|  |  |  |  |
| --- | --- | --- | --- |
| 工程经济学游戏——工程人马里奥 | | | |
| 名称 | 工程经济学游戏——工程人马里奥 | 游戏类型 | 冒险，益智，闯关 |
| 日期 | 2022年10月18日 | 版本号 | V2.0.0 |
| 文档大纲 | | | |
| **代码目录与结构** | | | |
| 策划历史记录 | | | |
| V1.0.0 | 最简单的“刷题”系统，只能是选择题，没有融入游戏 | | 2021年 |
| V1.5.0 | 融入了“俄罗斯方块”游戏元素，但玩法仍然单一 | | 2022.6月 |
| V2.0.0 | 拓展为 “工程人马里奥”，玩家和开发者有更多的选择 | | 2022.10.18 |
| 文档约定 | | | |
| 黑色 | 已确定内容 | | |
| 橙色 | 融入的工程经济学的内容 | | |
| 红色 | 重点 | | |
| 蓝色 | 最新版本更新内容 | | |
| 灰色 | 删除/暂时不做的内容 | | |
| *斜体* | *解释说明的内容* | | |

**目 录**

[1 main.py 1](#_Toc117010464)

[2 source 1](#_Toc117010465)

[2.1 source/components 1](#_Toc117010466)

[2.1.1 box.py 1](#_Toc117010467)

[2.1.2 brick.py 3](#_Toc117010468)

[2.1.3 coin.py 6](#_Toc117010469)

[2.1.4 enemy.py 8](#_Toc117010470)

[2.1.5 info.py 18](#_Toc117010471)

[2.1.6 player.py 22](#_Toc117010472)

[2.1.7 powerup.py 34](#_Toc117010473)

[2.1.8 stuff.py 40](#_Toc117010474)

[2.2 source/data 44](#_Toc117010475)

[2.2.1 source/data/maps 44](#_Toc117010476)

[2.2.2 source/data/player 112](#_Toc117010477)

[2.3 source/states 117](#_Toc117010478)

[2.3.1 level.py 117](#_Toc117010479)

[2.3.2 load\_screen.py 130](#_Toc117010480)

[2.3.3 main\_menu.py 132](#_Toc117010481)

[2.4 constants.py 135](#_Toc117010482)

[2.5 main.py 139](#_Toc117010483)

[2.6 setup.py 139](#_Toc117010484)

[2.7 tools.py 139](#_Toc117010485)

[3 resource 142](#_Toc117010486)

[3.1 resource/Graphics 142](#_Toc117010487)

[3.2 resource/QuestionProj 142](#_Toc117010488)

[3.3 resource/Sound 142](#_Toc117010489)

# main.py

import pygame as pg

from source.main import main

if \_\_name\_\_ == '\_\_main\_\_':

main()

pg.quit()

# source

## source/components

### box.py

import pygame as pg

from . import coin, powerup

from .. import constants as c

from .. import setup, tools

class Box(pg.sprite.Sprite):

def \_\_init\_\_(self, x, y, type, group=None, name=c.MAP\_BOX):

pg.sprite.Sprite.\_\_init\_\_(self)

self.frames = []

self.frame\_index = 0

self.load\_frames()

self.image = self.frames[self.frame\_index]

self.rect = self.image.get\_rect()

self.rect.x = x

self.rect.y = y

self.rest\_height = y

self.animation\_timer = 0

self.first\_half = True # First half of animation cycle

self.state = c.RESTING

self.y\_vel = 0

self.gravity = 1.2

self.type = type

self.group = group

self.name = name

def load\_frames(self):

sheet = setup.GFX['tile\_set']

frame\_rect\_list = [(384, 0, 16, 16), (400, 0, 16, 16),

(416, 0, 16, 16), (400, 0, 16, 16), (432, 0, 16, 16)]

for frame\_rect in frame\_rect\_list:

self.frames.append(tools.get\_image(sheet, \*frame\_rect,

c.BLACK, c.BRICK\_SIZE\_MULTIPLIER))

def update(self, game\_info):

self.current\_time = game\_info[c.CURRENT\_TIME]

if self.state == c.RESTING:

self.resting()

elif self.state == c.BUMPED:

self.bumped()

def resting(self):

time\_list = [375, 125, 125, 125]

if (self.current\_time - self.animation\_timer) > time\_list[self.frame\_index]:

self.frame\_index += 1

if self.frame\_index == 4:

self.frame\_index = 0

self.animation\_timer = self.current\_time

self.image = self.frames[self.frame\_index]

def bumped(self):

self.rect.y += self.y\_vel

self.y\_vel += self.gravity

if self.rect.y > self.rest\_height + 5:

self.rect.y = self.rest\_height

self.state = c.OPENED

if self.type == c.TYPE\_MUSHROOM:

self.group.add(powerup.Mushroom(self.rect.centerx, self.rect.y))

elif self.type == c.TYPE\_FIREFLOWER:

self.group.add(powerup.FireFlower(self.rect.centerx, self.rect.y))

elif self.type == c.TYPE\_LIFEMUSHROOM:

self.group.add(powerup.LifeMushroom(self.rect.centerx, self.rect.y))

elif self.type == c.EGGSHELL\_PROJ\_NUM[0]:

self.group.add(powerup.gong(self.rect.centerx, self.rect.y))

elif self.type == c.EGGSHELL\_PROJ\_NUM[1]:

self.group.add(powerup.cheng(self.rect.centerx, self.rect.y))

elif self.type == c.EGGSHELL\_PROJ\_NUM[2]:

self.group.add(powerup.jing(self.rect.centerx, self.rect.y))

elif self.type == c.EGGSHELL\_PROJ\_NUM[3]:

self.group.add(powerup.ji(self.rect.centerx, self.rect.y))

elif self.type == c.EGGSHELL\_PROJ\_NUM[4]:

self.group.add(powerup.xue(self.rect.centerx, self.rect.y))

elif self.type == c.JUDGMENT\_PROJ\_NUM[0]:

self.group.add(powerup.dui(self.rect.centerx, self.rect.y))

elif self.type == c.JUDGMENT\_PROJ\_NUM[1]:

self.group.add(powerup.cuo(self.rect.centerx, self.rect.y))

# todo: add more powerups

self.frame\_index = 4

self.image = self.frames[self.frame\_index]

def start\_bump(self, score\_group):

self.y\_vel = -6

self.state = c.BUMPED

if self.type == c.TYPE\_COIN:

self.group.add(coin.Coin(self.rect.centerx, self.rect.y, score\_group))

### brick.py

from . import coin, stuff, powerup

from .. import constants as c

from .. import setup

def create\_brick(brick\_group, item, level):

if c.COLOR in item:

color = item[c.COLOR]

else:

color = c.COLOR\_TYPE\_ORANGE

x, y, type = item['x'], item['y'], item['type']

if type == c.TYPE\_COIN:

brick\_group.add(Brick(x, y, type,

color, level.coin\_group))

elif (type == c.TYPE\_STAR or

type == c.TYPE\_FIREFLOWER or

type == c.TYPE\_LIFEMUSHROOM):

brick\_group.add(Brick(x, y, type,

color, level.powerup\_group))

else:

if c.BRICK\_NUM in item:

create\_brick\_list(brick\_group, item[c.BRICK\_NUM], x, y, type,

color, item['direction'])

else:

brick\_group.add(Brick(x, y, type, color))

def create\_brick\_list(brick\_group, num, x, y, type, color, direction):

''' direction:horizontal, create brick from left to right, direction:vertical, create brick from up to bottom '''

size = 43 # 16 \* c.BRICK\_SIZE\_MULTIPLIER is 43

tmp\_x, tmp\_y = x, y

for i in range(num):

if direction == c.VERTICAL:

tmp\_y = y + i \* size

else:

tmp\_x = x + i \* size

brick\_group.add(Brick(tmp\_x, tmp\_y, type, color))

class Brick(stuff.Stuff):

def \_\_init\_\_(self, x, y, type, color=c.ORANGE, group=None, name=c.MAP\_BRICK):

orange\_rect = [(16, 0, 16, 16), (432, 0, 16, 16)]

green\_rect = [(208, 32, 16, 16), (48, 32, 16, 16)]

if color == c.COLOR\_TYPE\_ORANGE:

frame\_rect = orange\_rect

else:

frame\_rect = green\_rect

stuff.Stuff.\_\_init\_\_(self, x, y, setup.GFX['tile\_set'],

frame\_rect, c.BRICK\_SIZE\_MULTIPLIER)

self.rest\_height = y

self.state = c.RESTING

self.y\_vel = 0

self.gravity = 1.2

self.type = type

if self.type == c.TYPE\_COIN:

self.coin\_num = 10

else:

self.coin\_num = 0

self.group = group

self.name = name

def update(self):

if self.state == c.BUMPED:

self.bumped()

def bumped(self):

self.rect.y += self.y\_vel

self.y\_vel += self.gravity

if self.rect.y >= self.rest\_height:

self.rect.y = self.rest\_height

if self.type == c.TYPE\_COIN:

if self.coin\_num > 0:

self.state = c.RESTING

else:

self.state = c.OPENED

elif self.type == c.TYPE\_STAR:

self.state = c.OPENED

self.group.add(powerup.Star(self.rect.centerx, self.rest\_height))

elif self.type == c.TYPE\_FIREFLOWER:

self.state = c.OPENED

self.group.add(powerup.FireFlower(self.rect.centerx, self.rest\_height))

elif self.type == c.TYPE\_LIFEMUSHROOM:

self.state = c.OPENED

self.group.add(powerup.LifeMushroom(self.rect.centerx, self.rest\_height))

else:

self.state = c.RESTING

def start\_bump(self, score\_group):

self.y\_vel -= 7

if self.type == c.TYPE\_COIN:

if self.coin\_num > 0:

self.group.add(coin.Coin(self.rect.centerx, self.rect.y, score\_group))

self.coin\_num -= 1

if self.coin\_num == 0:

self.frame\_index = 1

self.image = self.frames[self.frame\_index]

elif (self.type == c.TYPE\_STAR or

self.type == c.TYPE\_FIREFLOWER or

self.type == c.TYPE\_LIFEMUSHROOM):

self.frame\_index = 1

self.image = self.frames[self.frame\_index]

self.state = c.BUMPED

def change\_to\_piece(self, group):

arg\_list = [(self.rect.x, self.rect.y - (self.rect.height / 2), -2, -12),

(self.rect.right, self.rect.y - (self.rect.height / 2), 2, -12),

(self.rect.x, self.rect.y, -2, -6),

(self.rect.right, self.rect.y, 2, -6)]

for arg in arg\_list:

group.add(BrickPiece(\*arg))

self.kill()

class BrickPiece(stuff.Stuff):

def \_\_init\_\_(self, x, y, x\_vel, y\_vel):

stuff.Stuff.\_\_init\_\_(self, x, y, setup.GFX['tile\_set'],

[(68, 20, 8, 8)], c.BRICK\_SIZE\_MULTIPLIER)

self.x\_vel = x\_vel

self.y\_vel = y\_vel

self.gravity = .8

def update(self, \*args):

self.rect.x += self.x\_vel

self.rect.y += self.y\_vel

self.y\_vel += self.gravity

if self.rect.y > c.SCREEN\_HEIGHT:

self.kill()

### coin.py

import pygame as pg

from .. import constants as c

from .. import setup, tools

class Coin(pg.sprite.Sprite):

def \_\_init\_\_(self, x, y, score\_group):

pg.sprite.Sprite.\_\_init\_\_(self)

self.frames = []

self.frame\_index = 0

self.load\_frames()

self.image = self.frames[self.frame\_index]

self.rect = self.image.get\_rect()

self.rect.centerx = x

self.rect.bottom = y - 5

self.gravity = 1

self.y\_vel = -15

self.animation\_timer = 0

self.initial\_height = self.rect.bottom - 5

self.score\_group = score\_group

def load\_frames(self):

sheet = setup.GFX[c.ITEM\_SHEET]

frame\_rect\_list = [(52, 113, 8, 14), (4, 113, 8, 14),

(20, 113, 8, 14), (36, 113, 8, 14)]

for frame\_rect in frame\_rect\_list:

self.frames.append(tools.get\_image(sheet, \*frame\_rect,

c.BLACK, c.BRICK\_SIZE\_MULTIPLIER))

def update(self, game\_info):

self.current\_time = game\_info[c.CURRENT\_TIME]

self.spinning()

def spinning(self):

self.image = self.frames[self.frame\_index]

self.rect.y += self.y\_vel

self.y\_vel += self.gravity

if (self.current\_time - self.animation\_timer) > 80:

if self.frame\_index < 3:

self.frame\_index += 1

else:

self.frame\_index = 0

self.animation\_timer = self.current\_time

if self.rect.bottom > self.initial\_height:

self.kill()

class FlashCoin(pg.sprite.Sprite):

def \_\_init\_\_(self, x, y):

pg.sprite.Sprite.\_\_init\_\_(self)

self.frame\_index = 0

self.frames = []

self.load\_frames()

self.image = self.frames[self.frame\_index]

self.rect = self.image.get\_rect()

self.rect.x = x

self.rect.y = y

self.animation\_timer = 0

def load\_frames(self):

sheet = setup.GFX[c.ITEM\_SHEET]

frame\_rect\_list = [(1, 160, 5, 8), (9, 160, 5, 8),

(17, 160, 5, 8), (9, 160, 5, 8)]

for frame\_rect in frame\_rect\_list:

self.frames.append(tools.get\_image(sheet, \*frame\_rect,

c.BLACK, c.BRICK\_SIZE\_MULTIPLIER))

def update(self, current\_time):

time\_list = [375, 125, 125, 125]

if self.animation\_timer == 0:

self.animation\_timer = current\_time

elif (current\_time - self.animation\_timer) > time\_list[self.frame\_index]:

self.frame\_index += 1

if self.frame\_index == 4:

self.frame\_index = 0

self.animation\_timer = current\_time

self.image = self.frames[self.frame\_index]

class StaticCoin(pg.sprite.Sprite):

def \_\_init\_\_(self, x, y):

pg.sprite.Sprite.\_\_init\_\_(self)

self.frame\_index = 0

self.frames = []

self.load\_frames()

self.image = self.frames[self.frame\_index]

self.rect = self.image.get\_rect()

self.rect.x = x

self.rect.y = y

self.animation\_timer = 0

def load\_frames(self):

sheet = setup.GFX[c.ITEM\_SHEET]

frame\_rect\_list = [(3, 98, 9, 13), (19, 98, 9, 13),

(35, 98, 9, 13), (51, 98, 9, 13)]

for frame\_rect in frame\_rect\_list:

self.frames.append(tools.get\_image(sheet, \*frame\_rect,

c.BLACK, c.BRICK\_SIZE\_MULTIPLIER))

def update(self, game\_info):

self.current\_time = game\_info[c.CURRENT\_TIME]

time\_list = [375, 125, 125, 125]

if self.animation\_timer == 0:

self.animation\_timer = self.current\_time

elif (self.current\_time - self.animation\_timer) > time\_list[self.frame\_index]:

self.frame\_index += 1

if self.frame\_index == 4:

self.frame\_index = 0

self.animation\_timer = self.current\_time

self.image = self.frames[self.frame\_index]

### enemy.py

import math

import pygame as pg

from .. import constants as c

from .. import setup, tools

ENEMY\_SPEED = 1

def create\_enemy(item, level):

dir = c.LEFT if item['direction'] == 0 else c.RIGHT

color = item[c.COLOR]

if c.ENEMY\_RANGE in item:

in\_range = item[c.ENEMY\_RANGE]

range\_start = item['range\_start']

range\_end = item['range\_end']

else:

in\_range = False

range\_start = range\_end = 0

if item['type'] == c.ENEMY\_TYPE\_GOOMBA:

sprite = Goomba(item['x'], item['y'], dir, color,

in\_range, range\_start, range\_end)

elif item['type'] == c.ENEMY\_TYPE\_KOOPA:

sprite = Koopa(item['x'], item['y'], dir, color,

in\_range, range\_start, range\_end)

elif item['type'] == c.ENEMY\_TYPE\_FLY\_KOOPA:

isVertical = False if item['is\_vertical'] == 0 else True

sprite = FlyKoopa(item['x'], item['y'], dir, color,

in\_range, range\_start, range\_end, isVertical)

elif item['type'] == c.ENEMY\_TYPE\_PIRANHA:

sprite = Piranha(item['x'], item['y'], dir, color,

in\_range, range\_start, range\_end)

elif item['type'] == c.ENEMY\_TYPE\_FIRE\_KOOPA:

sprite = FireKoopa(item['x'], item['y'], dir, color,

in\_range, range\_start, range\_end, level)

elif item['type'] == c.ENEMY\_TYPE\_FIRESTICK:

'''use a number of fireballs to stimulate a firestick'''

sprite = []

num = item['num']

center\_x, center\_y = item['x'], item['y']

for i in range(num):

radius = i \* 21 # 8 \* 2.69 = 21

sprite.append(FireStick(center\_x, center\_y, dir, color,

radius))

return sprite

class Enemy(pg.sprite.Sprite):

def \_\_init\_\_(self):

pg.sprite.Sprite.\_\_init\_\_(self)

def setup\_enemy(self, x, y, direction, name, sheet, frame\_rect\_list,

in\_range, range\_start, range\_end, isVertical=False):

self.frames = []

self.frame\_index = 0

self.animate\_timer = 0

self.gravity = 1.5

self.state = c.WALK

self.name = name

self.direction = direction

self.load\_frames(sheet, frame\_rect\_list)

self.image = self.frames[self.frame\_index]

self.rect = self.image.get\_rect()

self.rect.x = x

self.rect.bottom = y

self.in\_range = in\_range

self.range\_start = range\_start

self.range\_end = range\_end

self.isVertical = isVertical

self.set\_velocity()

self.death\_timer = 0

def load\_frames(self, sheet, frame\_rect\_list):

for frame\_rect in frame\_rect\_list:

self.frames.append(tools.get\_image(sheet, \*frame\_rect,

c.BLACK, c.SIZE\_MULTIPLIER))

def set\_velocity(self):

if self.isVertical:

self.x\_vel = 0

self.y\_vel = ENEMY\_SPEED

else:

self.x\_vel = ENEMY\_SPEED \* -1 if self.direction == c.LEFT else ENEMY\_SPEED

self.y\_vel = 0

def update(self, game\_info, level):

self.current\_time = game\_info[c.CURRENT\_TIME]

self.handle\_state()

self.animation()

self.update\_position(level)

def handle\_state(self):

if (self.state == c.WALK or

self.state == c.FLY):

self.walking()

elif self.state == c.FALL:

self.falling()

elif self.state == c.JUMPED\_ON:

self.jumped\_on()

elif self.state == c.DEATH\_JUMP:

self.death\_jumping()

elif self.state == c.SHELL\_SLIDE:

self.shell\_sliding()

elif self.state == c.REVEAL:

self.revealing()

def walking(self):

if (self.current\_time - self.animate\_timer) > 125:

if self.direction == c.RIGHT:

if self.frame\_index == 4:

self.frame\_index += 1

elif self.frame\_index == 5:

self.frame\_index = 4

else:

if self.frame\_index == 0:

self.frame\_index += 1

elif self.frame\_index == 1:

self.frame\_index = 0

self.animate\_timer = self.current\_time

def falling(self):

if self.y\_vel < 10:

self.y\_vel += self.gravity

def jumped\_on(self):

pass

def death\_jumping(self):

self.rect.y += self.y\_vel

self.rect.x += self.x\_vel

self.y\_vel += self.gravity

if self.rect.y > c.SCREEN\_HEIGHT:

self.kill()

def shell\_sliding(self):

if self.direction == c.RIGHT:

self.x\_vel = 10

else:

self.x\_vel = -10

def revealing(self):

pass

def start\_death\_jump(self, direction):

self.y\_vel = -8

self.x\_vel = 2 if direction == c.RIGHT else -2

self.gravity = .5

self.frame\_index = 3

self.state = c.DEATH\_JUMP

def animation(self):

self.image = self.frames[self.frame\_index]

def update\_position(self, level):

self.rect.x += self.x\_vel

self.check\_x\_collisions(level)

if self.in\_range and self.isVertical:

if self.rect.y < self.range\_start:

self.rect.y = self.range\_start

self.y\_vel = ENEMY\_SPEED

elif self.rect.bottom > self.range\_end:

self.rect.bottom = self.range\_end

self.y\_vel = -1 \* ENEMY\_SPEED

self.rect.y += self.y\_vel

if (self.state != c.DEATH\_JUMP and

self.state != c.FLY):

self.check\_y\_collisions(level)

if self.rect.x <= 0:

self.kill()

elif self.rect.y > (level.viewport.bottom):

self.kill()

def check\_x\_collisions(self, level):

if self.in\_range and not self.isVertical:

if self.rect.x < self.range\_start:

self.rect.x = self.range\_start

self.change\_direction(c.RIGHT)

elif self.rect.right > self.range\_end:

self.rect.right = self.range\_end

self.change\_direction(c.LEFT)

else:

collider = pg.sprite.spritecollideany(self, level.ground\_step\_pipe\_group)

if collider:

if self.direction == c.RIGHT:

self.rect.right = collider.rect.left

self.change\_direction(c.LEFT)

elif self.direction == c.LEFT:

self.rect.left = collider.rect.right

self.change\_direction(c.RIGHT)

if self.state == c.SHELL\_SLIDE:

enemy = pg.sprite.spritecollideany(self, level.enemy\_group)

if enemy:

level.update\_score(100, enemy, 0)

level.move\_to\_dying\_group(level.enemy\_group, enemy)

enemy.start\_death\_jump(self.direction)

def change\_direction(self, direction):

self.direction = direction

if self.direction == c.RIGHT:

self.x\_vel = ENEMY\_SPEED

if self.state == c.WALK or self.state == c.FLY:

self.frame\_index = 4

else:

self.x\_vel = ENEMY\_SPEED \* -1

if self.state == c.WALK or self.state == c.FLY:

self.frame\_index = 0

def check\_y\_collisions(self, level):

