → WxPython
- ◆ ResizeUtils.py: image resolution
- ◆ ColorUtils.py: color channel, color distance
- ◆ CVBackwardCompat.py: aliases for OpenCV 2.x

```
import wx

app = wx.App()
frame = wx.Frame(None, title = 'Ex') # parent, title
frame.Show()
app.MainLoop()
```

```
import wx

app = wx.App()
style = wx.CLOSE_BOX | wx.MINIMIZE_BOX | wx.CAPTION | \
 wx.SYSTEM_MENU | wx.CLIP_CHILDREN
frame = wx.Frame(None, title = 'Ex', style = style)
frame.SetBackgroundColour(wx.Colour(200, 255, 200))
frame.Show()
app.MainLoop()
```

```
import cv2

capture = cv2.VideoCapture(0)

while (True):
 ret, frame = capture.read()
 cv2.imshow('Frame', frame)
 if cv2.waitKey(1) & 0xFF == ord('q'):
 break

capture.release()
cv2.destroyAllWindows()
```

```
import cv2
capture = cv2.VideoCapture(0)

preGray = None;
while (True):
 ret, frame = capture.read()
 h, w, c = frame.shape

 frame = cv2.flip(frame, 1)
 gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

```
if preGray is not None:
 for y in range (0, h, 2):
 for x in range (0, w, 2):
 i = 2 #red
 if abs(int(preGray[y][x])-int(gray[y][x])) > 25:
 frame[y][x][i] = 255
 frame[y][x+1][i] = 255
 frame[y+1][x][i] = 255
 frame[y+1][x+1][i] = 255

preGray = gray
cv2.imshow('Frame', frame)
if cv2.waitKey(1) & 0xFF == ord('q'):
 break

capture.release()
cv2.destroyAllWindows()
```

## #CVBackwardCompat.py

```
import cv2

CV_MAJOR_VERSION = int(cv2.__version__.split('.')[0])
if CV_MAJOR_VERSION < 3:
 # Create aliases to make parts of the OpenCV 2.x library
 # forward-compatible.

 cv2.LINE_AA = cv2.CV_AA

 cv2.CAP_PROP_FRAME_WIDTH = cv2.cv.CV_CAP_PROP_FRAME_WIDTH
 cv2.CAP_PROP_FRAME_HEIGHT = cv2.cv.CV_CAP_PROP_FRAME_HEIGHT
 cv2.FILLED = cv2.cv.CV_FILLED
```

## # ColorUtils.py

```
import math
import numpy

def extractChannel(src, channel, dst):
 dstShape = src.shape[:2] # width, height
 if dst is None or dst.shape != dstShape or \
 dst.dtype != numpy.uint8:
 dst = numpy.empty(dstShape, numpy.uint8)
 # channel: 2 → red, increment → 3
 dst[:] = src.flatten()[channel::3].reshape(dstShape)
 return dst
```

```
def colorDist(color0, color1):
 # Calculate a red-weighted color distance, as described
 # here: http://www.compuphase.com/cmetric.htm
 rMean = int((color0[2] + color1[2]) / 2)
 rDiff = int(color0[2] - color1[2])
 gDiff = int(color0[1] - color1[1])
 bDiff = int(color0[0] - color1[0])
 return math.sqrt(
 (((512 + rMean) * rDiff * rDiff) >> 8) +
 4 * gDiff * gDiff +
 (((767 - rMean) * bDiff * bDiff) >> 8))
def normColorDist(color0, color1):
 # Normalize based on the distance between (0, 0, 0) and
 # (255, 255, 255).
 return colorDist(color0, color1) / 764.8333151739665
```

## # ResizeUtils.py

```
from CVBackwardCompat import cv2

def cvResizeCapture(capture, preferredSize):
 w, h = preferredSize
 successW = capture.set(cv2.CAP_PROP_FRAME_WIDTH, w)
 successH = capture.set(cv2.CAP_PROP_FRAME_HEIGHT, h)
 afterW = int(capture.get(cv2.CAP_PROP_FRAME_WIDTH))
 afterH = int(capture.get(cv2.CAP_PROP_FRAME_HEIGHT))

 if afterW == w and afterH == h:
 return preferredSize
 w = int(capture.get(cv2.CAP_PROP_FRAME_WIDTH))
 h = int(capture.get(cv2.CAP_PROP_FRAME_HEIGHT))
 return (w, h)
```