# decrease runtime delay: when enemey is on the ground, don't check brick and box

if self.rect.bottom >= c.GROUND\_HEIGHT:

sprite\_group = level.ground\_step\_pipe\_group

else:

sprite\_group = pg.sprite.Group(level.ground\_step\_pipe\_group,

level.brick\_group, level.box\_group)

sprite = pg.sprite.spritecollideany(self, sprite\_group)

if sprite and sprite.name != c.MAP\_SLIDER:

if self.rect.top <= sprite.rect.top:

self.rect.bottom = sprite.rect.y

self.y\_vel = 0

self.state = c.WALK

level.check\_is\_falling(self)

class Goomba(Enemy):

def \_\_init\_\_(self, x, y, direction, color, in\_range,

range\_start, range\_end, name=c.GOOMBA):

Enemy.\_\_init\_\_(self)

frame\_rect\_list = self.get\_frame\_rect(color)

self.setup\_enemy(x, y, direction, name, setup.GFX[c.MAIN\_ENEMY\_SHEET],

frame\_rect\_list, in\_range, range\_start, range\_end)

# dead jump image

self.frames.append(pg.transform.flip(self.frames[2], False, True))

# right walk images

self.frames.append(pg.transform.flip(self.frames[0], True, False))

self.frames.append(pg.transform.flip(self.frames[1], True, False))

def get\_frame\_rect(self, color):

if color == c.COLOR\_TYPE\_GREEN:

frame\_rect\_list = [(48, 48, 16, 16), (64, 48, 16, 16),

(80, 48, 16, 16)]

else:

frame\_rect\_list = [(48, 16, 16, 16), (64, 16, 16, 16),

(80, 16, 16, 16)]

return frame\_rect\_list

def jumped\_on(self):

self.x\_vel = 0

self.frame\_index = 2

if self.death\_timer == 0:

self.death\_timer = self.current\_time

elif (self.current\_time - self.death\_timer) > 500:

self.kill()

class Koopa(Enemy):

def \_\_init\_\_(self, x, y, direction, color, in\_range,

range\_start, range\_end, name=c.KOOPA):

Enemy.\_\_init\_\_(self)

frame\_rect\_list = self.get\_frame\_rect(color)

self.setup\_enemy(x, y, direction, name, setup.GFX[c.ENEMY\_SHEET],

frame\_rect\_list, in\_range, range\_start, range\_end)

# dead jump image

self.frames.append(pg.transform.flip(self.frames[2], False, True))

# right walk images

self.frames.append(pg.transform.flip(self.frames[0], True, False))

self.frames.append(pg.transform.flip(self.frames[1], True, False))

def get\_frame\_rect(self, color):

if color == c.COLOR\_TYPE\_GREEN:

frame\_rect\_list = [(150, 0, 16, 24), (180, 0, 16, 24),

(360, 5, 16, 15)]

elif color == c.COLOR\_TYPE\_RED:

frame\_rect\_list = [(150, 30, 16, 24), (180, 30, 16, 24),

(360, 35, 16, 15)]

else:

frame\_rect\_list = [(150, 60, 16, 24), (180, 60, 16, 24),

(360, 65, 16, 15)]

return frame\_rect\_list

def jumped\_on(self):

self.x\_vel = 0

self.frame\_index = 2

x = self.rect.x

bottom = self.rect.bottom

self.rect = self.frames[self.frame\_index].get\_rect()

self.rect.x = x

self.rect.bottom = bottom

self.in\_range = False

class FlyKoopa(Enemy):

def \_\_init\_\_(self, x, y, direction, color, in\_range,

range\_start, range\_end, isVertical, name=c.FLY\_KOOPA):

Enemy.\_\_init\_\_(self)

frame\_rect\_list = self.get\_frame\_rect(color)

self.setup\_enemy(x, y, direction, name, setup.GFX[c.ENEMY\_SHEET],

frame\_rect\_list, in\_range, range\_start, range\_end, isVertical)

# dead jump image

self.frames.append(pg.transform.flip(self.frames[2], False, True))

# right walk images

self.frames.append(pg.transform.flip(self.frames[0], True, False))

self.frames.append(pg.transform.flip(self.frames[1], True, False))

self.state = c.FLY

def get\_frame\_rect(self, color):

if color == c.COLOR\_TYPE\_GREEN:

frame\_rect\_list = [(90, 0, 16, 24), (120, 0, 16, 24),

(330, 5, 16, 15)]

else:

frame\_rect\_list = [(90, 30, 16, 24), (120, 30, 16, 24),

(330, 35, 16, 15)]

return frame\_rect\_list

def jumped\_on(self):

self.x\_vel = 0

self.frame\_index = 2

x = self.rect.x

bottom = self.rect.bottom

self.rect = self.frames[self.frame\_index].get\_rect()

self.rect.x = x

self.rect.bottom = bottom

self.in\_range = False

self.isVertical = False

class FireKoopa(Enemy):

def \_\_init\_\_(self, x, y, direction, color, in\_range,

range\_start, range\_end, level, name=c.FIRE\_KOOPA):

Enemy.\_\_init\_\_(self)

frame\_rect\_list = [(2, 210, 32, 32), (42, 210, 32, 32),

(82, 210, 32, 32), (122, 210, 32, 32)]

self.setup\_enemy(x, y, direction, name, setup.GFX[c.ENEMY\_SHEET],

frame\_rect\_list, in\_range, range\_start, range\_end)

# right walk images

self.frames.append(pg.transform.flip(self.frames[0], True, False))

self.frames.append(pg.transform.flip(self.frames[1], True, False))

self.frames.append(pg.transform.flip(self.frames[2], True, False))

self.frames.append(pg.transform.flip(self.frames[3], True, False))

self.x\_vel = 0

self.gravity = 0.3

self.level = level

self.fire\_timer = 0

self.jump\_timer = 0

def load\_frames(self, sheet, frame\_rect\_list):

for frame\_rect in frame\_rect\_list:

self.frames.append(tools.get\_image(sheet, \*frame\_rect,

c.BLACK, c.BRICK\_SIZE\_MULTIPLIER))

def walking(self):

if (self.current\_time - self.animate\_timer) > 250:

if self.direction == c.RIGHT:

self.frame\_index += 1

if self.frame\_index > 7:

self.frame\_index = 4

else:

self.frame\_index += 1

if self.frame\_index > 3:

self.frame\_index = 0

self.animate\_timer = self.current\_time

self.shoot\_fire()

if self.should\_jump():

self.y\_vel = -7

def falling(self):

if self.y\_vel < 7:

self.y\_vel += self.gravity

self.shoot\_fire()

def should\_jump(self):

if (self.rect.x - self.level.player.rect.x) < 400:

if (self.current\_time - self.jump\_timer) > 2500:

self.jump\_timer = self.current\_time

return True

return False

def shoot\_fire(self):

if (self.current\_time - self.fire\_timer) > 3000:

self.fire\_timer = self.current\_time

self.level.enemy\_group.add(Fire(self.rect.x, self.rect.bottom - 20, self.direction))

class Fire(Enemy):

def \_\_init\_\_(self, x, y, direction, name=c.FIRE):

Enemy.\_\_init\_\_(self)

frame\_rect\_list = [(101, 253, 23, 8), (131, 253, 23, 8)]

in\_range, range\_start, range\_end = False, 0, 0

self.setup\_enemy(x, y, direction, name, setup.GFX[c.ENEMY\_SHEET],

frame\_rect\_list, in\_range, range\_start, range\_end)

# right images

self.frames.append(pg.transform.flip(self.frames[0], True, False))

self.frames.append(pg.transform.flip(self.frames[1], True, False))

self.state = c.FLY

self.x\_vel = 5 if self.direction == c.RIGHT else -5

def check\_x\_collisions(self, level):

sprite\_group = pg.sprite.Group(level.ground\_step\_pipe\_group,

level.brick\_group, level.box\_group)

sprite = pg.sprite.spritecollideany(self, sprite\_group)

if sprite:

self.kill()

def start\_death\_jump(self, direction):

self.kill()

class Piranha(Enemy):

def \_\_init\_\_(self, x, y, direction, color, in\_range,

range\_start, range\_end, name=c.PIRANHA):

Enemy.\_\_init\_\_(self)

frame\_rect\_list = self.get\_frame\_rect(color)

self.setup\_enemy(x, y, direction, name, setup.GFX[c.ENEMY\_SHEET],

frame\_rect\_list, in\_range, range\_start, range\_end)

self.state = c.REVEAL

self.y\_vel = 1

self.wait\_timer = 0

self.group = pg.sprite.Group()

self.group.add(self)

def get\_frame\_rect(self, color):

if color == c.COLOR\_TYPE\_GREEN:

frame\_rect\_list = [(390, 30, 16, 24), (420, 30, 16, 24)]

else:

frame\_rect\_list = [(390, 60, 16, 24), (420, 60, 16, 24)]

return frame\_rect\_list

def revealing(self):

if (self.current\_time - self.animate\_timer) > 250:

if self.frame\_index == 0:

self.frame\_index += 1

elif self.frame\_index == 1:

self.frame\_index = 0

self.animate\_timer = self.current\_time

def update\_position(self, level):

if self.check\_player\_is\_on(level):

pass

else:

if self.rect.y < self.range\_start:

self.rect.y = self.range\_start

self.y\_vel = 1

elif self.rect.bottom > self.range\_end:

if self.wait\_timer == 0:

self.wait\_timer = self.current\_time

elif (self.current\_time - self.wait\_timer) < 3000:

return

else:

self.wait\_timer = 0

self.rect.bottom = self.range\_end

self.y\_vel = -1

self.rect.y += self.y\_vel

def check\_player\_is\_on(self, level):

result = False

self.rect.y -= 5

sprite = pg.sprite.spritecollideany(level.player, self.group)

if sprite:

result = True

self.rect.y += 5

return result

def start\_death\_jump(self, direction):

self.kill()

class FireStick(pg.sprite.Sprite):

def \_\_init\_\_(self, center\_x, center\_y, direction, color, radius, name=c.FIRESTICK):

'''the firestick will rotate around the center of a circle'''

pg.sprite.Sprite.\_\_init\_\_(self)

self.frames = []

self.frame\_index = 0

self.animate\_timer = 0

self.name = name

rect\_list = [(96, 144, 8, 8), (104, 144, 8, 8),

(96, 152, 8, 8), (104, 152, 8, 8)]

self.load\_frames(setup.GFX[c.ITEM\_SHEET], rect\_list)

self.animate\_timer = 0

self.image = self.frames[self.frame\_index]

self.rect = self.image.get\_rect()

self.rect.x = center\_x - radius

self.rect.y = center\_y

self.center\_x = center\_x

self.center\_y = center\_y

self.radius = radius

self.angle = 0

def load\_frames(self, sheet, frame\_rect\_list):

for frame\_rect in frame\_rect\_list:

self.frames.append(tools.get\_image(sheet, \*frame\_rect,

c.BLACK, c.BRICK\_SIZE\_MULTIPLIER))

def update(self, game\_info, level):

self.current\_time = game\_info[c.CURRENT\_TIME]

if (self.current\_time - self.animate\_timer) > 100:

if self.frame\_index < 3:

self.frame\_index += 1

else:

self.frame\_index = 0

self.animate\_timer = self.current\_time

self.image = self.frames[self.frame\_index]

self.angle += 1

if self.angle == 360:

self.angle = 0

radian = math.radians(self.angle)

self.rect.x = self.center\_x + math.sin(radian) \* self.radius

self.rect.y = self.center\_y + math.cos(radian) \* self.radius

### info.py

import pygame as pg

from . import coin

from .. import constants as c

from .. import setup, tools

class Character(pg.sprite.Sprite):

def \_\_init\_\_(self, image):

pg.sprite.Sprite.\_\_init\_\_(self)

self.image = image

self.rect = self.image.get\_rect()

class Info():

def \_\_init\_\_(self, game\_info, state):

self.coin\_total = game\_info[c.COIN\_TOTAL]

self.total\_lives = game\_info[c.LIVES]

self.state = state

self.game\_info = game\_info

self.create\_font\_image\_dict()

self.create\_info\_labels()

self.create\_state\_labels()

self.flashing\_coin = coin.FlashCoin(280, 53)

def create\_font\_image\_dict(self):

self.image\_dict = {}

image\_list = []

image\_rect\_list = [ # 0 - 9

(3, 230, 7, 7), (12, 230, 7, 7), (19, 230, 7, 7),

(27, 230, 7, 7), (35, 230, 7, 7), (43, 230, 7, 7),

(51, 230, 7, 7), (59, 230, 7, 7), (67, 230, 7, 7),

(75, 230, 7, 7),

# A - Z

(83, 230, 7, 7), (91, 230, 7, 7), (99, 230, 7, 7),

(107, 230, 7, 7), (115, 230, 7, 7), (123, 230, 7, 7),

(3, 238, 7, 7), (11, 238, 7, 7), (20, 238, 7, 7),

(27, 238, 7, 7), (35, 238, 7, 7), (44, 238, 7, 7),

(51, 238, 7, 7), (59, 238, 7, 7), (67, 238, 7, 7),

(75, 238, 7, 7), (83, 238, 7, 7), (91, 238, 7, 7),

(99, 238, 7, 7), (108, 238, 7, 7), (115, 238, 7, 7),

(123, 238, 7, 7), (3, 246, 7, 7), (11, 246, 7, 7),

(20, 246, 7, 7), (27, 246, 7, 7), (48, 246, 7, 7),

# -\*

(68, 249, 6, 2), (75, 247, 6, 6)]

character\_string = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ -\*'

for character, image\_rect in zip(character\_string, image\_rect\_list):

self.image\_dict[character] = tools.get\_image(setup.GFX['text\_images'],

\*image\_rect, (92, 148, 252), 2.9)

def create\_info\_labels(self):

self.score\_text = []

self.coin\_count\_text = []

self.mario\_label = []

self.world\_label = []

self.time\_label = []

self.stage\_label = []

self.create\_label(self.score\_text, '000000', 75, 55)

self.create\_label(self.coin\_count\_text, '\*00', 300, 55)

self.create\_label(self.mario\_label, 'MARIO', 75, 30)

self.create\_label(self.world\_label, 'WORLD', 450, 30)

self.create\_label(self.time\_label, 'TIME', 625, 30)

self.create\_label(self.stage\_label, '1-1', 472, 55)

self.info\_labels = [self.score\_text, self.coin\_count\_text, self.mario\_label,

self.world\_label, self.time\_label, self.stage\_label]

def create\_state\_labels(self):

if self.state == c.MAIN\_MENU:

self.create\_main\_menu\_labels()

elif self.state == c.LOAD\_SCREEN:

self.create\_player\_image()

self.create\_load\_screen\_labels()

elif self.state == c.LEVEL:

self.create\_level\_labels()

elif self.state == c.GAME\_OVER:

self.create\_game\_over\_labels()

elif self.state == c.TIME\_OUT:

self.create\_time\_out\_labels()

def create\_player\_image(self):

self.life\_times\_image = tools.get\_image(setup.GFX['text\_images'],

75, 247, 6, 6, (92, 148, 252), 2.9)

self.life\_times\_rect = self.life\_times\_image.get\_rect(center=(378, 295))

self.life\_total\_label = []

self.create\_label(self.life\_total\_label, str(self.total\_lives), 450, 285)

if self.game\_info[c.PLAYER\_NAME] == c.PLAYER\_MARIO:

rect = (0, 0, 208, 225)

else:

rect = (208, 0, 208, 225)

self.player\_image = tools.get\_image(setup.GFX['player'],

\*rect, (0, 0, 0), 2.5/7)

self.player\_rect = self.player\_image.get\_rect(center=(320, 290))

def create\_main\_menu\_labels(self):

mario\_game = []

luigi\_game = []

top = []

top\_score = []

self.create\_label(mario\_game, c.PLAYER1, 272, 360)

self.create\_label(luigi\_game, c.PLAYER2, 272, 405)

self.create\_label(top, 'TOP - ', 290, 465)

self.create\_label(top\_score, '634634', 400, 465)

self.state\_labels = [mario\_game, luigi\_game, top, top\_score,

\*self.info\_labels]

def create\_load\_screen\_labels(self):

world\_label = []

self.stage\_label2 = []

self.create\_label(world\_label, 'WORLD', 280, 200)

self.create\_label(self.stage\_label2, '1-1', 430, 200)

self.state\_labels = [world\_label, self.stage\_label2,

\*self.info\_labels, self.life\_total\_label]

def create\_level\_labels(self):

self.time = c.GAME\_TIME\_OUT

self.current\_time = 0

self.clock\_time\_label = []

self.create\_label(self.clock\_time\_label, str(self.time), 645, 55)

self.state\_labels = [\*self.info\_labels, self.clock\_time\_label]

def create\_game\_over\_labels(self):

game\_label = []

over\_label = []

self.create\_label(game\_label, 'GAME', 280, 300)

self.create\_label(over\_label, 'OVER', 400, 300)

self.state\_labels = [game\_label, over\_label, \*self.info\_labels]

def create\_time\_out\_labels(self):

timeout\_label = []

self.create\_label(timeout\_label, 'TIME OUT', 290, 310)

self.state\_labels = [timeout\_label, \*self.info\_labels]

def create\_label(self, label\_list, string, x, y):

for letter in string:

label\_list.append(Character(self.image\_dict[letter]))

self.set\_label\_rects(label\_list, x, y)

def set\_label\_rects(self, label\_list, x, y):

for i, letter in enumerate(label\_list):

letter.rect.x = x + ((letter.rect.width + 3) \* i)

letter.rect.y = y

if letter.image == self.image\_dict['-']:

letter.rect.y += 7

letter.rect.x += 2

def update(self, level\_info, level=None):

self.level = level

self.handle\_level\_state(level\_info)

def handle\_level\_state(self, level\_info):

self.score = level\_info[c.SCORE]

self.update\_text(self.score\_text, self.score)

self.update\_text(self.coin\_count\_text, level\_info[c.COIN\_TOTAL])

self.update\_text(self.stage\_label, level\_info[c.LEVEL\_NUM])

self.flashing\_coin.update(level\_info[c.CURRENT\_TIME])

if self.state == c.LOAD\_SCREEN:

self.update\_text(self.stage\_label2, level\_info[c.LEVEL\_NUM])

if self.state == c.LEVEL:

if (level\_info[c.CURRENT\_TIME] - self.current\_time) > 1000:

self.current\_time = level\_info[c.CURRENT\_TIME]

self.time -= 1

self.update\_text(self.clock\_time\_label, self.time, True)

def update\_text(self, text, score, reset=False):

if reset and len(text) > len(str(score)):

text.remove(text[0])

index = len(text) - 1

for digit in reversed(str(score)):

rect = text[index].rect

text[index] = Character(self.image\_dict[digit])

text[index].rect = rect

index -= 1

def draw(self, surface):

self.draw\_info(surface, self.state\_labels)

if self.state == c.LOAD\_SCREEN:

surface.blit(self.player\_image, self.player\_rect)

surface.blit(self.life\_times\_image, self.life\_times\_rect)

surface.blit(self.flashing\_coin.image, self.flashing\_coin.rect)

def draw\_info(self, surface, label\_list):

for label in label\_list:

for letter in label:

surface.blit(letter.image, letter.rect)