```
from CVBackwardCompat import cv2
import wx

def wxBitmapFromCvImage(image):
 if len(image.shape) < 3:
 image = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)
 else:
 image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
 h, w = image.shape[:2]
 bitmap = wx.Bitmap.FromBuffer(w, h, image)

 return bitmap
```

## # Checkers.py

```
import threading
import wx
import CheckersModel
import WxUtils

class Checkers(wx.Frame):
 def __init__(self, checkersModel, title='Checkers'):

 self._checkersModel = checkersModel

 style = wx.CLOSE_BOX | wx.MINIMIZE_BOX | wx.CAPTION | \
 wx.SYSTEM_MENU | wx.CLIP_CHILDREN
 wx.Frame.__init__(self, None, title=title, style=style)
```

```
self.SetBackgroundColour(wx.Colour(232, 232, 232))
self.Bind(wx.EVT_CLOSE, self._onCloseWindow)

quitCommandID = wx.NewId()
self.Bind(wx.EVT_MENU, self._onQuitCommand,
 id=quitCommandID)
acceleratorTable = wx.AcceleratorTable([
 (wx.ACCEL_NORMAL, wx.WXK_ESCAPE,
 quitCommandID)
])
self.SetAcceleratorTable(acceleratorTable)

self._sceneStaticBitmap = wx.StaticBitmap(self)
self._boardStaticBitmap = wx.StaticBitmap(self)
self._showImages()
```

```
videosSizer = wx.BoxSizer(wx.HORIZONTAL)
videosSizer.Add(self._sceneStaticBitmap)
videosSizer.Add(self._boardStaticBitmap)

rotateBoardButton = wx.Button(
 self, label='Rotate board CCW')
rotateBoardButton.Bind(
 wx.EVT_BUTTON,
 self._onRotateBoardClicked)
flipBoardXButton = wx.Button(
 self, label='Flip board X')
flipBoardXButton.Bind(
 wx.EVT_BUTTON,
 self._onFlipBoardXClicked)
```

```
flipBoardYButton = wx.Button(
 self, label='Flip board Y')
flipBoardYButton.Bind(
 wx.EVT_BUTTON,
 self._onFlipBoardYClicked)

shadowDirectionRadioBox = wx.RadioBox(
 self, label='Shadow direction',
 choices=['up', 'left', 'down', 'right'])
shadowDirectionRadioBox.Bind(
 wx.EVT_RADIOBOX,
 self._onShadowDirectionSelected)
```

```
emptyFreqThresholdSlider = self._createLabeledSlider(
 'Empty threshold',
 self._checkersModel.emptyFreqThreshold * 100,
 self._onEmptyFreqThresholdSelected)

playerDistThresholdSlider = self._createLabeledSlider(
 'Player threshold',
 self._checkersModel.playerDistThreshold * 100,
 self._onPlayerDistThresholdSelected)

shadowDistThresholdSlider = self._createLabeledSlider(
 'Shadow threshold',
 self._checkersModel.shadowDistThreshold * 100,
 self._onShadowDistThresholdSelected)
```

```
controlsStyle = wx.ALIGN_CENTER_VERTICAL | wx.RIGHT
controlsBorder = 8
controlsSizer = wx.BoxSizer(wx.HORIZONTAL)
controlsSizer.Add(rotateBoardButton, 0,
 controlsStyle, controlsBorder)
controlsSizer.Add(flipBoardXButton, 0,
 controlsStyle, controlsBorder)
controlsSizer.Add(flipBoardYButton, 0,
 controlsStyle, controlsBorder)
...
controlsSizer.Add(playerDistThresholdSlider, 0,
 controlsStyle, controlsBorder)
controlsSizer.Add(shadowDistThresholdSlider, 0,
 controlsStyle, controlsBorder)
```

```
rootSizer = wx.BoxSizer(wx.VERTICAL)
rootSizer.Add(videosSizer)
rootSizer.Add(controlsSizer, 0, wx.EXPAND | wx.ALL,
 border=controlsBorder)
self.SetSizerAndFit(rootSizer)

self._captureThread = threading.Thread(# thread
 target=self._runCaptureLoop)
self._running = True
self._captureThread.start() # starting thread
```

```
def _createLabeledSlider(self, label, initialValue,
 callback):

 slider = wx.Slider(self, size=(180, 20))
 slider.SetValue(initialValue)
 slider.Bind(wx.EVT_SLIDER, callback)

 staticText = wx.StaticText(self, label=label)

 sizer = wx.BoxSizer(wx.VERTICAL)
 sizer.Add(slider, 0, wx.ALIGN_CENTER_HORIZONTAL)
 sizer.Add(staticText, 0, wx.ALIGN_CENTER_HORIZONTAL)
 return sizer
```

```
def _onCloseWindow(self, event):
 self._running = False # closing thread
 self.captureThread.join() # wait until thread has finished
 self.Destroy()
def _onQuitCommand(self, event):
 self.Close()
def _onRotateBoardClicked(self, event):
 self._checkersModel.boardRotation += 1
def _onFlipBoardXClicked(self, event):
 self._checkersModel.flipBoardX = \
 not self._checkersModel.flipBoardX
def _onFlipBoardYClicked(self, event):
 self._checkersModel.flipBoardY = \
 not self._checkersModel.flipBoardY
```

```
def _onShadowDirectionSelected(self, event):
 self._checkersModel.shadowDirection = event.Selection
def _onEmptyFreqThresholdSelected(self, event):
 self._checkersModel.emptyFreqThreshold = \
 event.Selection * 0.01
def _onPlayerDistThresholdSelected(self, event):
 self._checkersModel.playerDistThreshold = \
 event.Selection * 0.01
def _onShadowDistThresholdSelected(self, event):
 self._checkersModel.shadowDistThreshold = \
 event.Selection * 0.01
def _runCaptureLoop(self): # thread function
 while self._running:
 if self._checkersModel.update():
 wx.CallAfter(self._showImages)
 # call after the current/pending event handlers
 # have been completed
```

```
def _showImages(self):
 self._showImage(
 self._checkersModel.scene, self._sceneStaticBitmap,
 self._checkersModel.sceneSize)
 self._showImage(
 self._checkersModel.board, self._boardStaticBitmap,
 self._checkersModel.boardSize)
def _showImage(self, image, staticBitmap, size):
 if image is None:
 bitmap = wx.Bitmap(size[0], size[1], 3)
 else:
 bitmap = WxUtils.wxBitmapFromCvImage(image)
 # Show the bitmap.
 staticBitmap.SetBitmap(bitmap)
```

```
def main():
 import sys
 if len(sys.argv) < 2:
 cameraDeviceID = 0
 else:
 cameraDeviceID = int(sys.argv[1])
 checkersModel = CheckersModel.CheckersModel(
 cameraDeviceID=cameraDeviceID)
 app = wx.App()
 checkers = Checkers(checkersModel)
 checkers.Show()
 app.MainLoop()
if __name__ == '__main__':
 main()
```

## # CheckersModel.py

```
import numpy
import sklearn.cluster
from CVBackwardCompat import cv2
import ColorUtils
import ResizeUtils

ROTATION_0 = 0
ROTATION_CCW_90 = 1
ROTATION_180 = 2
ROTATION_CCW_270 = 3

DIRECTION_UP = 0
DIRECTION_LEFT = 1
DIRECTION_DOWN = 2
DIRECTION_RIGHT = 3
```

```
SQUARE_STATUS_UNKNOWN = -1
SQUARE_STATUS_EMPTY = 0
SQUARE_STATUS_PAWN_PLAYER_1 = 1
SQUARE_STATUS_KING_PLAYER_1 = 11
SQUARE_STATUS_PAWN_PLAYER_2 = 2
SQUARE_STATUS_KING_PLAYER_2 = 22
```