### player.py

import json

import os

import pygame as pg

from .. import constants as c

from .. import setup, tools

from ..components import powerup

class Player(pg.sprite.Sprite):

def \_\_init\_\_(self, player\_name):

pg.sprite.Sprite.\_\_init\_\_(self)

self.player\_name = player\_name

self.load\_data()

self.setup\_timer()

self.setup\_state()

self.setup\_speed()

self.load\_images()

if c.DEBUG:

self.right\_frames = self.big\_fire\_frames[0]

self.left\_frames = self.big\_fire\_frames[1]

self.big = True

self.fire = True

self.frame\_index = 0

self.state = c.WALK

self.image = self.right\_frames[self.frame\_index]

self.rect = self.image.get\_rect()

def restart(self):

'''restart after player is dead or go to next level'''

if self.dead:

self.dead = False

self.big = False

self.fire = False

self.set\_player\_image(self.small\_normal\_frames, 0)

self.right\_frames = self.small\_normal\_frames[0]

self.left\_frames = self.small\_normal\_frames[1]

self.state = c.STAND

def load\_data(self):

player\_file = str(self.player\_name) + '.json'

file\_path = os.path.join('source', 'data', 'player', player\_file)

f = open(file\_path)

self.player\_data = json.load(f)

def setup\_timer(self):

self.walking\_timer = 0

self.death\_timer = 0

self.flagpole\_timer = 0

self.transition\_timer = 0

self.hurt\_invincible\_timer = 0

self.invincible\_timer = 0

self.last\_fireball\_time = 0

def setup\_state(self):

self.facing\_right = True

self.allow\_jump = True

self.allow\_fireball = True

self.dead = False

self.big = False

self.fire = False

self.hurt\_invincible = False

self.invincible = False

self.crouching = False

def setup\_speed(self):

speed = self.player\_data[c.PLAYER\_SPEED]

self.x\_vel = 0

self.y\_vel = 0

self.max\_walk\_vel = speed[c.MAX\_WALK\_SPEED]

self.max\_run\_vel = speed[c.MAX\_RUN\_SPEED]

self.max\_y\_vel = speed[c.MAX\_Y\_VEL]

self.walk\_accel = speed[c.WALK\_ACCEL]

self.run\_accel = speed[c.RUN\_ACCEL]

self.jump\_vel = speed[c.JUMP\_VEL]

self.gravity = c.GRAVITY

self.max\_x\_vel = self.max\_walk\_vel

self.x\_accel = self.walk\_accel

def load\_images(self):

sheet = setup.GFX['player']

frames\_list = self.player\_data[c.PLAYER\_FRAMES]

self.right\_frames = []

self.left\_frames = []

self.right\_small\_normal\_frames = []

self.left\_small\_normal\_frames = []

self.right\_big\_normal\_frames = []

self.left\_big\_normal\_frames = []

self.right\_big\_fire\_frames = []

self.left\_big\_fire\_frames = []

for name, frames in frames\_list.items():

for frame in frames:

image = tools.get\_image(sheet, frame['x'], frame['y'],

frame['width'], frame['height'],

c.BLACK, c.SIZE\_MULTIPLIER/13)

left\_image = pg.transform.flip(image, True, False)

if name == c.RIGHT\_SMALL\_NORMAL:

self.right\_small\_normal\_frames.append(image)

self.left\_small\_normal\_frames.append(left\_image)

elif name == c.RIGHT\_BIG\_NORMAL:

self.right\_big\_normal\_frames.append(image)

self.left\_big\_normal\_frames.append(left\_image)

elif name == c.RIGHT\_BIG\_FIRE:

self.right\_big\_fire\_frames.append(image)

self.left\_big\_fire\_frames.append(left\_image)

self.small\_normal\_frames = [self.right\_small\_normal\_frames,

self.left\_small\_normal\_frames]

self.big\_normal\_frames = [self.right\_big\_normal\_frames,

self.left\_big\_normal\_frames]

self.big\_fire\_frames = [self.right\_big\_fire\_frames,

self.left\_big\_fire\_frames]

self.all\_images = [self.right\_small\_normal\_frames,

self.left\_small\_normal\_frames,

self.right\_big\_normal\_frames,

self.left\_big\_normal\_frames,

self.right\_big\_fire\_frames,

self.left\_big\_fire\_frames]

self.right\_frames = self.small\_normal\_frames[0]

self.left\_frames = self.small\_normal\_frames[1]

def update(self, keys, game\_info, fire\_group):

self.current\_time = game\_info[c.CURRENT\_TIME]

self.handle\_state(keys, fire\_group)

self.check\_if\_hurt\_invincible()

self.check\_if\_invincible()

self.animation()

def handle\_state(self, keys, fire\_group):

if self.state == c.STAND:

self.standing(keys, fire\_group)

elif self.state == c.WALK:

self.walking(keys, fire\_group)

elif self.state == c.JUMP:

self.jumping(keys, fire\_group)

elif self.state == c.FALL:

self.falling(keys, fire\_group)

elif self.state == c.DEATH\_JUMP:

self.jumping\_to\_death()

elif self.state == c.FLAGPOLE:

self.flag\_pole\_sliding()

elif self.state == c.WALK\_AUTO:

self.walking\_auto()

elif self.state == c.END\_OF\_LEVEL\_FALL:

self.y\_vel += self.gravity

elif self.state == c.IN\_CASTLE:

self.frame\_index = 0

elif self.state == c.SMALL\_TO\_BIG:

self.changing\_to\_big()

elif self.state == c.BIG\_TO\_SMALL:

self.changing\_to\_small()

elif self.state == c.BIG\_TO\_FIRE:

self.changing\_to\_fire()

elif self.state == c.DOWN\_TO\_PIPE:

self.y\_vel = 1

self.rect.y += self.y\_vel

elif self.state == c.UP\_OUT\_PIPE:

self.y\_vel = -1

self.rect.y += self.y\_vel

if self.rect.bottom < self.up\_pipe\_y:

self.state = c.STAND

def check\_to\_allow\_jump(self, keys):

if not keys[tools.keybinding['jump']]: # and not tools.tmp[-1] > 4000:#todo

self.allow\_jump = True

def check\_to\_allow\_fireball(self, keys):

if not keys[tools.keybinding['action']]:

self.allow\_fireball = True

def standing(self, keys, fire\_group):

self.check\_to\_allow\_jump(keys)

self.check\_to\_allow\_fireball(keys)

self.frame\_index = 0

self.x\_vel = 0

self.y\_vel = 0

if keys[tools.keybinding['action']]:

if self.fire and self.allow\_fireball:

self.shoot\_fireball(fire\_group)

if keys[tools.keybinding['down']]:

self.update\_crouch\_or\_not(True)

if keys[tools.keybinding['left']]:

self.facing\_right = False

self.update\_crouch\_or\_not()

self.state = c.WALK

elif keys[tools.keybinding['right']]: # or tools.tmp[-1] > 1300:#todo

self.facing\_right = True

self.update\_crouch\_or\_not()

self.state = c.WALK

elif keys[tools.keybinding['jump']]: # or tools.tmp[-1] > 4000:#todo

if self.allow\_jump:

self.state = c.JUMP

self.y\_vel = self.jump\_vel

if not keys[tools.keybinding['down']]:

self.update\_crouch\_or\_not()

def update\_crouch\_or\_not(self, isDown=False):

if not self.big:

self.crouching = True if isDown else False

return

if not isDown and not self.crouching:

return

self.crouching = True if isDown else False

frame\_index = 7 if isDown else 0

bottom = self.rect.bottom

left = self.rect.x

if self.facing\_right:

self.image = self.right\_frames[frame\_index]

else:

self.image = self.left\_frames[frame\_index]

self.rect = self.image.get\_rect()

self.rect.bottom = bottom

self.rect.x = left

self.frame\_index = frame\_index

def walking(self, keys, fire\_group):

self.check\_to\_allow\_jump(keys)

self.check\_to\_allow\_fireball(keys)

if self.frame\_index == 0:

self.frame\_index += 1

self.walking\_timer = self.current\_time

elif (self.current\_time - self.walking\_timer >

self.calculate\_animation\_speed()):

if self.frame\_index < 3:

self.frame\_index += 1

else:

self.frame\_index = 1

self.walking\_timer = self.current\_time

if keys[tools.keybinding['action']]:

self.max\_x\_vel = self.max\_run\_vel

self.x\_accel = self.run\_accel

if self.fire and self.allow\_fireball:

self.shoot\_fireball(fire\_group)

else:

self.max\_x\_vel = self.max\_walk\_vel

self.x\_accel = self.walk\_accel

if keys[tools.keybinding['jump']]: # or tools.tmp[-1] > 4000:#todo

if self.allow\_jump:

self.state = c.JUMP

if abs(self.x\_vel) > 3:

self.y\_vel = self.jump\_vel - .5

else:

self.y\_vel = self.jump\_vel

if keys[tools.keybinding['left']]:

self.facing\_right = False

if self.x\_vel > 0:

self.frame\_index = 5

self.x\_accel = c.SMALL\_TURNAROUND

self.x\_vel = self.cal\_vel(self.x\_vel, self.max\_x\_vel, self.x\_accel, True)

elif keys[tools.keybinding['right']]: # or tools.tmp[-1] > 1300:#todo

self.facing\_right = True

if self.x\_vel < 0:

self.frame\_index = 5

self.x\_accel = c.SMALL\_TURNAROUND

self.x\_vel = self.cal\_vel(self.x\_vel, self.max\_x\_vel, self.x\_accel)

else:

if self.facing\_right:

if self.x\_vel > 0:

self.x\_vel -= self.x\_accel

else:

self.x\_vel = 0

self.state = c.STAND

else:

if self.x\_vel < 0:

self.x\_vel += self.x\_accel

else:

self.x\_vel = 0

self.state = c.STAND

def jumping(self, keys, fire\_group):

""" y\_vel value: positive is down, negative is up """

self.check\_to\_allow\_fireball(keys)

self.allow\_jump = False

self.frame\_index = 4

self.gravity = c.JUMP\_GRAVITY

self.y\_vel += self.gravity

if self.y\_vel >= 0 and self.y\_vel < self.max\_y\_vel:

self.gravity = c.GRAVITY

self.state = c.FALL

if keys[tools.keybinding['right']]: # or tools.tmp[-1] > 1300:#todo

self.x\_vel = self.cal\_vel(self.x\_vel, self.max\_x\_vel, self.x\_accel)

elif keys[tools.keybinding['left']]:

self.x\_vel = self.cal\_vel(self.x\_vel, self.max\_x\_vel, self.x\_accel, True)

if not keys[tools.keybinding['jump']]: # and not tools.tmp[-1] > 4000:#todo

self.gravity = c.GRAVITY

self.state = c.FALL

if keys[tools.keybinding['action']]:

if self.fire and self.allow\_fireball:

self.shoot\_fireball(fire\_group)

def falling(self, keys, fire\_group):

self.check\_to\_allow\_fireball(keys)

self.y\_vel = self.cal\_vel(self.y\_vel, self.max\_y\_vel, self.gravity)

if keys[tools.keybinding['right']]:

self.x\_vel = self.cal\_vel(self.x\_vel, self.max\_x\_vel, self.x\_accel)

elif keys[tools.keybinding['left']]:

self.x\_vel = self.cal\_vel(self.x\_vel, self.max\_x\_vel, self.x\_accel, True)

if keys[tools.keybinding['action']]:

if self.fire and self.allow\_fireball:

self.shoot\_fireball(fire\_group)

def jumping\_to\_death(self):

if self.death\_timer == 0:

self.death\_timer = self.current\_time

elif (self.current\_time - self.death\_timer) > 500:

self.rect.y += self.y\_vel

self.y\_vel += self.gravity

def cal\_vel(self, vel, max\_vel, accel, isNegative=False):

""" max\_vel and accel must > 0 """

if isNegative:

new\_vel = vel \* -1

else:

new\_vel = vel

if (new\_vel + accel) < max\_vel:

new\_vel += accel

else:

new\_vel = max\_vel

if isNegative:

return new\_vel \* -1

else:

return new\_vel

def calculate\_animation\_speed(self):

if self.x\_vel == 0:

animation\_speed = 130

elif self.x\_vel > 0:

animation\_speed = 130 - (self.x\_vel \* 13)

else:

animation\_speed = 130 - (self.x\_vel \* 13 \* -1)

return animation\_speed

def shoot\_fireball(self, powerup\_group):

if (self.current\_time - self.last\_fireball\_time) > 500:

self.allow\_fireball = False

powerup\_group.add(powerup.FireBall(self.rect.right,

self.rect.y, self.facing\_right))

self.last\_fireball\_time = self.current\_time

self.frame\_index = 6

def flag\_pole\_sliding(self):

self.state = c.FLAGPOLE

self.x\_vel = 0

self.y\_vel = 5

if self.flagpole\_timer == 0:

self.flagpole\_timer = self.current\_time

elif self.rect.bottom < 493:

if (self.current\_time - self.flagpole\_timer) < 65:

self.frame\_index = 9

elif (self.current\_time - self.flagpole\_timer) < 130:

self.frame\_index = 10

else:

self.flagpole\_timer = self.current\_time

elif self.rect.bottom >= 493:

self.frame\_index = 10

def walking\_auto(self):

self.max\_x\_vel = 5

self.x\_accel = self.walk\_accel

self.x\_vel = self.cal\_vel(self.x\_vel, self.max\_x\_vel, self.x\_accel)

if (self.walking\_timer == 0 or (self.current\_time - self.walking\_timer) > 200):

self.walking\_timer = self.current\_time

elif (self.current\_time - self.walking\_timer >

self.calculate\_animation\_speed()):

if self.frame\_index < 3:

self.frame\_index += 1

else:

self.frame\_index = 1

self.walking\_timer = self.current\_time

def changing\_to\_big(self):

timer\_list = [135, 200, 365, 430, 495, 560, 625, 690, 755, 820, 885]

# size value 0:small, 1:middle, 2:big

size\_list = [1, 0, 1, 0, 1, 2, 0, 1, 2, 0, 2]

frames = [(self.small\_normal\_frames, 0), (self.small\_normal\_frames, 7),

(self.big\_normal\_frames, 0)]

if self.transition\_timer == 0:

self.big = True

self.change\_index = 0

self.transition\_timer = self.current\_time

elif (self.current\_time - self.transition\_timer) > timer\_list[self.change\_index]:

if (self.change\_index + 1) >= len(timer\_list):

# player becomes big

self.transition\_timer = 0

self.set\_player\_image(self.big\_normal\_frames, 0)

self.state = c.WALK

self.right\_frames = self.right\_big\_normal\_frames

self.left\_frames = self.left\_big\_normal\_frames

else:

frame, frame\_index = frames[size\_list[self.change\_index]]

self.set\_player\_image(frame, frame\_index)

self.change\_index += 1

def changing\_to\_small(self):

timer\_list = [265, 330, 395, 460, 525, 590, 655, 720, 785, 850, 915]

# size value 0:big, 1:middle, 2:small

size\_list = [0, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2]

frames = [(self.big\_normal\_frames, 4), (self.big\_normal\_frames, 8),

(self.small\_normal\_frames, 8)]

if self.transition\_timer == 0:

self.change\_index = 0

self.transition\_timer = self.current\_time

elif (self.current\_time - self.transition\_timer) > timer\_list[self.change\_index]:

if (self.change\_index + 1) >= len(timer\_list):

# player becomes small

self.transition\_timer = 0

self.set\_player\_image(self.small\_normal\_frames, 0)

self.state = c.WALK

self.big = False

self.fire = False

self.hurt\_invincible = True

self.right\_frames = self.right\_small\_normal\_frames

self.left\_frames = self.left\_small\_normal\_frames

else:

frame, frame\_index = frames[size\_list[self.change\_index]]

self.set\_player\_image(frame, frame\_index)

self.change\_index += 1

def changing\_to\_fire(self):

timer\_list = [65, 195, 260, 325, 390, 455, 520, 585, 650, 715, 780, 845, 910, 975]

# size value 0:fire, 1:big green, 2:big red, 3:big black

size\_list = [0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1]

frames = [(self.big\_fire\_frames, 3), (self.big\_normal\_frames, 3),

(self.big\_fire\_frames, 3), (self.big\_normal\_frames, 3)]

if self.transition\_timer == 0:

self.change\_index = 0

self.transition\_timer = self.current\_time

elif (self.current\_time - self.transition\_timer) > timer\_list[self.change\_index]:

if (self.change\_index + 1) >= len(timer\_list):

# player becomes fire

self.transition\_timer = 0

self.set\_player\_image(self.big\_fire\_frames, 3)

self.fire = True

self.state = c.WALK

self.right\_frames = self.right\_big\_fire\_frames

self.left\_frames = self.left\_big\_fire\_frames

else:

frame, frame\_index = frames[size\_list[self.change\_index]]

self.set\_player\_image(frame, frame\_index)

self.change\_index += 1

def set\_player\_image(self, frames, frame\_index):

self.frame\_index = frame\_index

if self.facing\_right:

self.right\_frames = frames[0]

self.image = frames[0][frame\_index]

else:

self.left\_frames = frames[1]

self.image = frames[1][frame\_index]

bottom = self.rect.bottom

centerx = self.rect.centerx

self.rect = self.image.get\_rect()

self.rect.bottom = bottom

self.rect.centerx = centerx

def check\_if\_hurt\_invincible(self):

if self.hurt\_invincible:

if self.hurt\_invincible\_timer == 0:

self.hurt\_invincible\_timer = self.current\_time

self.hurt\_invincible\_timer2 = self.current\_time

elif (self.current\_time - self.hurt\_invincible\_timer) < 2000:

if (self.current\_time - self.hurt\_invincible\_timer2) < 35:

self.image.set\_alpha(0)

elif (self.current\_time - self.hurt\_invincible\_timer2) < 70:

self.image.set\_alpha(255)

self.hurt\_invincible\_timer2 = self.current\_time

else:

self.hurt\_invincible = False

self.hurt\_invincible\_timer = 0

for frames in self.all\_images:

for image in frames:

image.set\_alpha(255)

def check\_if\_invincible(self):

if self.invincible:

if self.invincible\_timer == 0:

self.invincible\_timer = self.current\_time

self.invincible\_timer2 = self.current\_time

elif (self.current\_time - self.invincible\_timer) < 10000:

if (self.current\_time - self.invincible\_timer2) < 35:

self.image.set\_alpha(0)

elif (self.current\_time - self.invincible\_timer2) < 70:

self.image.set\_alpha(255)

self.invincible\_timer2 = self.current\_time

elif (self.current\_time - self.invincible\_timer) < 12000:

if (self.current\_time - self.invincible\_timer2) < 100:

self.image.set\_alpha(0)

elif (self.current\_time - self.invincible\_timer2) < 200:

self.image.set\_alpha(255)

self.invincible\_timer2 = self.current\_time

else:

self.invincible = False

self.invincible\_timer = 0

for frames in self.all\_images:

for image in frames:

image.set\_alpha(255)

def animation(self):

if self.facing\_right:

self.image = self.right\_frames[self.frame\_index]

else:

self.image = self.left\_frames[self.frame\_index]

def start\_death\_jump(self, game\_info):

self.dead = True

self.y\_vel = -11

self.gravity = .5

self.frame\_index = 6

self.state = c.DEATH\_JUMP

### powerup.py

import pygame as pg

from . import stuff

from .. import constants as c

from .. import setup

class Powerup(stuff.Stuff):

def \_\_init\_\_(self, x, y, sheet, image\_rect\_list, scale):

stuff.Stuff.\_\_init\_\_(self, x, y, sheet, image\_rect\_list, scale)

self.rect.centerx = x

self.state = c.REVEAL

self.y\_vel = -1

self.x\_vel = 0

self.direction = c.RIGHT

self.box\_height = y

self.gravity = 1

self.max\_y\_vel = 8

self.animate\_timer = 0

def update\_position(self, level):

self.rect.x += self.x\_vel

self.check\_x\_collisions(level)

self.rect.y += self.y\_vel

self.check\_y\_collisions(level)

if self.rect.x <= 0:

self.kill()

elif self.rect.y > (level.viewport.bottom):

self.kill()

def check\_x\_collisions(self, level):

sprite\_group = pg.sprite.Group(level.ground\_step\_pipe\_group,

level.brick\_group, level.box\_group)

sprite = pg.sprite.spritecollideany(self, sprite\_group)

if sprite:

if self.direction == c.RIGHT:

self.rect.right = sprite.rect.left - 1

self.direction = c.LEFT

elif self.direction == c.LEFT:

self.rect.left = sprite.rect.right

self.direction = c.RIGHT

self.x\_vel = self.speed if self.direction == c.RIGHT else -1 \* self.speed

if sprite.name == c.MAP\_BRICK:

self.x\_vel = 0

def check\_y\_collisions(self, level):

sprite\_group = pg.sprite.Group(level.ground\_step\_pipe\_group,

level.brick\_group, level.box\_group)

sprite = pg.sprite.spritecollideany(self, sprite\_group)

if sprite:

self.y\_vel = 0

self.rect.bottom = sprite.rect.top

self.state = c.SLIDE

level.check\_is\_falling(self)

def animation(self):

self.image = self.frames[self.frame\_index]

class Mushroom(Powerup):

def \_\_init\_\_(self, x, y):