```
class CheckersModel(object):
 @property
 def sceneSize(self):
 return self._sceneSize
```

```
 @property
 def scene(self):
 return self._scene
```

```
@property
def sceneGray(self):
 return self._sceneGray

@property
def boardSize(self):
 return self._boardSize

@property
def board(self):
 return self._board

@property
def squareStatuses(self):
 return self._squareStatuses

@property
def boardRotation(self):
 return self._boardRotation
```

```
@boardRotation.setter
def boardRotation(self, value):
 self._boardRotation = value % 4
@property
def shadowDirection(self):
 return self._shadowDirection
@shadowDirection.setter
def shadowDirection(self, value):
 self._shadowDirection = value % 4

def __init__(self, patternSize=(7, 7), cameraDeviceID=0,
 sceneSize=(800, 600)):
 self.emptyFreqThreshold = 0.3
 self.playerDistThreshold = 0.4
 self.shadowDistThreshold = 0.45
```

```
self._boardRotation = ROTATION_0
self.flipBoardX = False
self.flipBoardY = False
self._shadowDirection = DIRECTION_UP

self._scene = None
self._sceneGray = None
self._board = None

self._patternSize = patternSize
self._numCorners = patternSize[0] * patternSize[1]
```

```
self._squareFreqs = numpy.empty(
 (patternSize[1] + 1, patternSize[0] + 1),
 numpy.float32)

self._squareDists = numpy.empty(
 (patternSize[1] + 1, patternSize[0] + 1),
 numpy.float32)

self._squareStatuses = numpy.empty(
 (patternSize[1] + 1, patternSize[0] + 1),
 numpy.int8)

self._squareStatuses.fill(SQUARE_STATUS_UNKNOWN)

cluster initializing by k-means (k=2)
self._clusterer = sklearn.cluster.MiniBatchKMeans(2)
```

```
self._capture = cv2.VideoCapture(cameraDeviceID)
self._sceneSize = ResizeUtils.cvResizeCapture(
 self._capture, sceneSize)
w, h = self._sceneSize

self._lastCorners = None
self._lastCornersSceneGray = numpy.empty(
 (h, w), numpy.uint8)
self._boardHomography = None

self._squareWidth = min(w, h) // (max(patternSize) + 1)
self._squareArea = self._squareWidth ** 2
```

```
self._boardSize = (
 (patternSize[0] + 1) * self._squareWidth,
 (patternSize[1] + 1) * self._squareWidth
)

reference corners to find Homography
self._referenceCorners = []
for x in range(patternSize[0]):
 for y in range(patternSize[1]):
 self._referenceCorners += [[x, y]]
self._referenceCorners = numpy.array(
 self._referenceCorners, numpy.float32)
self._referenceCorners *= self._squareWidth
self._referenceCorners += self._squareWidth
```

```
def update(self, drawCorners=False,
 drawSquareStatuses=True):

 if not self._updateScene():
 return False # Failure

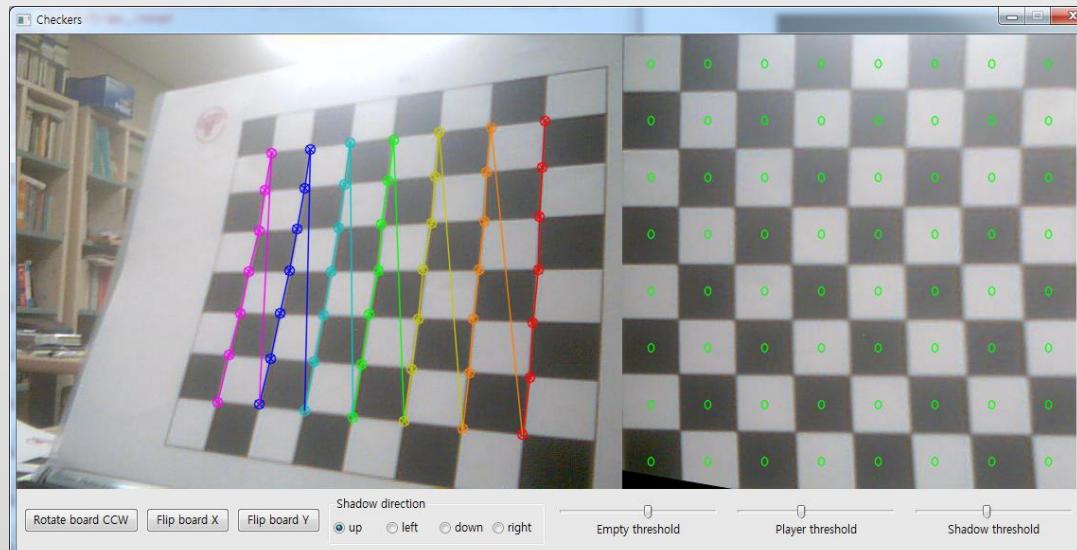
 self._updateBoardHomography()
 self._updateBoard()

 if drawCorners and self._lastCorners is not None:
 # Draw the board's grid.
 cv2.drawChessboardCorners(
 self._scene, self._patternSize,
 self._lastCorners, True)
```

# Computer Vision (10)

```
def update(self, drawCorners=True,
 drawSquareStatuses=True):

def _runCaptureLoop(self): # Checkers.py
 while self._running:
 if self._checkersModel.update():
 #wx.CallAfter(self._showImages)
 self._showImages()
```



```
if drawSquareStatuses:
 for i in range(self._patternSize[0] + 1):
 for j in range(self._patternSize[1] + 1):
 self._drawSquareStatus(i, j)
return True # Success

def _updateScene(self):
 success, self._scene = self._capture.read(self._scene)
 if not success:
 return False # Failure
 # Use the red channel as grayscale.
 self._sceneGray = ColorUtils.extractChannel(
 self._scene, 2, self._sceneGray)

return True # Success
```

```
def _updateBoardHomography(self):
 if self._lastCorners is not None:
 corners, status, error = cv2.calcOpticalFlowPyrLK(
 self._lastCornersSceneGray,
 self._sceneGray, self._lastCorners, None)
 # Status is 1 if tracked, 0 if not tracked.
 numCornersTracked = sum(status)
 # If not tracked, error is invalid so set it to 0.
 error[:] = error * status
 meanError = (sum(error) / numCornersTracked)[0]
 if meanError < 4.0: # uses corners by tracking
 # The board's old corners and homography are
 # still good enough.
 return
```

```
Find the corners in the board's grid.
cornersWereFound, corners = cv2.findChessboardCorners(
 self._sceneGray, self._patternSize,
 flags=cv2.CALIB_CB_ADAPTIVE_THRESH)
if cornersWereFound:
 # Find the homography.
 corners = numpy.array(corners, numpy.float32)
 corners = corners.reshape(self._numCorners, 2)
 self._boardHomography, matches = \
 cv2.findHomography(\
 corners, self._referenceCorners, cv2.RANSAC)
 # Record the corners and their image.
 self._lastCorners = corners
 self._lastCornersSceneGray[:] = self._sceneGray
```

```
def _updateBoard(self):
 if self._boardHomography is not None:
 # Warp the board to obtain a bird's-eye view.
 self._board = cv2.warpPerspective(# perspective by H
 self._scene, self._boardHomography,
 self._boardSize, self._board)
 # Rotate and flip the board.
 flipX = self.flipBoardX
 flipY = self.flipBoardY
 if self._boardRotation == ROTATION_CCW_90:
 cv2.transpose(self._board, self._board)
 flipX = not flipX
```

```
elif self._boardRotation == ROTATION_180:
 flipX = not flipX
 flipY = not flipY
elif self._boardRotation == ROTATION_CCW_270:
 cv2.transpose(self._board, self._board)
 flipY = not flipY
if flipX:
 if flipY:
 cv2.flip(self._board, -1, self._board)
 else:
 cv2.flip(self._board, 1, self._board)
elif flipY:
 cv2.flip(self._board, 0, self._board)
```