Powerup.\_\_init\_\_(self, x, y, setup.GFX[c.ITEM\_SHEET],

[(0, 0, 16, 16)], c.SIZE\_MULTIPLIER)

self.type = c.TYPE\_MUSHROOM

self.speed = 2

def update(self, game\_info, level):

if self.state == c.REVEAL:

self.rect.y += self.y\_vel

if self.rect.bottom <= self.box\_height:

self.rect.bottom = self.box\_height

self.y\_vel = 0

self.state = c.SLIDE

elif self.state == c.SLIDE:

self.x\_vel = self.speed if self.direction == c.RIGHT else -1 \* self.speed

elif self.state == c.FALL:

if self.y\_vel < self.max\_y\_vel:

self.y\_vel += self.gravity

if self.state == c.SLIDE or self.state == c.FALL:

self.update\_position(level)

self.animation()

class LifeMushroom(Mushroom):

def \_\_init\_\_(self, x, y):

Powerup.\_\_init\_\_(self, x, y, setup.GFX[c.ITEM\_SHEET],

[(16, 0, 16, 16)], c.SIZE\_MULTIPLIER)

self.type = c.TYPE\_LIFEMUSHROOM

self.speed = 2

class gong(Mushroom):

def \_\_init\_\_(self, x, y):

Powerup.\_\_init\_\_(self, x, y, setup.MY\_GFX[c.EGGSHELL\_PROJ[0]],

[(0, 0, 64, 64)], c.SIZE\_MULTIPLIER / 4)

self.type = c.EGGSHELL\_PROJ\_NUM[0]

self.speed = 0

class cheng(Mushroom):

def \_\_init\_\_(self, x, y):

Powerup.\_\_init\_\_(self, x, y, setup.MY\_GFX[c.EGGSHELL\_PROJ[1]],

[(0, 0, 64, 64)], c.SIZE\_MULTIPLIER / 4)

self.type = c.EGGSHELL\_PROJ\_NUM[1]

self.speed = 0

class jing(Mushroom):

def \_\_init\_\_(self, x, y):

Powerup.\_\_init\_\_(self, x, y, setup.MY\_GFX[c.EGGSHELL\_PROJ[2]],

[(0, 0, 64, 64)], c.SIZE\_MULTIPLIER / 4)

self.type = c.EGGSHELL\_PROJ\_NUM[2]

self.speed = 0

class ji(Mushroom):

def \_\_init\_\_(self, x, y):

Powerup.\_\_init\_\_(self, x, y, setup.MY\_GFX[c.EGGSHELL\_PROJ[3]],

[(0, 0, 64, 64)], c.SIZE\_MULTIPLIER / 4)

self.type = c.EGGSHELL\_PROJ\_NUM[3]

self.speed = 0

class xue(Mushroom):

def \_\_init\_\_(self, x, y):

Powerup.\_\_init\_\_(self, x, y, setup.MY\_GFX[c.EGGSHELL\_PROJ[4]],

[(0, 0, 64, 64)], c.SIZE\_MULTIPLIER / 4)

self.type = c.EGGSHELL\_PROJ\_NUM[4]

self.speed = 0

class dui(Mushroom):

def \_\_init\_\_(self, x, y):

Powerup.\_\_init\_\_(self, x, y, setup.MY\_GFX[c.JUDGMENT\_PROJ[0]],

[(0, 0, 256, 256)], c.SIZE\_MULTIPLIER / 16)

self.type = c.JUDGMENT\_PROJ\_NUM[0]

self.speed = 0

class cuo(Mushroom):

def \_\_init\_\_(self, x, y):

Powerup.\_\_init\_\_(self, x, y, setup.MY\_GFX[c.JUDGMENT\_PROJ[1]],

[(0, 0, 256, 256)], c.SIZE\_MULTIPLIER / 16)

self.type = c.JUDGMENT\_PROJ\_NUM[1]

self.speed = 0

class FireFlower(Powerup):

def \_\_init\_\_(self, x, y):

frame\_rect\_list = [(0, 32, 16, 16), (16, 32, 16, 16),

(32, 32, 16, 16), (48, 32, 16, 16)]

Powerup.\_\_init\_\_(self, x, y, setup.GFX[c.ITEM\_SHEET],

frame\_rect\_list, c.SIZE\_MULTIPLIER)

self.type = c.TYPE\_FIREFLOWER

def update(self, game\_info, \*args):

self.current\_time = game\_info[c.CURRENT\_TIME]

if self.state == c.REVEAL:

self.rect.y += self.y\_vel

if self.rect.bottom <= self.box\_height:

self.rect.bottom = self.box\_height

self.y\_vel = 0

self.state = c.RESTING

if (self.current\_time - self.animate\_timer) > 30:

if self.frame\_index < 3:

self.frame\_index += 1

else:

self.frame\_index = 0

self.animate\_timer = self.current\_time

self.animation()

class Star(Powerup):

def \_\_init\_\_(self, x, y):

frame\_rect\_list = [(1, 48, 15, 16), (17, 48, 15, 16),

(33, 48, 15, 16), (49, 48, 15, 16)]

Powerup.\_\_init\_\_(self, x, y, setup.GFX[c.ITEM\_SHEET],

frame\_rect\_list, c.SIZE\_MULTIPLIER)

self.type = c.TYPE\_STAR

self.gravity = .4

self.speed = 5

def update(self, game\_info, level):

self.current\_time = game\_info[c.CURRENT\_TIME]

if self.state == c.REVEAL:

self.rect.y += self.y\_vel

if self.rect.bottom <= self.box\_height:

self.rect.bottom = self.box\_height

self.y\_vel = -2

self.state = c.BOUNCING

elif self.state == c.BOUNCING:

self.y\_vel += self.gravity

self.x\_vel = self.speed if self.direction == c.RIGHT else -1 \* self.speed

if (self.current\_time - self.animate\_timer) > 30:

if self.frame\_index < 3:

self.frame\_index += 1

else:

self.frame\_index = 0

self.animate\_timer = self.current\_time

if self.state == c.BOUNCING:

self.update\_position(level)

self.animation()

def check\_y\_collisions(self, level):

sprite\_group = pg.sprite.Group(level.ground\_step\_pipe\_group,

level.brick\_group, level.box\_group)

sprite = pg.sprite.spritecollideany(self, sprite\_group)

if sprite:

if self.rect.top > sprite.rect.top:

self.y\_vel = 5

else:

self.rect.bottom = sprite.rect.y

self.y\_vel = -5

class FireBall(Powerup):

def \_\_init\_\_(self, x, y, facing\_right):

# first 3 Frames are flying, last 4 frams are exploding

frame\_rect\_list = [(96, 144, 8, 8), (104, 144, 8, 8),

(96, 152, 8, 8), (104, 152, 8, 8),

(112, 144, 16, 16), (112, 160, 16, 16),

(112, 176, 16, 16)]

Powerup.\_\_init\_\_(self, x, y, setup.GFX[c.ITEM\_SHEET],

frame\_rect\_list, c.SIZE\_MULTIPLIER)

self.type = c.TYPE\_FIREBALL

self.y\_vel = 10

self.gravity = .9

self.state = c.FLYING

self.rect.right = x

if facing\_right:

self.direction = c.RIGHT

self.x\_vel = 12

else:

self.direction = c.LEFT

self.x\_vel = -12

def update(self, game\_info, level):

self.current\_time = game\_info[c.CURRENT\_TIME]

if self.state == c.FLYING or self.state == c.BOUNCING:

self.y\_vel += self.gravity

if (self.current\_time - self.animate\_timer) > 200:

if self.frame\_index < 3:

self.frame\_index += 1

else:

self.frame\_index = 0

self.animate\_timer = self.current\_time

self.update\_position(level)

elif self.state == c.EXPLODING:

if (self.current\_time - self.animate\_timer) > 50:

if self.frame\_index < 6:

self.frame\_index += 1

else:

self.kill()

self.animate\_timer = self.current\_time

self.animation()

def check\_x\_collisions(self, level):

sprite\_group = pg.sprite.Group(level.ground\_step\_pipe\_group,

level.brick\_group, level.box\_group)

sprite = pg.sprite.spritecollideany(self, sprite\_group)

if sprite:

self.change\_to\_explode()

def check\_y\_collisions(self, level):

sprite\_group = pg.sprite.Group(level.ground\_step\_pipe\_group,

level.brick\_group, level.box\_group)

sprite = pg.sprite.spritecollideany(self, sprite\_group)

enemy = pg.sprite.spritecollideany(self, level.enemy\_group)

if sprite:

if self.rect.top > sprite.rect.top:

self.change\_to\_explode()

else:

self.rect.bottom = sprite.rect.y

self.y\_vel = -8

if self.direction == c.RIGHT:

self.x\_vel = 15

else:

self.x\_vel = -15

self.state = c.BOUNCING

elif enemy:

if (enemy.name != c.FIRESTICK):

level.update\_score(100, enemy, 0)

level.move\_to\_dying\_group(level.enemy\_group, enemy)

enemy.start\_death\_jump(self.direction)

self.change\_to\_explode()

def change\_to\_explode(self):

self.frame\_index = 4

self.state = c.EXPLODING

### stuff.py

import pygame as pg

from .. import constants as c

from .. import setup, tools

class Collider(pg.sprite.Sprite):

def \_\_init\_\_(self, x, y, width, height, name):

pg.sprite.Sprite.\_\_init\_\_(self)

self.image = pg.Surface((width, height)).convert()

self.rect = self.image.get\_rect()

self.rect.x = x

self.rect.y = y

self.name = name

if c.DEBUG:

self.image.fill(c.RED)

class Checkpoint(pg.sprite.Sprite):

def \_\_init\_\_(self, x, y, width, height, type, enemy\_groupid=0, map\_index=0, name=c.MAP\_CHECKPOINT):

pg.sprite.Sprite.\_\_init\_\_(self)

self.image = pg.Surface((width, height))

self.rect = self.image.get\_rect()

self.rect.x = x

self.rect.y = y

self.type = type

self.enemy\_groupid = enemy\_groupid

self.map\_index = map\_index

self.name = name

class Stuff(pg.sprite.Sprite):

def \_\_init\_\_(self, x, y, sheet, image\_rect\_list, scale):

pg.sprite.Sprite.\_\_init\_\_(self)

self.frames = []

self.frame\_index = 0

for image\_rect in image\_rect\_list:

self.frames.append(tools.get\_image(sheet,

\*image\_rect, c.BLACK, scale))

self.image = self.frames[self.frame\_index]

self.rect = self.image.get\_rect()

self.rect.x = x

self.rect.y = y

def update(self, \*args):

pass

class Pole(Stuff):

def \_\_init\_\_(self, x, y):

Stuff.\_\_init\_\_(self, x, y, setup.GFX['tile\_set'],

[(263, 144, 2, 16)], c.BRICK\_SIZE\_MULTIPLIER)

class PoleTop(Stuff):

def \_\_init\_\_(self, x, y):

Stuff.\_\_init\_\_(self, x, y, setup.GFX['tile\_set'],

[(228, 120, 8, 8)], c.BRICK\_SIZE\_MULTIPLIER)

class Flag(Stuff):

def \_\_init\_\_(self, x, y):

Stuff.\_\_init\_\_(self, x, y, setup.GFX[c.ITEM\_SHEET],

[(128, 32, 16, 16)], c.SIZE\_MULTIPLIER)

self.state = c.TOP\_OF\_POLE

self.y\_vel = 5

def update(self):

if self.state == c.SLIDE\_DOWN:

self.rect.y += self.y\_vel

if self.rect.bottom >= 485:

self.state = c.BOTTOM\_OF\_POLE

class CastleFlag(Stuff):

def \_\_init\_\_(self, x, y):

Stuff.\_\_init\_\_(self, x, y, setup.GFX[c.ITEM\_SHEET],

[(129, 2, 14, 14)], c.SIZE\_MULTIPLIER)

self.y\_vel = -2

self.target\_height = y

def update(self):

if self.rect.bottom > self.target\_height:

self.rect.y += self.y\_vel

class Digit(pg.sprite.Sprite):

def \_\_init\_\_(self, image):

pg.sprite.Sprite.\_\_init\_\_(self)

self.image = image

self.rect = self.image.get\_rect()

class Score():

def \_\_init\_\_(self, x, y, score):

self.x = x

self.y = y

self.y\_vel = -3

self.create\_images\_dict()

self.score = score

self.create\_score\_digit()

self.distance = 130 if self.score == 1000 else 75

def create\_images\_dict(self):

self.image\_dict = {}

digit\_rect\_list = [(1, 168, 3, 8), (5, 168, 3, 8),

(8, 168, 4, 8), (0, 0, 0, 0),

(12, 168, 4, 8), (16, 168, 5, 8),

(0, 0, 0, 0), (0, 0, 0, 0),

(20, 168, 4, 8), (0, 0, 0, 0)]

digit\_string = '0123456789'

for digit, image\_rect in zip(digit\_string, digit\_rect\_list):

self.image\_dict[digit] = tools.get\_image(setup.GFX[c.ITEM\_SHEET],

\*image\_rect, c.BLACK, c.BRICK\_SIZE\_MULTIPLIER)

def create\_score\_digit(self):

self.digit\_group = pg.sprite.Group()

self.digit\_list = []

for digit in str(self.score):

self.digit\_list.append(Digit(self.image\_dict[digit]))

for i, digit in enumerate(self.digit\_list):

digit.rect = digit.image.get\_rect()

digit.rect.x = self.x + (i \* 10)

digit.rect.y = self.y

def update(self, score\_list):

for digit in self.digit\_list:

digit.rect.y += self.y\_vel

if (self.y - self.digit\_list[0].rect.y) > self.distance:

score\_list.remove(self)

def draw(self, screen):

for digit in self.digit\_list:

screen.blit(digit.image, digit.rect)

class Pipe(Stuff):

def \_\_init\_\_(self, x, y, width, height, type, name=c.MAP\_PIPE):

if type == c.PIPE\_TYPE\_HORIZONTAL:

rect = [(32, 128, 37, 30)]

else:

rect = [(0, 160, 32, 30)]

Stuff.\_\_init\_\_(self, x, y, setup.GFX['tile\_set'],

rect, c.BRICK\_SIZE\_MULTIPLIER)

self.name = name

self.type = type

if type != c.PIPE\_TYPE\_HORIZONTAL:

self.create\_image(x, y, height)

def create\_image(self, x, y, pipe\_height):

img = self.image

rect = self.image.get\_rect()

width = rect.w

height = rect.h

self.image = pg.Surface((width, pipe\_height)).convert()

self.rect = self.image.get\_rect()

self.rect.x = x

self.rect.y = y

top\_height = height // 2 + 3

bottom\_height = height // 2 - 3

self.image.blit(img, (0, 0), (0, 0, width, top\_height))

num = (pipe\_height - top\_height) // bottom\_height + 1

for i in range(num):

y = top\_height + i \* bottom\_height

self.image.blit(img, (0, y), (0, top\_height, width, bottom\_height))

self.image.set\_colorkey(c.BLACK)

def check\_ignore\_collision(self, level):

if self.type == c.PIPE\_TYPE\_HORIZONTAL:

return True

elif level.player.state == c.DOWN\_TO\_PIPE:

return True

return False

class Slider(Stuff):

def \_\_init\_\_(self, x, y, num, direction, range\_start, range\_end, vel, name=c.MAP\_SLIDER):

Stuff.\_\_init\_\_(self, x, y, setup.GFX[c.ITEM\_SHEET],

[(64, 128, 15, 8)], 2.8)

self.name = name

self.create\_image(x, y, num)

self.range\_start = range\_start

self.range\_end = range\_end

self.direction = direction

if self.direction == c.VERTICAL:

self.y\_vel = vel

else:

self.x\_vel = vel

def create\_image(self, x, y, num):

'''original slider image is short, we need to multiple it '''

if num == 1:

return

img = self.image

rect = self.image.get\_rect()

width = rect.w

height = rect.h

self.image = pg.Surface((width \* num, height)).convert()

self.rect = self.image.get\_rect()

self.rect.x = x

self.rect.y = y

for i in range(num):

x = i \* width

self.image.blit(img, (x, 0))

self.image.set\_colorkey(c.BLACK)

def update(self):

if self.direction == c.VERTICAL:

self.rect.y += self.y\_vel

if self.rect.y < -self.rect.h:

self.rect.y = c.SCREEN\_HEIGHT

self.y\_vel = -1

elif self.rect.y > c.SCREEN\_HEIGHT:

self.rect.y = -self.rect.h

self.y\_vel = 1

elif self.rect.y < self.range\_start:

self.rect.y = self.range\_start

self.y\_vel = 1

elif self.rect.bottom > self.range\_end:

self.rect.bottom = self.range\_end

self.y\_vel = -1

else:

self.rect.x += self.x\_vel

if self.rect.x < self.range\_start:

self.rect.x = self.range\_start

self.x\_vel = 1

elif self.rect.left > self.range\_end:

self.rect.left = self.range\_end

self.x\_vel = -1

## source/data

### source/data/maps

#### level\_1.json

{

"image\_name": "level\_1",

"maps": [

{

"start\_x": 0,

"end\_x": 9086,

"player\_x": 110,

"player\_y": 538

},

{

"start\_x": 9090,

"end\_x": 9890,

"player\_x": 80,

"player\_y": 60

},

{

"start\_x": 6740,

"end\_x": 9086,

"player\_x": 270,

"player\_y": 450

}

],

"ground": [

{

"x": 0,

"y": 538,

"width": 2953,

"height": 60

},

{

"x": 3048,

"y": 538,

"width": 635,

"height": 60

},

{

"x": 3819,

"y": 538,

"width": 2735,

"height": 60

},

{

"x": 6647,

"y": 538,

"width": 3250,

"height": 60

}

],

"pipe": [

{

"x": 1201,

"y": 451,

"width": 83,

"height": 84,

"type": 0

},

{

"x": 1629,

"y": 409,

"width": 83,

"height": 126,

"type": 0

},

{

"x": 1972,

"y": 366,

"width": 83,

"height": 170,

"type": 0

},

{

"x": 2444,

"y": 366,

"width": 83,

"height": 170,

"type": 1

},

{

"x": 6987,

"y": 451,

"width": 83,

"height": 84,

"type": 0

},

{

"x": 7673,

"y": 451,

"width": 83,

"height": 84,

"type": 0

},

{

"x": 9724,

"y": 451,

"width": 40,

"height": 84,

"type": 2

}

],

"step": [

{

"x": 5745,

"y": 495,

"width": 40,

"height": 44

},

{

"x": 5788,

"y": 452,

"width": 40,

"height": 44

},

{

"x": 5831,

"y": 409,

"width": 40,

"height": 44

},

{

"x": 5874,

"y": 366,

"width": 40,

"height": 176

},

{

"x": 6001,

"y": 366,

"width": 40,

"height": 176

},

{

"x": 6044,

"y": 408,

"width": 40,

"height": 44

},

{

"x": 6087,

"y": 452,

"width": 40,

"height": 44

},

{

"x": 6130,

"y": 495,

"width": 40,

"height": 44

},

{

"x": 6345,

"y": 495,

"width": 40,

"height": 44

},

{

"x": 6388,

"y": 452,

"width": 40,

"height": 44

},

{

"x": 6431,

"y": 409,

"width": 40,

"height": 44

},

{

"x": 6474,

"y": 366,

"width": 40,

"height": 44

},

{

"x": 6517,

"y": 366,

"width": 40,

"height": 176

},

{

"x": 6644,

"y": 366,

"width": 40,

"height": 176

},

{

"x": 6687,

"y": 408,

"width": 40,

"height": 44

},

{

"x": 6728,

"y": 452,

"width": 40,

"height": 44

},

{

"x": 6771,

"y": 495,

"width": 40,

"height": 44

},

{

"x": 7760,

"y": 495,

"width": 40,

"height": 44

},

{

"x": 7803,

"y": 452,

"width": 40,

"height": 44

},

{

"x": 7845,

"y": 409,

"width": 40,

"height": 44

},

{

"x": 7888,

"y": 366,

"width": 40,

"height": 44

},

{

"x": 7931,

"y": 323,

"width": 40,

"height": 44

},

{

"x": 7974,

"y": 280,

"width": 40,

"height": 44

},

{

"x": 8017,

"y": 237,

"width": 40,

"height": 44

},

{

"x": 8060,

"y": 194,

"width": 40,

"height": 44

},

{

"x": 8103,

"y": 194,

"width": 40,

"height": 360

},

{

"x": 8488,

"y": 495,

"width": 40,

"height": 44

},

{

"x": 9821,

"y": 64,

"width": 70,

"height": 530

}

],

"coin": [

{

"x": 9318,

"y": 369

},

{

"x": 9362,

"y": 369

},

{

"x": 9406,

"y": 369

},

{

"x": 9450,

"y": 369

},

{

"x": 9494,

"y": 369

},

{

"x": 9538,

"y": 369

},

{

"x": 9582,

"y": 369

},

{

"x": 9318,

"y": 284

},

{

"x": 9362,

"y": 284

},

{

"x": 9406,

"y": 284

},

{

"x": 9450,

"y": 284

},

{

"x": 9494,

"y": 284

},

{

"x": 9538,

"y": 284

},

{

"x": 9582,

"y": 284

},

{

"x": 9362,

"y": 198

},

{

"x": 9406,

"y": 198

},

{

"x": 9450,

"y": 198

},

{

"x": 9494,

"y": 198

},

{

"x": 9538,

"y": 198

}

],

"brick": [

{

"x": 0,

"y": 365,

"type": 2

},

{

"x": 858,

"y": 365,

"type": 0

},

{

"x": 944,

"y": 365,

"type": 0

},

{

"x": 1030,

"y": 365,

"type": 0

},

{

"x": 3299,

"y": 365,

"type": 0

},

{

"x": 3385,

"y": 365,

"type": 0

},

{

"x": 3430,

"y": 193,

"type": 0

},

{

"x": 3473,

"y": 193,

"type": 0

},

{

"x": 3473,

"y": 193,

"type": 0

},

{

"x": 3516,

"y": 193,

"type": 0

},

{

"x": 3559,

"y": 193,

"type": 0

},

{

"x": 3602,

"y": 193,

"type": 0

},

{

"x": 3645,

"y": 193,

"type": 0

},

{

"x": 3688,

"y": 193,

"type": 0

},

{

"x": 3731,

"y": 193,

"type": 0

},

{

"x": 3901,

"y": 193,

"type": 0

},

{

"x": 3944,

"y": 193,

"type": 0

},

{

"x": 3987,

"y": 193,

"type": 0

},

{

"x": 4030,

"y": 365,

"type": 1

},

{

"x": 4287,

"y": 365,

"type": 0

},

{

"x": 4330,

"y": 365,

"type": 2

},

{

"x": 5058,

"y": 365,

"type": 0

},

{

"x": 5187,

"y": 193,

"type": 0

},

{

"x": 5230,

"y": 193,

"type": 0

},

{

"x": 5273,

"y": 193,

"type": 0

},

{

"x": 5488,

"y": 193,

"type": 0

},

{

"x": 5574,

"y": 193,

"type": 0

},

{

"x": 5617,

"y": 193,

"type": 0

},

{

"x": 5531,

"y": 365,

"type": 0

},

{

"x": 5574,

"y": 365,

"type": 0

},

{

"x": 7202,

"y": 365,

"type": 0

},

{

"x": 7245,

"y": 365,

"type": 0

},

{

"x": 7331,

"y": 365,

"type": 0

},

{

"x": 9090,

"y": 64,

"type": 0,

"color": 1,

"brick\_num": 11,

"direction": 1

},

{

"x": 9310,

"y": 64,

"type": 0,

"color": 1,

"brick\_num": 7,

"direction": 0

},

{

"x": 9310,

"y": 406,

"type": 0,

"color": 1,

"brick\_num": 7,

"direction": 0

},

{

"x": 9310,

"y": 449,

"type": 0,

"color": 1,

"brick\_num": 7,

"direction": 0

},

{

"x": 9310,

"y": 492,

"type": 0,

"color": 1,

"brick\_num": 7,

"direction": 0

}

],

"box": [

{

"x": 135,

"y": 365,

"type": 101

},

{

"x": 225,

"y": 365,

"type": 102

},

{

"x": 315,

"y": 365,

"type": 103

},

{

"x": 405,

"y": 365,

"type": 104

},

{

"x": 495,

"y": 365,

"type": 105

},

{

"x": 720,

"y": 365,

"type": 1

},

{

"x": 901,

"y": 365,

"type": 3

},

{

"x": 987,

"y": 365,

"type": 1

},

{

"x": 943,

"y": 193,

"type": 1

},

{

"x": 2391,

"y": 366,

"type": 106

},

{

"x": 2341,

"y": 366,

"type": 106

},

{

"x": 2291,

"y": 366,

"type": 106

},

{

"x": 2241,

"y": 366,

"type": 106

},

{

"x": 2532,

"y": 366,

"type": 107

},

{

"x": 2582,

"y": 366,

"type": 106

},

{

"x": 2632,

"y": 366,

"type": 107

},

{

"x": 2682,

"y": 366,

"type": 106

},

{

"x": 3342,

"y": 365,

"type": 4

},

{

"x": 4030,

"y": 193,

"type": 1

},

{

"x": 4544,

"y": 365,

"type": 1

},

{

"x": 4672,

"y": 365,

"type": 1

},

{

"x": 4672,

"y": 193,

"type": 4

},

{

"x": 4800,

"y": 365,

"type": 1

},

{

"x": 5531,

"y": 193,

"type": 1

},

{

"x": 7288,

"y": 365,

"type": 1

}

],

"enemy": [

{

"0": [

{

"x": 1120,

"y": 538,

"direction": 0,

"type": 0,

"color": 0

}

]

},

{

"1": [

{

"x": 1920,

"y": 538,

"direction": 0,

"type": 0,

"color": 0

}

]

},

{

"2": [

{

"x": 2320,

"y": 538,

"direction": 0,

"type": 0,

"color": 0

},

{

"x": 2380,

"y": 538,

"direction": 0,

"type": 0,

"color": 0

}

]

},

{

"3": [

{

"x": 3640,

"y": 193,

"direction": 0,

"type": 0,

"color": 0

},

{

"x": 3700,

"y": 193,

"direction": 0,

"type": 0,

"color": 0

}

]

},

{

"4": [

{

"x": 4330,

"y": 538,

"direction": 0,

"type": 0,

"color": 0

}

]

},

{

"5": [

{

"x": 4700,

"y": 538,

"direction": 0,

"type": 1,

"color": 1

}

]

},

{

"6": [

{

"x": 4900,

"y": 538,

"direction": 0,

"type": 0,

"color": 0

},

{

"x": 4960,

"y": 538,

"direction": 0,

"type": 0,

"color": 0

}

]

},

{

"7": [

{

"x": 5300,

"y": 538,

"direction": 0,

"type": 0,

"color": 0

},

{

"x": 5360,

"y": 538,

"direction": 0,

"type": 0,

"color": 0

}

]

},

{

"8": [

{

"x": 5600,

"y": 538,

"direction": 0,

"type": 0,

"color": 0

},

{

"x": 5660,

"y": 538,

"direction": 0,

"type": 0,

"color": 0

}

]

},

{

"9": [

{

"x": 7550,

"y": 538,

"direction": 0,

"type": 0,

"color": 0

},

{

"x": 7610,

"y": 538,

"direction": 0,

"type": 0,

"color": 0

}

]

}

],

"checkpoint": [

{

"x": 510,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 0

},

{

"x": 1400,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 1

},

{

"x": 1740,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 2

},

{

"x": 3050,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 3

},

{

"x": 3750,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 4

},

{

"x": 4100,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 5

},

{

"x": 4300,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 6

},

{

"x": 4700,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 7

},

{

"x": 5000,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 8

},

{

"x": 6940,

"y": 0,

"width": 100,

"height": 600,

"type": 0,

"enemy\_groupid": 9

},

{

"x": 2460,

"y": 439,

"width": 50,

"height": 10,

"type": 6,

"map\_index": 1

},

{

"x": 8504,

"y": -10,

"width": 6,

"height": 600,

"type": 1

},

{

"x": 8780,

"y": 0,

"width": 10,

"height": 600,

"type": 2

},

{

"x": 2740,

"y": 360,

"width": 40,

"height": 12,

"type": 3

},

{

"x": 9731,

"y": 458,

"width": 10,

"height": 70,

"type": 4

},

{

"x": 9800,

"y": 458,

"width": 10,

"height": 70,

"type": 5,

"map\_index": 2

}

],

"flagpole": [

{

"x": 8470,

"y": 116,

"type": 0

},

{

"x": 8505,

"y": 97,

"type": 1

},

{

"x": 8505,

"y": 137,

"type": 1

},

{

"x": 8505,

"y": 177,

"type": 1

},

{

"x": 8505,

"y": 217,

"type": 1

},

{

"x": 8505,

"y": 257,

"type": 1

},

{

"x": 8505,

"y": 297,

"type": 1

},

{

"x": 8505,

"y": 337,

"type": 1

},

{

"x": 8505,

"y": 377,

"type": 1

},

{

"x": 8505,

"y": 417,

"type": 1

},

{

"x": 8505,

"y": 450,

"type": 1

},

{

"x": 8497,

"y": 97,

"type": 2

}

]