```
for i in range(self._patternSize[0] + 1):
 for j in range(self._patternSize[1] + 1):
 self._updateSquareData(i, j)
for i in range(self._patternSize[0] + 1):
 for j in range(self._patternSize[1] + 1):
 self._updateSquareStatus(i, j)
def _updateSquareData(self, i, j):
 x0 = i * self._squareWidth
 x1 = x0 + self._squareWidth
 y0 = j * self._squareWidth
 y1 = y0 + self._squareWidth
 # Find the two dominant colors in the square.
 self._clusterer.fit(\ # clustering
 self._board[y0:y1, x0:x1].reshape(
 self._squareArea, 3))
```

```
Find the proportion of the square's area that is
occupied by the less dominant color.
mean for 0, 1 label, complexity: freq
freq = numpy.mean(self._clusterer.labels_)
if freq > 0.5:
 freq = 1.0 - freq

Find the distance between the dominant colors.
dist = ColorUtils.normColorDist(
 self._clusterer.cluster_centers_[0],
 self._clusterer.cluster_centers_[1])

self._squareFreqs[j, i] = freq
self._squareDists[j, i] = dist
```

```
def _updateSquareStatus(self, i, j):
 freq = self._squareFreqs[j, i]
 dist = self._squareDists[j, i]
 if self._shadowDirection == DIRECTION_UP:
 if j > 0:
 neighborFreq = self._squareFreqs[j - 1, i]
 neighborDist = self._squareDists[j - 1, i]
 else:
 neighborFreq = None
 neighborDist = None
```

```
elif self._shadowDirection == DIRECTION_LEFT:
 if i > 0:
 neighborFreq = self._squareFreqs[j, i - 1]
 neighborDist = self._squareDists[j, i - 1]
 else:
 neighborFreq = None
 neighborDist = None
elif self._shadowDirection == DIRECTION_DOWN:
 if j < self._patternSize[1]:
 neighborFreq = self._squareFreqs[j + 1, i]
 neighborDist = self._squareDists[j + 1, i]
 else:
 neighborFreq = None
 neighborDist = None
```

```
elif self._shadowDirection == DIRECTION_RIGHT:
 if i < self._patternSize[0]:
 neighborFreq = self._squareFreqs[j, i + 1]
 neighborDist = self._squareDists[j, i + 1]
 else:
 neighborFreq = None
 neighborDist = None
else:
 neighborFreq = None
 neighborDist = None
castsShadow = \
 neighborFreq is not None and \
 neighborFreq < self.emptyFreqThreshold and \
 neighborDist is not None and \
 neighborDist > self.shadowDistThreshold
```

```
if freq < self.emptyFreqThreshold:
 squareStatus = SQUARE_STATUS_EMPTY
else:
 if dist < self.playerDistThreshold:
 if castsShadow:
 squareStatus = SQUARE_STATUS_KING_PLAYER_1
 else:
 squareStatus = SQUARE_STATUS_PAWN_PLAYER_1
 else:
 if castsShadow:
 squareStatus = SQUARE_STATUS_KING_PLAYER_2
 else:
 squareStatus = SQUARE_STATUS_PAWN_PLAYER_2
self._squareStatuses[j, i] = squareStatus
```

```
def _drawSquareStatus(self, i, j):
 x0 = i * self._squareWidth
 y0 = j * self._squareWidth
 squareStatus = self._squareStatuses[j, i]
 if squareStatus >= 0:
 text = str(squareStatus)
 textSize, textBaseline = cv2.getTextSize(
 text, cv2.FONT_HERSHEY_PLAIN, 1.0, 1)
 xCenter = x0 + self._squareWidth // 2
 yCenter = y0 + self._squareWidth // 2
 textCenter = (xCenter - textSize[0] // 2,
 yCenter + textBaseline)
 cv2.putText(self._board, text, textCenter,
 cv2.FONT_HERSHEY_PLAIN, 1.0,
 (0, 255, 0), 1, cv2.LINE_AA)
```