}

#### level\_2.json

{

"image\_name": "level\_2",

"maps": [

{

"start\_x": 0,

"end\_x": 8232,

"player\_x": 110,

"player\_y": 0

},

{

"start\_x": 8233,

"end\_x": 9860,

"player\_x": 150,

"player\_y": 451

},

{

"start\_x": 9862,

"end\_x": 10662,

"player\_x": 50,

"player\_y": 0

}

],

"ground": [

{

"x": 0,

"y": 540,

"width": 3426,

"height": 60

},

{

"x": 3560,

"y": 540,

"width": 1582,

"height": 60

},

{

"x": 5230,

"y": 410,

"width": 86,

"height": 190

},

{

"x": 5404,

"y": 540,

"width": 510,

"height": 60

},

{

"x": 6218,

"y": 540,

"width": 340,

"height": 60

},

{

"x": 6860,

"y": 540,

"width": 730,

"height": 190

},

{

"x": 7590,

"y": 540,

"width": 642,

"height": 60

},

{

"x": 8233,

"y": 540,

"width": 1627,

"height": 60

},

{

"x": 9862,

"y": 540,

"width": 800,

"height": 60

}

],

"pipe": [

{

"x": 4415,

"y": 410,

"width": 82,

"height": 130,

"type": 0

},

{

"x": 4672,

"y": 367,

"width": 82,

"height": 170,

"type": 0

},

{

"x": 4929,

"y": 453,

"width": 82,

"height": 86,

"type": 0

},

{

"x": 7114,

"y": 325,

"width": 104,

"height": 86,

"type": 2

},

{

"x": 7632,

"y": 410,

"width": 82,

"height": 130,

"type": 1

},

{

"x": 7802,

"y": 410,

"width": 104,

"height": 130,

"type": 1

},

{

"x": 7974,

"y": 410,

"width": 104,

"height": 130,

"type": 1

},

{

"x": 8360,

"y": 453,

"width": 82,

"height": 86,

"type": 0

},

{

"x": 10500,

"y": 453,

"width": 104,

"height": 86,

"type": 2

}

],

"step": [

{

"x": 730,

"y": 495,

"width": 40,

"height": 44

},

{

"x": 816,

"y": 451,

"width": 40,

"height": 88

},

{

"x": 900,

"y": 410,

"width": 40,

"height": 132

},

{

"x": 986,

"y": 367,

"width": 40,

"height": 176

},

{

"x": 1072,

"y": 367,

"width": 40,

"height": 176

},

{

"x": 1158,

"y": 410,

"width": 40,

"height": 132

},

{

"x": 1330,

"y": 410,

"width": 40,

"height": 132

},

{

"x": 1416,

"y": 451,

"width": 40,

"height": 88

},

{

"x": 5702,

"y": 496,

"width": 40,

"height": 44

},

{

"x": 5745,

"y": 452,

"width": 40,

"height": 88

},

{

"x": 5788,

"y": 410,

"width": 40,

"height": 132

},

{

"x": 5831,

"y": 367,

"width": 40,

"height": 176

},

{

"x": 5874,

"y": 367,

"width": 40,

"height": 176

},

{

"x": 7208,

"y": 66,

"width": 74,

"height": 344

},

{

"x": 8448,

"y": 496,

"width": 40,

"height": 44

},

{

"x": 8491,

"y": 452,

"width": 40,

"height": 88

},

{

"x": 8534,

"y": 410,

"width": 40,

"height": 132

},

{

"x": 8577,

"y": 367,

"width": 40,

"height": 176

},

{

"x": 8620,

"y": 324,

"width": 40,

"height": 220

},

{

"x": 8663,

"y": 281,

"width": 40,

"height": 264

},

{

"x": 8706,

"y": 237,

"width": 40,

"height": 308

},

{

"x": 8749,

"y": 194,

"width": 40,

"height": 352

},

{

"x": 8792,

"y": 194,

"width": 40,

"height": 352

},

{

"x": 9176,

"y": 496,

"width": 40,

"height": 44

},

{

"x": 10593,

"y": 66,

"width": 74,

"height": 474

}

],

"slider": [

{

"x": 5994,

"y": 0,

"num": 3,

"direction": 1,

"range\_start": -50,

"range\_end": 650,

"velocity": 1

},

{

"x": 5994,

"y": 260,

"num": 3,

"direction": 1,

"range\_start": -50,

"range\_end": 650,

"velocity": 1

},

{

"x": 6642,

"y": 0,

"num": 3,

"direction": 1,

"range\_start": -50,

"range\_end": 650,

"velocity": -1

},

{

"x": 6642,

"y": 260,

"num": 3,

"direction": 1,

"range\_start": -50,

"range\_end": 650,

"velocity": -1

}

],

"coin": [

{

"x": 1724,

"y": 328

},

{

"x": 1764,

"y": 200

},

{

"x": 1808,

"y": 200

},

{

"x": 1852,

"y": 200

},

{

"x": 1896,

"y": 200

},

{

"x": 1936,

"y": 328

},

{

"x": 2494,

"y": 328

},

{

"x": 2538,

"y": 328

},

{

"x": 2580,

"y": 328

},

{

"x": 2622,

"y": 328

},

{

"x": 2924,

"y": 328

},

{

"x": 3608,

"y": 200

},

{

"x": 3650,

"y": 200

},

{

"x": 3692,

"y": 200

},

{

"x": 3734,

"y": 200

},

{

"x": 3776,

"y": 200

},

{

"x": 10126,

"y": 328

},

{

"x": 10168,

"y": 328

},

{

"x": 10210,

"y": 328

},

{

"x": 10252,

"y": 328

},

{

"x": 10294,

"y": 328

},

{

"x": 10336,

"y": 328

},

{

"x": 10378,

"y": 328

},

{

"x": 10420,

"y": 328

},

{

"x": 10084,

"y": 500

},

{

"x": 10126,

"y": 500

},

{

"x": 10168,

"y": 500

},

{

"x": 10210,

"y": 500

},

{

"x": 10252,

"y": 500

},

{

"x": 10294,

"y": 500

},

{

"x": 10336,

"y": 500

},

{

"x": 10378,

"y": 500

},

{

"x": 10420,

"y": 500

}

],

"brick": [

{

"x": 0,

"y": 66,

"type": 0,

"color": 1,

"brick\_num": 11,

"direction": 1

},

{

"x": 254,

"y": 66,

"type": 0,

"color": 1,

"brick\_num": 83,

"direction": 0

},

{

"x": 3816,

"y": 66,

"type": 6,

"color": 1

},

{

"x": 3859,

"y": 66,

"type": 0,

"color": 1,

"brick\_num": 47,

"direction": 0

},

{

"x": 1244,

"y": 324,

"type": 1,

"color": 1

},

{

"x": 1672,

"y": 281,

"type": 0,

"color": 1,

"brick\_num": 3,

"direction": 1

},

{

"x": 1715,

"y": 367,

"type": 0,

"color": 1

},

{

"x": 1758,

"y": 281,

"type": 0,

"color": 1,

"brick\_num": 3,

"direction": 1

},

{

"x": 1801,

"y": 281,

"type": 0,

"color": 1

},

{

"x": 1844,

"y": 281,

"type": 0,

"color": 1

},

{

"x": 1887,

"y": 281,

"type": 0,

"color": 1,

"brick\_num": 3,

"direction": 1

},

{

"x": 1930,

"y": 367,

"type": 0,

"color": 1

},

{

"x": 1973,

"y": 367,

"type": 0,

"color": 1

},

{

"x": 1973,

"y": 324,

"type": 0,

"color": 1

},

{

"x": 1973,

"y": 281,

"type": 2,

"color": 1

},

{

"x": 2230,

"y": 195,

"type": 0,

"color": 1,

"brick\_num": 5,

"direction": 1

},

{

"x": 2273,

"y": 195,

"type": 0,

"color": 1,

"brick\_num": 5,

"direction": 1

},

{

"x": 2316,

"y": 109,

"type": 0,

"color": 1,

"brick\_num": 2,

"direction": 1

},

{

"x": 2316,

"y": 367,

"type": 0,

"color": 1,

"brick\_num": 3,

"direction": 1

},

{

"x": 2359,

"y": 109,

"type": 0,

"color": 1,

"brick\_num": 2,

"direction": 1

},

{

"x": 2359,

"y": 367,

"type": 0,

"color": 1,

"brick\_num": 3,

"direction": 1

},

{

"x": 2487,

"y": 109,

"type": 0,

"color": 1,

"brick\_num": 6,

"direction": 0

},

{

"x": 2487,

"y": 152,

"type": 0,

"color": 1,

"brick\_num": 6,

"direction": 0

},

{

"x": 2659,

"y": 195,

"type": 0,

"color": 1,

"brick\_num": 2,

"direction": 0

},

{

"x": 2659,

"y": 238,

"type": 0,

"color": 1,

"brick\_num": 2,

"direction": 0

},

{

"x": 2659,

"y": 281,

"type": 0,

"color": 1,

"brick\_num": 2,

"direction": 0

},

{

"x": 2659,

"y": 324,

"type": 0,

"color": 1,

"brick\_num": 2,

"direction": 0

},

{

"x": 2487,

"y": 367,

"type": 0,

"color": 1,

"brick\_num": 6,

"direction": 0

},

{

"x": 2830,

"y": 109,

"type": 0,

"color": 1,

"brick\_num": 3,

"direction": 0

},

{

"x": 2830,

"y": 152,

"type": 0,

"color": 1,

"brick\_num": 3,

"direction": 0

},

{

"x": 2873,

"y": 195,

"type": 0,

"color": 1,

"brick\_num": 5,

"direction": 1

},

{

"x": 2916,

"y": 367,

"type": 0,

"color": 1

},

{

"x": 2959,

"y": 367,

"type": 0,

"color": 1

},

{

"x": 2958,

"y": 324,

"type": 4,

"color": 1

},

{

"x": 3087,

"y": 195,

"type": 0,

"color": 1,

"brick\_num": 5,

"direction": 1

},

{

"x": 3130,

"y": 195,

"type": 0,

"color": 1,

"brick\_num": 5,

"direction": 1

},

{

"x": 3259,

"y": 195,

"type": 0,

"color": 1,

"brick\_num": 4,

"direction": 0

},

{

"x": 3259,

"y": 238,

"type": 0,

"color": 1,

"brick\_num": 4,

"direction": 0

},

{

"x": 3259,

"y": 367,

"type": 0,

"color": 1,

"brick\_num": 4,

"direction": 0

},

{

"x": 3602,

"y": 281,

"type": 0,

"color": 1,

"brick\_num": 6,

"direction": 0

},

{

"x": 3602,

"y": 324,

"type": 0,

"color": 1,

"brick\_num": 6,

"direction": 0

},

{

"x": 6218,

"y": 324,

"type": 0,

"color": 1,

"brick\_num": 5,

"direction": 0

},

{

"x": 6433,

"y": 324,

"type": 4,

"color": 1

},

{

"x": 6860,

"y": 410,

"type": 0,

"color": 1,

"brick\_num": 17,

"direction": 0

},

{

"x": 6860,

"y": 453,

"type": 0,

"color": 1,

"brick\_num": 17,

"direction": 0

},

{

"x": 6860,

"y": 496,

"type": 0,

"color": 1,

"brick\_num": 17,

"direction": 0

},

{

"x": 6903,

"y": 66,

"type": 0,

"color": 1,

"brick\_num": 7,

"direction": 0

},

{

"x": 7290,

"y": 66,

"type": 0,

"color": 1,

"brick\_num": 17,

"direction": 0

},

{

"x": 7290,

"y": 109,

"type": 0,

"color": 1,

"brick\_num": 7,

"direction": 0

},

{

"x": 7290,

"y": 152,

"type": 0,

"color": 1,

"brick\_num": 7,

"direction": 0

},

{

"x": 7290,

"y": 195,

"type": 0,

"color": 1,

"brick\_num": 7,

"direction": 0

},

{

"x": 7290,

"y": 238,

"type": 0,

"color": 1,

"brick\_num": 7,

"direction": 0

},

{

"x": 7290,

"y": 281,

"type": 0,

"color": 1,

"brick\_num": 7,

"direction": 0

},

{

"x": 7290,

"y": 324,

"type": 0,

"color": 1,

"brick\_num": 7,

"direction": 0

},

{

"x": 7290,

"y": 367,

"type": 0,

"color": 1,

"brick\_num": 7,

"direction": 0

},

{

"x": 8140,

"y": 66,

"type": 0,

"color": 1,

"brick\_num": 11,

"direction": 1

},

{

"x": 8183,

"y": 66,

"type": 0,

"color": 1,

"brick\_num": 11,

"direction": 1

},

{

"x": 9863,

"y": 66,

"type": 0,

"color": 1,

"brick\_num": 11,

"direction": 1

},

{

"x": 9992,

"y": 66,

"type": 0,

"color": 1,

"brick\_num": 12,

"direction": 0

},

{

"x": 9992,

"y": 109,

"type": 0,

"color": 1,

"brick\_num": 12,

"direction": 0

},

{

"x": 9992,

"y": 152,

"type": 0,

"color": 1,

"brick\_num": 12,

"direction": 0

},

{

"x": 9992,

"y": 195,

"type": 0,

"color": 1,

"brick\_num": 12,

"direction": 0

},

{

"x": 9992,

"y": 367,

"type": 0,

"color": 1,

"brick\_num": 12,

"direction": 0

},

{

"x": 10508,

"y": 66,

"type": 0,

"color": 1,

"brick\_num": 9,

"direction": 1

},

{

"x": 10551,

"y": 66,

"type": 0,

"color": 1,

"brick\_num": 9,

"direction": 1

}

],

"box": [

{

"x": 426,

"y": 367,

"type": 3

},

{

"x": 470,

"y": 367,

"type": 1

},

{

"x": 514,

"y": 367,

"type": 1

},

{

"x": 556,

"y": 367,

"type": 1

},

{

"x": 598,

"y": 367,

"type": 1

}

],

"enemy": [

{

"0": [

{

"x": 685,

"y": 540,

"direction": 0,

"type": 0,

"color": 1

},

{

"x": 730,

"y": 495,

"direction": 0,

"type": 0,

"color": 1

}

]

},

{

"1": [

{

"x": 1260,

"y": 540,

"direction": 0,

"type": 0,

"color": 1

}

]

},

{

"2": [

{

"x": 2040,

"y": 538,

"direction": 0,

"type": 1,

"color": 0

},

{

"x": 2100,

"y": 538,

"direction": 0,

"type": 1,

"color": 0

}

]

},

{

"3": [

{

"x": 2670,

"y": 538,

"direction": 0,

"type": 1,

"color": 0

},

{

"x": 2800,

"y": 538,

"direction": 0,

"type": 0,

"color": 1

},

{

"x": 2880,

"y": 538,

"direction": 0,

"type": 0,

"color": 1

}

]

},

{

"4": [

{

"x": 3130,

"y": 195,

"direction": 0,

"type": 0,

"color": 1

}

]

},

{

"5": [

{

"x": 3330,

"y": 367,

"direction": 0,

"type": 0,

"color": 1

},

{

"x": 3400,

"y": 367,

"direction": 0,

"type": 0,

"color": 1

}

]

},

{

"6": [

{

"x": 4246,

"y": 538,

"direction": 0,

"type": 0,

"color": 1

},

{

"x": 4306,

"y": 538,

"direction": 0,

"type": 0,

"color": 1

},

{

"x": 4366,

"y": 538,

"direction": 0,

"type": 0,

"color": 1

}

]

},

{

"7": [

{

"x": 4440,

"y": 490,

"direction": 0,

"type": 3,

"color": 0,

"range": 1,

"range\_start": 360,

"range\_end": 490

},

{

"x": 4696,

"y": 445,

"direction": 0,

"type": 3,

"color": 0,

"range": 1,

"range\_start": 315,

"range\_end": 445

},

{

"x": 4954,

"y": 535,

"direction": 0,

"type": 3,

"color": 0,

"range": 1,

"range\_start": 400,

"range\_end": 535

}

]

},

{

"8": [

{

"x": 4890,

"y": 538,

"direction": 0,

"type": 0,

"color": 1

}

]

},

{

"9": [

{

"x": 5786,

"y": 410,

"direction": 0,

"type": 0,

"color": 1

},

{

"x": 5846,

"y": 367,

"direction": 0,

"type": 0,

"color": 1

}

]

},

{

"10": [

{

"x": 6500,

"y": 538,

"direction": 0,

"type": 1,

"color": 2,

"range": 1,

"range\_start": 6220,

"range\_end": 6755

}

]

},

{

"11": [

{

"x": 8383,

"y": 535,

"direction": 0,

"type": 3,

"color": 0,

"range": 1,

"range\_start": 400,

"range\_end": 535

}

]

}

],

"checkpoint": [

{

"x": 110,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 0

},

{

"x": 728,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 1

},

{

"x": 1400,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 2

},

{

"x": 2070,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 3

},

{

"x": 2530,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 4

},

{

"x": 2730,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 5

},

{

"x": 3646,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 6

},

{

"x": 3800,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 7

},

{

"x": 4290,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 8

},

{

"x": 5186,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 9

},

{

"x": 5900,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 10

},

{

"x": 8383,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 11

},

{

"x": 7117,

"y": 330,

"width": 10,

"height": 80,

"type": 4

},

{

"x": 7187,

"y": 330,

"width": 10,

"height": 80,

"type": 5,

"map\_index": 1

},

{

"x": 7634,

"y": 540,

"width": 78,

"height": 10,

"type": 6,

"map\_index": 2

},

{

"x": 7804,

"y": 540,

"width": 78,

"height": 10,

"type": 6,

"map\_index": 2

},

{

"x": 7976,

"y": 540,

"width": 78,

"height": 10,

"type": 6,

"map\_index": 2

},

{

"x": 9195,

"y": -15,

"width": 5,

"height": 600,

"type": 1

},

{

"x": 9470,

"y": 0,

"width": 10,

"height": 600,

"type": 2

},

{

"x": 10502,

"y": 460,

"width": 10,

"height": 80,

"type": 4

},

{

"x": 10572,

"y": 460,

"width": 10,

"height": 80,

"type": 5,

"map\_index": 1

}

],

"flagpole": [

{

"x": 9156,

"y": 116,

"type": 0

}

]

}

#### level\_3.json

{

"image\_name": "level\_3",

"ground": [

{

"x": 0,

"y": 538,

"width": 686,

"height": 60

},

{

"x": 774,

"y": 496,

"width": 166,

"height": 36

},

{

"x": 1031,

"y": 368,

"width": 340,

"height": 36

},

{

"x": 1074,

"y": 406,

"width": 252,

"height": 194

},

{

"x": 1118,

"y": 196,

"width": 208,

"height": 36

},

{

"x": 1374,

"y": 496,

"width": 126,

"height": 36

},

{

"x": 1504,

"y": 325,

"width": 208,

"height": 36

},

{

"x": 1544,

"y": 360,

"width": 126,

"height": 240

},

{

"x": 1716,

"y": 153,

"width": 296,

"height": 36

},

{

"x": 1760,

"y": 190,

"width": 208,

"height": 410

},

{

"x": 2146,

"y": 538,

"width": 166,

"height": 36

},

{

"x": 2534,

"y": 538,

"width": 208,

"height": 36

},

{

"x": 2574,

"y": 198,

"width": 166,

"height": 36

},

{

"x": 2790,

"y": 538,

"width": 208,

"height": 36

},

{

"x": 3002,

"y": 366,

"width": 126,

"height": 36

},

{

"x": 3044,

"y": 406,

"width": 40,

"height": 194

},

{

"x": 3262,

"y": 238,

"width": 252,

"height": 36

},

{

"x": 3304,

"y": 274,

"width": 166,

"height": 326

},

{

"x": 4204,

"y": 452,

"width": 166,

"height": 36

},

{

"x": 4246,

"y": 488,

"width": 84,

"height": 112

},

{

"x": 4462,

"y": 282,

"width": 338,

"height": 36

},

{

"x": 4502,

"y": 318,

"width": 252,

"height": 282

},

{

"x": 4847,

"y": 538,

"width": 126,

"height": 60

},

{

"x": 4976,

"y": 368,

"width": 166,

"height": 36

},

{

"x": 5018,

"y": 406,

"width": 84,

"height": 194

},

{

"x": 5232,

"y": 368,

"width": 166,

"height": 36

},

{

"x": 5274,

"y": 406,

"width": 84,

"height": 194

},

{

"x": 5532,

"y": 538,

"width": 1508,

"height": 60

}

],

"step": [

{

"x": 5917,

"y": 366,

"width": 80,

"height": 176

},

{

"x": 6002,

"y": 280,

"width": 80,

"height": 264

},

{

"x": 6090,

"y": 194,

"width": 80,

"height": 352

},

{

"x": 6516,

"y": 495,

"width": 40,

"height": 44

}

],

"slider": [

{

"x": 2400,

"y": 338,

"num": 3,

"direction": 1,

"range\_start": 238,

"range\_end": 560

},

{

"x": 3590,

"y": 322,

"num": 3,

"direction": 0,

"range\_start": 3590,

"range\_end": 3750

},

{

"x": 3790,

"y": 366,

"num": 3,

"direction": 0,

"range\_start": 3790,

"range\_end": 3950

},

{

"x": 5440,

"y": 238,

"num": 3,

"direction": 0,

"range\_start": 5440,

"range\_end": 5600

}

],

"coin": [

{

"x": 1166,

"y": 156

},

{

"x": 1208,

"y": 156

},

{

"x": 1250,

"y": 156

},

{

"x": 1594,

"y": 70

},

{

"x": 1636,

"y": 70

},

{

"x": 2152,

"y": 240

},

{

"x": 2194,

"y": 240

},

{

"x": 2580,

"y": 156

},

{

"x": 2622,

"y": 156

},

{

"x": 2664,

"y": 156

},

{

"x": 2706,

"y": 156

},

{

"x": 3652,

"y": 198

},

{

"x": 3694,

"y": 198

},

{

"x": 3996,

"y": 156

},

{

"x": 4038,

"y": 156

},

{

"x": 4166,

"y": 156

},

{

"x": 4208,

"y": 156

},

{

"x": 4852,

"y": 496

},

{

"x": 4894,

"y": 496

},

{

"x": 4936,

"y": 496

},

{

"x": 5154,

"y": 198

},

{

"x": 5196,

"y": 198

}

],

"box": [

{

"x": 2530,

"y": 408,

"type": 3

}

],

"enemy": [

{

"0": [

{

"x": 1284,

"y": 196,

"direction": 0,

"type": 1,

"color": 2,

"range": 1,

"range\_start": 1114,

"range\_end": 1324

}

]

},

{

"1": [

{

"x": 1886,

"y": 152,

"direction": 0,

"type": 0,

"color": 0

},

{

"x": 1946,

"y": 152,

"direction": 0,

"type": 0,

"color": 0

}

]

},

{

"2": [

{

"x": 3160,

"y": 220,

"direction": 0,

"type": 2,

"color": 2,

"range": 1,

"range\_start": 200,

"range\_end": 440,

"is\_vertical": 1

}

]

},

{

"3": [

{

"x": 3458,

"y": 238,

"direction": 0,

"type": 0,

"color": 0,

"range": 1,

"range\_start": 3262,

"range\_end": 3508

}

]

},

{

"4": [

{

"x": 4740,

"y": 282,

"direction": 0,

"type": 1,

"color": 2,

"range": 1,

"range\_start": 4460,

"range\_end": 4798

}

]

},

{

"5": [

{

"x": 4880,

"y": 260,

"direction": 0,

"type": 2,

"color": 2,

"range": 1,

"range\_start": 230,

"range\_end": 490,

"is\_vertical": 1

}

]

},

{

"6": [

{

"x": 5836,

"y": 538,

"direction": 0,

"type": 1,

"color": 2,

"range": 1,

"range\_start": 5530,

"range\_end": 5908

}

]

}

],

"checkpoint": [

{

"x": 690,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 0

},

{

"x": 1292,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 1

},

{

"x": 2300,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 2

},

{

"x": 2870,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 3

},

{

"x": 4154,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 4

},

{

"x": 4466,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 5

},

{

"x": 5246,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 6

},

{

"x": 6536,

"y": -15,

"width": 6,

"height": 600,

"type": 1,

"enemy\_groupid": 0

},

{

"x": 6853,

"y": 0,

"width": 10,

"height": 600,

"type": 2,

"enemy\_groupid": 0

}

],

"flagpole": [

{

"x": 6497,

"y": 116,

"type": 0

},

{

"x": 8505,

"y": 97,

"type": 1

},

{

"x": 8505,

"y": 137,

"type": 1

},

{

"x": 8505,

"y": 177,

"type": 1

},

{

"x": 8505,

"y": 217,

"type": 1

},

{

"x": 8505,

"y": 257,

"type": 1

},

{

"x": 8505,

"y": 297,

"type": 1

},

{

"x": 8505,

"y": 337,

"type": 1

},

{

"x": 8505,

"y": 377,

"type": 1

},

{

"x": 8505,

"y": 417,

"type": 1

},

{

"x": 8505,

"y": 450,

"type": 1

},

{

"x": 8497,

"y": 97,

"type": 2

}

]

}

#### level\_4.json

{

"image\_name": "level\_4",

"maps": [

{

"start\_x": 0,

"end\_x": 6844,

"player\_x": 110,

"player\_y": 275

}

],

"ground": [

{

"x": 0,

"y": 409,

"width": 556,

"height": 191

},

{

"x": 0,

"y": 366,

"width": 215,

"height": 43

},

{

"x": 0,

"y": 323,

"width": 172,

"height": 43

},

{

"x": 0,

"y": 280,

"width": 129,

"height": 43

},

{

"x": 623,

"y": 409,

"width": 470,

"height": 191

},

{

"x": 1222,

"y": 409,

"width": 129,

"height": 191

},

{

"x": 1480,

"y": 409,

"width": 2956,

"height": 191

},

{

"x": 1480,

"y": 366,

"width": 1584,

"height": 43

},

{

"x": 4436,

"y": 537,

"width": 1031,

"height": 60

},

{

"x": 4952,

"y": 409,

"width": 170,

"height": 130

},

{

"x": 5252,

"y": 409,

"width": 213,

"height": 130

},

{

"x": 6024,

"y": 366,

"width": 126,

"height": 170

},

{

"x": 6024,

"y": 537,

"width": 820,

"height": 60

}

],

"step": [

{

"x": 0,

"y": 63,

"width": 1008,

"height": 127

},

{

"x": 966,

"y": 190,

"width": 41,

"height": 87

},

{

"x": 1008,

"y": 63,

"width": 556,

"height": 41

},

{

"x": 1566,

"y": 63,

"width": 1502,

"height": 170

},

{

"x": 1566,

"y": 234,

"width": 41,

"height": 41

},

{

"x": 2080,

"y": 234,

"width": 41,

"height": 43

},

{

"x": 2552,

"y": 234,

"width": 41,

"height": 43

},

{

"x": 2852,

"y": 234,

"width": 41,

"height": 43

},

{

"x": 3066,

"y": 63,

"width": 3778,

"height": 40

},

{

"x": 3238,

"y": 366,

"width": 41,

"height": 43

},

{

"x": 3409,

"y": 104,

"width": 41,

"height": 86

},

{

"x": 3581,

"y": 366,

"width": 41,

"height": 43

},

{

"x": 3752,

"y": 104,

"width": 41,

"height": 86

},

{

"x": 3924,

"y": 366,

"width": 41,

"height": 43

},

{

"x": 3580,

"y": 366,

"width": 41,

"height": 43

},

{

"x": 4138,

"y": 104,

"width": 297,

"height": 86

},

{

"x": 5252,

"y": 104,

"width": 213,

"height": 86

},

{

"x": 6067,

"y": 104,

"width": 84,

"height": 128

},

{

"x": 5468,

"y": 409,

"width": 540,

"height": 44

}

],

"slider": [

{

"x": 5750,

"y": 238,

"num": 2,

"direction": 0,

"range\_start": 5720,

"range\_end": 5930

}

],

"box": [

{

"x": 1266,

"y": 236,

"type": 3

}

],

"enemy": [

{

"0": [

{

"x": 1275,

"y": 422,

"direction": 0,

"type": 4,

"color": 0,

"num": 6

}

]

},

{

"1": [

{

"x": 2091,

"y": 249,

"direction": 0,

"type": 4,

"color": 0,

"num": 6

}

]

},

{

"2": [

{

"x": 2562,

"y": 249,

"direction": 0,

"type": 4,

"color": 0,

"num": 6

}

]

},

{

"3": [

{

"x": 2862,

"y": 249,

"direction": 0,

"type": 4,

"color": 0,

"num": 6

}

]

},

{

"4": [

{

"x": 3250,

"y": 380,

"direction": 0,

"type": 4,

"color": 0,

"num": 6

}

]

},

{

"5": [

{

"x": 3592,

"y": 380,

"direction": 0,

"type": 4,

"color": 0,

"num": 6

}

]

},

{

"6": [

{

"x": 3762,

"y": 163,

"direction": 0,

"type": 4,

"color": 0,

"num": 6

}

]

},

{

"7": [

{

"x": 5880,

"y": 405,

"direction": 0,

"type": 5,

"color": 0,

"range": 1,

"range\_start": 5750,

"range\_end": 5975

}

]

}

],

"checkpoint": [

{

"x": 577,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 0

},

{

"x": 1050,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 1

},

{

"x": 1700,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 2

},

{

"x": 2162,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 3

},

{

"x": 2550,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 4

},

{

"x": 2892,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 5

},

{

"x": 3062,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 6

},

{

"x": 3900,

"y": 0,

"width": 10,

"height": 600,

"type": 0,

"enemy\_groupid": 7

},

{

"x": 6026,

"y": 324,

"width": 40,

"height": 40,

"type": 7,

"enemy\_groupid": 0

},

{

"x": 6510,

"y": 0,

"width": 10,

"height": 600,

"type": 2,

"enemy\_groupid": 0

}

]

}

### source/data/player

#### mario.json

{

"image\_name": "player",

"image\_frames": {

"right\_small\_normal": [

{

"x": 0,

"y": 0,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 0,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 0,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 0,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 0,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 0,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 0,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 0,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 0,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 0,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 0,

"width": 208,

"height": 225

}

],

"right\_big\_normal": [

{

"x": 0,

"y": 225,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 225,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 225,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 225,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 225,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 225,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 225,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 225,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 225,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 225,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 225,

"width": 208,

"height": 225

}

],

"right\_big\_fire": [

{

"x": 0,

"y": 450,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 450,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 450,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 450,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 450,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 450,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 450,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 450,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 450,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 450,

"width": 208,

"height": 225

},

{

"x": 0,

"y": 450,

"width": 208,

"height": 225

}

]

},

"speed": {

"max\_walk\_speed": 6,

"max\_run\_speed": 12,

"max\_y\_velocity": 11,

"walk\_accel": 0.15,

"run\_accel": 0.3,

"jump\_velocity": -10.5

}

}

## source/states

### level.py

import json

import os

import pygame as pg

from .. import constants as c

from .. import setup, tools

from ..components import info, stuff, player, brick, box, enemy, coin

class Level(tools.State):

def \_\_init\_\_(self):

tools.State.\_\_init\_\_(self)

self.mixer = pg.mixer

self.sfx\_dict = None

self.player = None

def startup(self, current\_time, persist):

self.game\_info = persist

self.persist = self.game\_info

self.game\_info[c.CURRENT\_TIME] = current\_time

self.death\_timer = 0

self.castle\_timer = 0

self.moving\_score\_list = []

self.overhead\_info = info.Info(self.game\_info, c.LEVEL)

self.load\_map()

self.setup\_background()

self.setup\_maps()

self.ground\_group = self.setup\_collide(c.MAP\_GROUND)

self.step\_group = self.setup\_collide(c.MAP\_STEP)

self.setup\_pipe()

self.setup\_slider()

self.setup\_static\_coin()

self.setup\_brick\_and\_box()

self.setup\_player()

self.setup\_enemies()

self.setup\_checkpoints()

self.setup\_flagpole()

self.setup\_sprite\_groups()

if self.game\_info[c.LEVEL\_NUM] == 1:

self.bgm1\_sound()

if self.game\_info[c.LEVEL\_NUM] == 2:

self.close\_bgm1\_sound()

self.sfx\_dict['bgm0'].play(-1)

# todo: sound

def bgm1\_sound(self):

self.sfx\_dict = setup.SFX

self.bgm1 = self.sfx\_dict['关卡1']

self.bgm1.play(-1)

def close\_bgm1\_sound(self):

self.bgm1.stop()

def add\_time(self, n):

self.overhead\_info.time += n

def add\_life(self, n):

self.game\_info[c.LIVES] += n

def load\_map(self):

map\_file = 'level\_' + str(self.game\_info[c.LEVEL\_NUM]) + '.json'

file\_path = os.path.join('source', 'data', 'maps', map\_file)

f = open(file\_path)

self.map\_data = json.load(f)

f.close()

def setup\_background(self):

img\_name = self.map\_data[c.MAP\_IMAGE]

self.background = setup.GFX[img\_name]

self.bg\_rect = self.background.get\_rect()

self.background = pg.transform.scale(self.background,

(int(self.bg\_rect.width \* c.BACKGROUND\_MULTIPLER),

int(self.bg\_rect.height \* c.BACKGROUND\_MULTIPLER)))

self.bg\_rect = self.background.get\_rect()

self.level = pg.Surface((self.bg\_rect.w, self.bg\_rect.h)).convert()

self.viewport = setup.SCREEN.get\_rect(bottom=self.bg\_rect.bottom)

def setup\_maps(self):

self.map\_list = []

if c.MAP\_MAPS in self.map\_data:

for data in self.map\_data[c.MAP\_MAPS]:

self.map\_list.append((data['start\_x'], data['end\_x'], data['player\_x'], data['player\_y']))

self.start\_x, self.end\_x, self.player\_x, self.player\_y = self.map\_list[0]

else:

self.start\_x = 0

self.end\_x = self.bg\_rect.w

self.player\_x = 110

self.player\_y = c.GROUND\_HEIGHT

def change\_map(self, index, type):

self.start\_x, self.end\_x, self.player\_x, self.player\_y = self.map\_list[index]

self.viewport.x = self.start\_x

if type == c.CHECKPOINT\_TYPE\_MAP:

self.player.rect.x = self.viewport.x + self.player\_x

self.player.rect.bottom = self.player\_y

self.player.state = c.STAND

elif type == c.CHECKPOINT\_TYPE\_PIPE\_UP:

self.player.rect.x = self.viewport.x + self.player\_x

self.player.rect.bottom = c.GROUND\_HEIGHT

self.player.state = c.UP\_OUT\_PIPE

self.player.up\_pipe\_y = self.player\_y

def setup\_collide(self, name):

group = pg.sprite.Group()

if name in self.map\_data:

for data in self.map\_data[name]:

group.add(stuff.Collider(data['x'], data['y'],

data['width'], data['height'], name))

return group

def setup\_pipe(self):

self.pipe\_group = pg.sprite.Group()

if c.MAP\_PIPE in self.map\_data:

for data in self.map\_data[c.MAP\_PIPE]:

self.pipe\_group.add(stuff.Pipe(data['x'], data['y'],

data['width'], data['height'], data['type']))

def setup\_slider(self):

self.slider\_group = pg.sprite.Group()

if c.MAP\_SLIDER in self.map\_data:

for data in self.map\_data[c.MAP\_SLIDER]:

if c.VELOCITY in data:

vel = data[c.VELOCITY]

else:

vel = 1

self.slider\_group.add(stuff.Slider(data['x'], data['y'], data['num'],

data['direction'], data['range\_start'], data['range\_end'], vel))

def setup\_static\_coin(self):

self.static\_coin\_group = pg.sprite.Group()

if c.MAP\_COIN in self.map\_data:

for data in self.map\_data[c.MAP\_COIN]:

self.static\_coin\_group.add(coin.StaticCoin(data['x'], data['y']))

def setup\_brick\_and\_box(self):

self.coin\_group = pg.sprite.Group()

self.powerup\_group = pg.sprite.Group()

self.brick\_group = pg.sprite.Group()

self.brickpiece\_group = pg.sprite.Group()

if c.MAP\_BRICK in self.map\_data:

for data in self.map\_data[c.MAP\_BRICK]:

brick.create\_brick(self.brick\_group, data, self)

self.box\_group = pg.sprite.Group()

if c.MAP\_BOX in self.map\_data:

for data in self.map\_data[c.MAP\_BOX]:

if data['type'] == c.TYPE\_COIN:

self.box\_group.add(box.Box(data['x'], data['y'], data['type'], self.coin\_group))

else:

self.box\_group.add(box.Box(data['x'], data['y'], data['type'], self.powerup\_group))

def setup\_player(self):

if self.player is None:

self.player = player.Player(self.game\_info[c.PLAYER\_NAME])

else:

self.player.restart()

self.player.rect.x = self.viewport.x + self.player\_x

self.player.rect.bottom = self.player\_y

if c.DEBUG:

self.player.rect.x = self.viewport.x + c.DEBUG\_START\_X

self.player.rect.bottom = c.DEBUG\_START\_y

self.viewport.x = self.player.rect.x - 110

def setup\_enemies(self):

self.enemy\_group\_list = []

index = 0

for data in self.map\_data[c.MAP\_ENEMY]:

group = pg.sprite.Group()

for item in data[str(index)]:

group.add(enemy.create\_enemy(item, self))

self.enemy\_group\_list.append(group)

index += 1

def setup\_checkpoints(self):

self.checkpoint\_group = pg.sprite.Group()

for data in self.map\_data[c.MAP\_CHECKPOINT]:

if c.ENEMY\_GROUPID in data:

enemy\_groupid = data[c.ENEMY\_GROUPID]

else:

enemy\_groupid = 0

if c.MAP\_INDEX in data:

map\_index = data[c.MAP\_INDEX]

else:

map\_index = 0

self.checkpoint\_group.add(stuff.Checkpoint(data['x'], data['y'], data['width'],

data['height'], data['type'], enemy\_groupid, map\_index))

def setup\_flagpole(self):

self.flagpole\_group = pg.sprite.Group()

if c.MAP\_FLAGPOLE in self.map\_data:

for data in self.map\_data[c.MAP\_FLAGPOLE]:

if data['type'] == c.FLAGPOLE\_TYPE\_FLAG:

sprite = stuff.Flag(data['x'], data['y'])

self.flag = sprite

elif data['type'] == c.FLAGPOLE\_TYPE\_POLE:

sprite = stuff.Pole(data['x'], data['y'])

else:

sprite = stuff.PoleTop(data['x'], data['y'])

self.flagpole\_group.add(sprite)

def setup\_sprite\_groups(self):

self.dying\_group = pg.sprite.Group()

self.enemy\_group = pg.sprite.Group()

self.shell\_group = pg.sprite.Group()

self.ground\_step\_pipe\_group = pg.sprite.Group(self.ground\_group,

self.pipe\_group, self.step\_group, self.slider\_group)

self.player\_group = pg.sprite.Group(self.player)

def update(self, surface, keys, current\_time):

self.game\_info[c.CURRENT\_TIME] = self.current\_time = current\_time

self.handle\_states(keys)

self.draw(surface)

def handle\_states(self, keys):

self.update\_all\_sprites(keys)

def update\_all\_sprites(self, keys):

if self.player.dead:

self.player.update(keys, self.game\_info, self.powerup\_group)

if self.current\_time - self.death\_timer > 3000:

self.update\_game\_info()

self.done = True

elif self.player.state == c.IN\_CASTLE:

self.player.update(keys, self.game\_info, None)

self.flagpole\_group.update()

if self.current\_time - self.castle\_timer > 2000:

self.update\_game\_info()

self.done = True

elif self.in\_frozen\_state():

self.player.update(keys, self.game\_info, None)

self.check\_checkpoints()

self.update\_viewport()

self.overhead\_info.update(self.game\_info, self.player)

for score in self.moving\_score\_list:

score.update(self.moving\_score\_list)

else:

self.player.update(keys, self.game\_info, self.powerup\_group)

self.flagpole\_group.update()

self.check\_checkpoints()

self.slider\_group.update()

self.static\_coin\_group.update(self.game\_info)

self.enemy\_group.update(self.game\_info, self)

self.shell\_group.update(self.game\_info, self)

self.brick\_group.update()

self.box\_group.update(self.game\_info)

self.powerup\_group.update(self.game\_info, self)

self.coin\_group.update(self.game\_info)

self.brickpiece\_group.update()

self.dying\_group.update(self.game\_info, self)

self.update\_player\_position()

self.check\_for\_player\_death()

self.update\_viewport()

self.overhead\_info.update(self.game\_info, self.player)

for score in self.moving\_score\_list:

score.update(self.moving\_score\_list)

def check\_checkpoints(self):

checkpoint = pg.sprite.spritecollideany(self.player, self.checkpoint\_group)

if checkpoint:

if checkpoint.type == c.CHECKPOINT\_TYPE\_ENEMY:

group = self.enemy\_group\_list[checkpoint.enemy\_groupid]

self.enemy\_group.add(group)

elif checkpoint.type == c.CHECKPOINT\_TYPE\_FLAG:

self.player.state = c.FLAGPOLE

if self.player.rect.bottom < self.flag.rect.y:

self.player.rect.bottom = self.flag.rect.y

self.flag.state = c.SLIDE\_DOWN

self.update\_flag\_score()

elif checkpoint.type == c.CHECKPOINT\_TYPE\_CASTLE:

self.player.state = c.IN\_CASTLE

self.player.x\_vel = 0

self.castle\_timer = self.current\_time

self.flagpole\_group.add(stuff.CastleFlag(8745, 322))

elif (checkpoint.type == c.CHECKPOINT\_TYPE\_MUSHROOM and

self.player.y\_vel < 0):

mushroom\_box = box.Box(checkpoint.rect.x, checkpoint.rect.bottom - 40,

c.TYPE\_LIFEMUSHROOM, self.powerup\_group)

mushroom\_box.start\_bump(self.moving\_score\_list)

self.box\_group.add(mushroom\_box)

self.player.y\_vel = 7

self.player.rect.y = mushroom\_box.rect.bottom

self.player.state = c.FALL

elif checkpoint.type == c.CHECKPOINT\_TYPE\_PIPE:

self.player.state = c.WALK\_AUTO

elif checkpoint.type == c.CHECKPOINT\_TYPE\_PIPE\_UP:

self.change\_map(checkpoint.map\_index, checkpoint.type)

elif checkpoint.type == c.CHECKPOINT\_TYPE\_MAP:

self.change\_map(checkpoint.map\_index, checkpoint.type)

elif checkpoint.type == c.CHECKPOINT\_TYPE\_BOSS:

self.player.state = c.WALK\_AUTO

checkpoint.kill()

def update\_flag\_score(self):

base\_y = c.GROUND\_HEIGHT - 80

y\_score\_list = [(base\_y, 100), (base\_y - 120, 400),

(base\_y - 200, 800), (base\_y - 320, 2000),

(0, 5000)]

for y, score in y\_score\_list:

if self.player.rect.y > y:

self.update\_score(score, self.flag)

break

def update\_player\_position(self):

if self.player.state == c.UP\_OUT\_PIPE:

return

self.player.rect.x += round(self.player.x\_vel)

if self.player.rect.x < self.start\_x:

self.player.rect.x = self.start\_x

elif self.player.rect.right > self.end\_x:

self.player.rect.right = self.end\_x

self.check\_player\_x\_collisions()

if not self.player.dead:

self.player.rect.y += round(self.player.y\_vel)

self.check\_player\_y\_collisions()

def check\_player\_x\_collisions(self):

ground\_step\_pipe = pg.sprite.spritecollideany(self.player, self.ground\_step\_pipe\_group)

brick = pg.sprite.spritecollideany(self.player, self.brick\_group)

box = pg.sprite.spritecollideany(self.player, self.box\_group)

enemy = pg.sprite.spritecollideany(self.player, self.enemy\_group)

shell = pg.sprite.spritecollideany(self.player, self.shell\_group)

powerup = pg.sprite.spritecollideany(self.player, self.powerup\_group)

coin = pg.sprite.spritecollideany(self.player, self.static\_coin\_group)

if box:

self.adjust\_player\_for\_x\_collisions(box)

elif brick:

self.adjust\_player\_for\_x\_collisions(brick)

elif ground\_step\_pipe:

if (ground\_step\_pipe.name == c.MAP\_PIPE and

ground\_step\_pipe.type == c.PIPE\_TYPE\_HORIZONTAL):

return

self.adjust\_player\_for\_x\_collisions(ground\_step\_pipe)

elif powerup:

if powerup.type == c.TYPE\_MUSHROOM:

self.update\_score(1000, powerup, 0)

if not self.player.big:

self.player.y\_vel = -1

self.player.state = c.SMALL\_TO\_BIG

elif powerup.type == c.TYPE\_FIREFLOWER:

self.update\_score(1000, powerup, 0)

if not self.player.big:

self.player.state = c.SMALL\_TO\_BIG

elif self.player.big and not self.player.fire:

self.player.state = c.BIG\_TO\_FIRE

elif powerup.type == c.TYPE\_STAR:

self.update\_score(1000, powerup, 0)

self.player.invincible = True

elif powerup.type == c.TYPE\_LIFEMUSHROOM:

self.update\_score(500, powerup, 0)

self.add\_life(2)

elif powerup.type == c.EGGSHELL\_PROJ\_NUM[0]:

self.update\_score(1000, powerup, 0)

self.add\_time(30)

elif powerup.type == c.EGGSHELL\_PROJ\_NUM[1]:

self.update\_score(1000, powerup, 0)

self.add\_time(30)

elif powerup.type == c.EGGSHELL\_PROJ\_NUM[2]:

self.update\_score(1000, powerup, 0)

self.add\_time(30)

elif powerup.type == c.EGGSHELL\_PROJ\_NUM[3]:

self.update\_score(1000, powerup, 0)

self.add\_time(30)

elif powerup.type == c.EGGSHELL\_PROJ\_NUM[4]:

self.update\_score(1000, powerup, 0)

self.add\_time(30)

elif powerup.type == c.JUDGMENT\_PROJ\_NUM[0]:

self.update\_score(1000, powerup, 0)

self.add\_time(30)

elif powerup.type == c.JUDGMENT\_PROJ\_NUM[1]:

self.update\_score(1000, powerup, 0)

self.add\_time(-40)

# todo: add other powerup types

if powerup.type != c.TYPE\_FIREBALL:

powerup.kill()

elif enemy:

if self.player.invincible:

self.update\_score(100, enemy, 0)

self.move\_to\_dying\_group(self.enemy\_group, enemy)

direction = c.RIGHT if self.player.facing\_right else c.LEFT

enemy.start\_death\_jump(direction)

elif self.player.hurt\_invincible:

pass

elif self.player.big:

self.player.y\_vel = -1

self.player.state = c.BIG\_TO\_SMALL

else:

self.player.start\_death\_jump(self.game\_info)

self.death\_timer = self.current\_time

elif shell:

if shell.state == c.SHELL\_SLIDE:

if self.player.invincible:

self.update\_score(200, shell, 0)

self.move\_to\_dying\_group(self.shell\_group, shell)

direction = c.RIGHT if self.player.facing\_right else c.LEFT

shell.start\_death\_jump(direction)

elif self.player.hurt\_invincible:

pass

elif self.player.big:

self.player.y\_vel = -1

self.player.state = c.BIG\_TO\_SMALL

else:

self.player.start\_death\_jump(self.game\_info)

self.death\_timer = self.current\_time

else:

self.update\_score(400, shell, 0)

if self.player.rect.x < shell.rect.x:

self.player.rect.left = shell.rect.x

shell.direction = c.RIGHT

shell.x\_vel = 10

else:

self.player.rect.x = shell.rect.left

shell.direction = c.LEFT

shell.x\_vel = -10

shell.rect.x += shell.x\_vel \* 4

shell.state = c.SHELL\_SLIDE

elif coin:

self.update\_score(100, coin, 1)

coin.kill()

def adjust\_player\_for\_x\_collisions(self, collider):

if collider.name == c.MAP\_SLIDER:

return

if self.player.rect.x < collider.rect.x:

self.player.rect.right = collider.rect.left

else:

self.player.rect.left = collider.rect.right

self.player.x\_vel = 0

def check\_player\_y\_collisions(self):

ground\_step\_pipe = pg.sprite.spritecollideany(self.player, self.ground\_step\_pipe\_group)

enemy = pg.sprite.spritecollideany(self.player, self.enemy\_group)

shell = pg.sprite.spritecollideany(self.player, self.shell\_group)

# decrease runtime delay: when player is on the ground, don't check brick and box

if self.player.rect.bottom < c.GROUND\_HEIGHT:

brick = pg.sprite.spritecollideany(self.player, self.brick\_group)

box = pg.sprite.spritecollideany(self.player, self.box\_group)

brick, box = self.prevent\_collision\_conflict(brick, box)

else:

brick, box = False, False

if box:

self.adjust\_player\_for\_y\_collisions(box)

if box.type == c.JUDGMENT\_PROJ\_NUM[0]:

self.add\_time(-20)

elif box.type == c.JUDGMENT\_PROJ\_NUM[1]:

self.add\_time(-20)

# todo: add other box types

elif brick:

self.adjust\_player\_for\_y\_collisions(brick)

elif ground\_step\_pipe:

self.adjust\_player\_for\_y\_collisions(ground\_step\_pipe)

elif enemy:

if self.player.invincible:

self.update\_score(100, enemy, 0)

self.move\_to\_dying\_group(self.enemy\_group, enemy)

direction = c.RIGHT if self.player.facing\_right else c.LEFT

enemy.start\_death\_jump(direction)

elif (enemy.name == c.PIRANHA or

enemy.name == c.FIRESTICK or

enemy.name == c.FIRE\_KOOPA or

enemy.name == c.FIRE):

pass

elif self.player.y\_vel > 0:

self.update\_score(100, enemy, 0)

enemy.state = c.JUMPED\_ON

if enemy.name == c.GOOMBA:

self.move\_to\_dying\_group(self.enemy\_group, enemy)

elif enemy.name == c.KOOPA or enemy.name == c.FLY\_KOOPA:

self.enemy\_group.remove(enemy)

self.shell\_group.add(enemy)

self.player.rect.bottom = enemy.rect.top

self.player.state = c.JUMP

self.player.y\_vel = -7

elif shell:

if self.player.y\_vel > 0:

if shell.state != c.SHELL\_SLIDE:

shell.state = c.SHELL\_SLIDE

if self.player.rect.centerx < shell.rect.centerx:

shell.direction = c.RIGHT

shell.rect.left = self.player.rect.right + 5

else:

shell.direction = c.LEFT

shell.rect.right = self.player.rect.left - 5

self.check\_is\_falling(self.player)

self.check\_if\_player\_on\_IN\_pipe()

def prevent\_collision\_conflict(self, sprite1, sprite2):

if sprite1 and sprite2:

distance1 = abs(self.player.rect.centerx - sprite1.rect.centerx)

distance2 = abs(self.player.rect.centerx - sprite2.rect.centerx)

if distance1 < distance2:

sprite2 = False

else:

sprite1 = False

return sprite1, sprite2

def adjust\_player\_for\_y\_collisions(self, sprite):

if self.player.rect.top > sprite.rect.top:

if sprite.name == c.MAP\_BRICK:

self.check\_if\_enemy\_on\_brick\_box(sprite)

if sprite.state == c.RESTING:

if self.player.big and sprite.type == c.TYPE\_NONE:

sprite.change\_to\_piece(self.dying\_group)

else:

if sprite.type == c.TYPE\_COIN:

self.update\_score(200, sprite, 1)

sprite.start\_bump(self.moving\_score\_list)

elif sprite.name == c.MAP\_BOX:

self.check\_if\_enemy\_on\_brick\_box(sprite)

if sprite.state == c.RESTING:

if sprite.type == c.TYPE\_COIN:

self.update\_score(200, sprite, 1)

sprite.start\_bump(self.moving\_score\_list)

elif (sprite.name == c.MAP\_PIPE and

sprite.type == c.PIPE\_TYPE\_HORIZONTAL):

return

self.player.y\_vel = 7

self.player.rect.top = sprite.rect.bottom

self.player.state = c.FALL

else:

self.player.y\_vel = 0

self.player.rect.bottom = sprite.rect.top

if self.player.state == c.FLAGPOLE:

self.player.state = c.WALK\_AUTO

elif self.player.state == c.END\_OF\_LEVEL\_FALL:

self.player.state = c.WALK\_AUTO

else:

self.player.state = c.WALK

def check\_if\_enemy\_on\_brick\_box(self, brick):

brick.rect.y -= 5

enemy = pg.sprite.spritecollideany(brick, self.enemy\_group)

if enemy:

self.update\_score(100, enemy, 0)

self.move\_to\_dying\_group(self.enemy\_group, enemy)

if self.player.rect.centerx > brick.rect.centerx:

direction = c.RIGHT

else:

direction = c.LEFT

enemy.start\_death\_jump(direction)

brick.rect.y += 5

def in\_frozen\_state(self):

if (self.player.state == c.SMALL\_TO\_BIG or

self.player.state == c.BIG\_TO\_SMALL or

self.player.state == c.BIG\_TO\_FIRE or

self.player.state == c.DEATH\_JUMP or

self.player.state == c.DOWN\_TO\_PIPE or

self.player.state == c.UP\_OUT\_PIPE):

return True

else:

return False

def check\_is\_falling(self, sprite):

sprite.rect.y += 1

check\_group = pg.sprite.Group(self.ground\_step\_pipe\_group,

self.brick\_group, self.box\_group)

if pg.sprite.spritecollideany(sprite, check\_group) is None:

if (sprite.state == c.WALK\_AUTO or

sprite.state == c.END\_OF\_LEVEL\_FALL):

sprite.state = c.END\_OF\_LEVEL\_FALL

elif (sprite.state != c.JUMP and

sprite.state != c.FLAGPOLE and

not self.in\_frozen\_state()):

sprite.state = c.FALL

sprite.rect.y -= 1

def check\_for\_player\_death(self):

if (self.player.rect.y > c.SCREEN\_HEIGHT or

self.overhead\_info.time <= 0):

self.player.start\_death\_jump(self.game\_info)

self.death\_timer = self.current\_time

def check\_if\_player\_on\_IN\_pipe(self):

'''check if player is on the pipe which can go down in to it '''

self.player.rect.y += 1

pipe = pg.sprite.spritecollideany(self.player, self.pipe\_group)

if pipe and pipe.type == c.PIPE\_TYPE\_IN:

if (self.player.crouching and

self.player.rect.x < pipe.rect.centerx and

self.player.rect.right > pipe.rect.centerx):

self.player.state = c.DOWN\_TO\_PIPE

self.player.rect.y -= 1

def update\_game\_info(self):

# todo update bgm

if self.player.dead:

self.persist[c.LIVES] -= 1

self.close\_bgm1\_sound()

if self.persist[c.LIVES] == 0:

self.close\_bgm1\_sound()

self.next = c.GAME\_OVER

elif self.overhead\_info.time == 0:

self.close\_bgm1\_sound()

self.next = c.TIME\_OUT

elif self.player.dead:

self.close\_bgm1\_sound()

self.next = c.LOAD\_SCREEN

else:

self.close\_bgm1\_sound()

self.game\_info[c.LEVEL\_NUM] += 1

self.next = c.LOAD\_SCREEN

def update\_viewport(self):

right\_best = self.viewport.x + self.viewport.w \* (1 - 0.618)

left\_best = self.viewport.x + self.viewport.w \* 0.3

player\_center = self.player.rect.centerx

if (self.player.x\_vel > 0 and

player\_center >= right\_best and

self.viewport.right < self.end\_x):

self.viewport.x += round(self.player.x\_vel)

elif self.player.x\_vel < 0 and player\_center <= left\_best and self.viewport.x > self.start\_x:

self.viewport.x += round(self.player.x\_vel)

def move\_to\_dying\_group(self, group, sprite):

group.remove(sprite)

self.dying\_group.add(sprite)

def update\_score(self, score, sprite, coin\_num=0):

self.game\_info[c.SCORE] += score

self.game\_info[c.COIN\_TOTAL] += coin\_num

x = sprite.rect.x

y = sprite.rect.y - 10

self.moving\_score\_list.append(stuff.Score(x, y, score))

def draw(self, surface):

self.level.blit(self.background, self.viewport, self.viewport)

self.powerup\_group.draw(self.level)

self.brick\_group.draw(self.level)

self.box\_group.draw(self.level)

self.coin\_group.draw(self.level)

self.dying\_group.draw(self.level)

self.brickpiece\_group.draw(self.level)

self.flagpole\_group.draw(self.level)

self.shell\_group.draw(self.level)

self.enemy\_group.draw(self.level)

self.player\_group.draw(self.level)

self.static\_coin\_group.draw(self.level)

self.slider\_group.draw(self.level)

self.pipe\_group.draw(self.level)

for score in self.moving\_score\_list:

score.draw(self.level)

if c.DEBUG:

self.ground\_step\_pipe\_group.draw(self.level)

self.checkpoint\_group.draw(self.level)

surface.blit(self.level, (0, 0), self.viewport)

self.overhead\_info.draw(surface)

### load\_screen.py

import pygame as pg

from .. import constants as c

from .. import setup, tools

from ..components import info

class LoadScreen(tools.State):

def \_\_init\_\_(self):

tools.State.\_\_init\_\_(self)

self.screen1 = setup.GFX['load\_screen1']

self.screen2 = setup.GFX['load\_screen2']

self.time\_list = [16000 \* 1, 16000 \* 2, 16000 \* 2 + 2000, 16000 \* 2 + 2000 + 100, 16000 \* 2 + 2000 + 200]

def startup(self, current\_time, persist):

self.start\_time = current\_time

self.persist = persist

self.game\_info = self.persist

self.next = self.set\_next\_state()

info\_state = self.set\_info\_state()

self.overhead\_info = info.Info(self.game\_info, info\_state)

self.sound()

def set\_next\_state(self):

return c.LEVEL

def set\_info\_state(self):

return c.LOAD\_SCREEN

def update(self, surface, keys, current\_time):

# 如果按下enter键：

if keys[pg.K\_SPACE]:

self.close\_sound()

self.done = True

else:

if (current\_time - self.start\_time) < self.time\_list[0]:

surface.blit(self.screen1, (0, 0))

elif (current\_time - self.start\_time) < self.time\_list[1]:

surface.blit(self.screen2, (0, 0))

elif (current\_time - self.start\_time) < self.time\_list[2]:

surface.fill(c.BLACK)

self.overhead\_info.update(self.game\_info)

self.overhead\_info.draw(surface)

elif (current\_time - self.start\_time) < self.time\_list[3]:

surface.fill(c.BLACK)

elif (current\_time - self.start\_time) < self.time\_list[4]:

surface.fill((106, 150, 252))

else:

self.close\_sound()

self.done = True

def sound(self):

self.sfx\_dict = setup.SFX

self.sfx\_dict['加载《晴天》'].play()

def close\_sound(self):

self.sfx\_dict['加载《晴天》'].stop()

class GameOver(LoadScreen):

def \_\_init\_\_(self):

LoadScreen.\_\_init\_\_(self)

self.time\_list = [3000, 3200, 3235]

def set\_next\_state(self):

return c.MAIN\_MENU

def set\_info\_state(self):

return c.GAME\_OVER

class TimeOut(LoadScreen):

def \_\_init\_\_(self):

LoadScreen.\_\_init\_\_(self)

self.time\_list = [2400, 2600, 2635]

def set\_next\_state(self):

if self.persist[c.LIVES] == 0:

return c.GAME\_OVER

else:

return c.LOAD\_SCREEN

def set\_info\_state(self):

return c.TIME\_OUT

### main\_menu.py

import pygame as pg

from .. import constants as c

from .. import setup

from .. import tools

from ..components import info

class Menu(tools.State):

def \_\_init\_\_(self):

tools.State.\_\_init\_\_(self)

persist = {c.COIN\_TOTAL: 0,

c.SCORE: 0,

c.LIVES: 5,

c.TOP\_SCORE: 0,

c.CURRENT\_TIME: 0.0,

c.LEVEL\_NUM: 1,

c.PLAYER\_NAME: c.PLAYER\_MARIO}

self.startup(0.0, persist)

def startup(self, current\_time, persist):

self.next = c.LOAD\_SCREEN

self.persist = persist

self.game\_info = persist

self.overhead\_info = info.Info(self.game\_info, c.MAIN\_MENU)

self.setup\_background()

self.setup\_player()

self.setup\_cursor()

self.sound()

def sound(self):

self.sfx\_dict = setup.SFX

self.sfx\_dict['菜单《Something\_Just\_Like\_This》'].play(-1)

self.sfx\_dict['菜单《Something\_Just\_Like\_This》'].set\_volume(0.3)

def close\_sound(self):

self.sfx\_dict['菜单《Something\_Just\_Like\_This》'].stop()

def setup\_background(self):

self.background = setup.GFX['level\_1']

self.background\_rect = self.background.get\_rect()

self.background = pg.transform.scale(self.background,

(int(self.background\_rect.width \* c.BACKGROUND\_MULTIPLER),

int(self.background\_rect.height \* c.BACKGROUND\_MULTIPLER)))

self.viewport = setup.SCREEN.get\_rect(bottom=setup.SCREEN\_RECT.bottom)

self.image\_dict = {}

image = tools.get\_image(setup.GFX['title\_screen'], 1, 60, 176, 88,

(255, 0, 220), c.SIZE\_MULTIPLIER)

rect = image.get\_rect()

rect.x, rect.y = (170, 100)

self.image\_dict['GAME\_NAME\_BOX'] = (image, rect)

def setup\_player(self):

self.player\_list = []

player\_rect\_info = [(0, 0, 208, 225), (208, 0, 208, 225)]

for rect in player\_rect\_info:

image = tools.get\_image(setup.GFX['player'],

\*rect, c.BLACK, 2.5/4.5)

rect = image.get\_rect()

rect.x, rect.bottom = 110, c.GROUND\_HEIGHT

self.player\_list.append((image, rect))

self.player\_index = 0

def setup\_cursor(self):

self.cursor = pg.sprite.Sprite()

self.cursor.image = tools.get\_image(setup.GFX[c.ITEM\_SHEET], 24, 160, 8, 8, c.BLACK, 3)

rect = self.cursor.image.get\_rect()

rect.x, rect.y = (220, 358)

self.cursor.rect = rect

self.cursor.state = c.PLAYER1

def update(self, surface, keys, current\_time):

self.current\_time = current\_time

self.game\_info[c.CURRENT\_TIME] = self.current\_time

self.player\_image = self.player\_list[self.player\_index][0]

self.player\_rect = self.player\_list[self.player\_index][1]

self.update\_cursor(keys)

self.overhead\_info.update(self.game\_info)

surface.blit(self.background, self.viewport, self.viewport)

surface.blit(self.image\_dict['GAME\_NAME\_BOX'][0],

self.image\_dict['GAME\_NAME\_BOX'][1])

surface.blit(self.player\_image, self.player\_rect)

surface.blit(self.cursor.image, self.cursor.rect)

self.overhead\_info.draw(surface)

def update\_cursor(self, keys):

if self.cursor.state == c.PLAYER1:

self.cursor.rect.y = 358

if keys[pg.K\_DOWN]:

self.cursor.state = c.PLAYER2

self.player\_index = 1

self.game\_info[c.PLAYER\_NAME] = c.PLAYER\_LUIGI

elif self.cursor.state == c.PLAYER2:

self.cursor.rect.y = 403

if keys[pg.K\_UP]:

self.cursor.state = c.PLAYER1

self.player\_index = 0

self.game\_info[c.PLAYER\_NAME] = c.PLAYER\_MARIO

if keys[pg.K\_RETURN]:

self.reset\_game\_info()

self.close\_sound()

self.done = True

def reset\_game\_info(self):

self.game\_info[c.COIN\_TOTAL] = 0

self.game\_info[c.SCORE] = 0

self.game\_info[c.LIVES] = 3

self.game\_info[c.CURRENT\_TIME] = 0.0

self.game\_info[c.LEVEL\_NUM] = 1

self.persist = self.game\_info

## constants.py

DEBUG = False

DEBUG\_START\_X = 110

DEBUG\_START\_y = 538

SCREEN\_HEIGHT = 600

SCREEN\_WIDTH = 800

SCREEN\_SIZE = (SCREEN\_WIDTH, SCREEN\_HEIGHT)

ORIGINAL\_CAPTION = "工程经济学——工程人马里奥历险记"

## COLORS ##

# R G B

GRAY = (100, 100, 100)

NAVYBLUE = (60, 60, 100)

WHITE = (255, 255, 255)

RED = (255, 0, 0)

GREEN = (0, 255, 0)

FOREST\_GREEN = (31, 162, 35)

BLUE = (0, 0, 255)

SKY\_BLUE = (39, 145, 251)

YELLOW = (255, 255, 0)

ORANGE = (255, 128, 0)

PURPLE = (255, 0, 255)

CYAN = (0, 255, 255)

BLACK = (0, 0, 0)

NEAR\_BLACK = (19, 15, 48)

COMBLUE = (233, 232, 255)

GOLD = (255, 215, 0)

BGCOLOR = WHITE

SIZE\_MULTIPLIER = 2.5

BRICK\_SIZE\_MULTIPLIER = 2.69

BACKGROUND\_MULTIPLER = 1

GROUND\_HEIGHT = SCREEN\_HEIGHT - 62

GAME\_TIME\_OUT = 301

# STATES FOR ENTIRE GAME

MAIN\_MENU = 'main menu'

LOAD\_SCREEN = 'load screen'

TIME\_OUT = 'time out'

GAME\_OVER = 'game over'

LEVEL = 'level'

# MAIN MENU CURSOR STATES

PLAYER1 = '1 PLAYER GAME'

PLAYER2 = '2 PLAYER GAME'

# GAME INFO DICTIONARY KEYS

COIN\_TOTAL = 'coin total'

SCORE = 'score'

TOP\_SCORE = 'top score'

LIVES = 'lives'

CURRENT\_TIME = 'current time'

LEVEL\_NUM = 'level num'

PLAYER\_NAME = 'player name'

PLAYER\_MARIO = 'mario'

PLAYER\_LUIGI = 'luigi'

# MAP COMPONENTS

MAP\_IMAGE = 'image\_name'

MAP\_MAPS = 'maps'

SUB\_MAP = 'sub\_map'

MAP\_GROUND = 'ground'

MAP\_PIPE = 'pipe'

PIPE\_TYPE\_NONE = 0

PIPE\_TYPE\_IN = 1 # can go down in the pipe

PIPE\_TYPE\_HORIZONTAL = 2 # can go right in the pipe

MAP\_STEP = 'step'

MAP\_BRICK = 'brick'

BRICK\_NUM = 'brick\_num'

TYPE\_NONE = 0

TYPE\_COIN = 1

TYPE\_STAR = 2

MAP\_BOX = 'box'

TYPE\_MUSHROOM = 3

TYPE\_FIREFLOWER = 4

TYPE\_FIREBALL = 5

TYPE\_LIFEMUSHROOM = 6

MAP\_ENEMY = 'enemy'

ENEMY\_TYPE\_GOOMBA = 0

ENEMY\_TYPE\_KOOPA = 1

ENEMY\_TYPE\_FLY\_KOOPA = 2

ENEMY\_TYPE\_PIRANHA = 3

ENEMY\_TYPE\_FIRESTICK = 4

ENEMY\_TYPE\_FIRE\_KOOPA = 5

ENEMY\_RANGE = 'range'

MAP\_CHECKPOINT = 'checkpoint'

ENEMY\_GROUPID = 'enemy\_groupid'

MAP\_INDEX = 'map\_index'

CHECKPOINT\_TYPE\_ENEMY = 0

CHECKPOINT\_TYPE\_FLAG = 1

CHECKPOINT\_TYPE\_CASTLE = 2

CHECKPOINT\_TYPE\_MUSHROOM = 3

CHECKPOINT\_TYPE\_PIPE = 4 # trigger player to go right in a pipe

CHECKPOINT\_TYPE\_PIPE\_UP = 5 # trigger player to another map and go up out of a pipe

CHECKPOINT\_TYPE\_MAP = 6 # trigger player to go to another map

CHECKPOINT\_TYPE\_BOSS = 7 # defeat the boss

MAP\_FLAGPOLE = 'flagpole'

FLAGPOLE\_TYPE\_FLAG = 0

FLAGPOLE\_TYPE\_POLE = 1

FLAGPOLE\_TYPE\_TOP = 2

MAP\_SLIDER = 'slider'

HORIZONTAL = 0

VERTICAL = 1

VELOCITY = 'velocity'

MAP\_COIN = 'coin'

# COMPONENT COLOR

COLOR = 'color'

COLOR\_TYPE\_ORANGE = 0

COLOR\_TYPE\_GREEN = 1

COLOR\_TYPE\_RED = 2

# BRICK STATES

RESTING = 'resting'

BUMPED = 'bumped'

OPENED = 'opened'

# MUSHROOM STATES

REVEAL = 'reveal'

SLIDE = 'slide'

# Player FRAMES

PLAYER\_FRAMES = 'image\_frames'

RIGHT\_SMALL\_NORMAL = 'right\_small\_normal'

RIGHT\_BIG\_NORMAL = 'right\_big\_normal'

RIGHT\_BIG\_FIRE = 'right\_big\_fire'

# PLAYER States

STAND = 'standing'

WALK = 'walk'

JUMP = 'jump'

FALL = 'fall'

FLY = 'fly'

SMALL\_TO\_BIG = 'small to big'

BIG\_TO\_FIRE = 'big to fire'

BIG\_TO\_SMALL = 'big to small'

FLAGPOLE = 'flag pole'

WALK\_AUTO = 'walk auto' # not handle key input in this state

END\_OF\_LEVEL\_FALL = 'end of level fall'

IN\_CASTLE = 'in castle'

DOWN\_TO\_PIPE = 'down to pipe'

UP\_OUT\_PIPE = 'up out of pipe'

# PLAYER FORCES

PLAYER\_SPEED = 'speed'

WALK\_ACCEL = 'walk\_accel'

RUN\_ACCEL = 'run\_accel'

JUMP\_VEL = 'jump\_velocity'

MAX\_Y\_VEL = 'max\_y\_velocity'

MAX\_RUN\_SPEED = 'max\_run\_speed'

MAX\_WALK\_SPEED = 'max\_walk\_speed'

SMALL\_TURNAROUND = .35

JUMP\_GRAVITY = .30

GRAVITY = 1.00

# LIST of ENEMIES

GOOMBA = 'goomba'

KOOPA = 'koopa'

FLY\_KOOPA = 'fly koopa'

FIRE\_KOOPA = 'fire koopa'

FIRE = 'fire'

PIRANHA = 'piranha'

FIRESTICK = 'firestick'

# GOOMBA Stuff

LEFT = 'left'

RIGHT = 'right'

JUMPED\_ON = 'jumped on'

DEATH\_JUMP = 'death jump'

# KOOPA STUFF

SHELL\_SLIDE = 'shell slide'

# FLAG STATE

TOP\_OF\_POLE = 'top of pole'

SLIDE\_DOWN = 'slide down'

BOTTOM\_OF\_POLE = 'bottom of pole'

# FIREBALL STATE

FLYING = 'flying'

BOUNCING = 'bouncing'

EXPLODING = 'exploding'

# IMAGE SHEET

MAIN\_ENEMY\_SHEET = 'enemies'

ENEMY\_SHEET = 'smb\_enemies\_sheet'

ITEM\_SHEET = 'item\_objects'

# MY ADDITIONS

EGGSHELL\_PROJ = ['工', '程', '经', '济', '学']

EGGSHELL\_PROJ\_NUM = [101, 102, 103, 104, 105]

JUDGMENT\_PROJ = ['对', '错']

JUDGMENT\_PROJ\_NUM = [106, 107]

## main.py

from . import constants as c

from . import tools

from .states import main\_menu, load\_screen, level

def main():

game = tools.Control()

state\_dict = {c.MAIN\_MENU: main\_menu.Menu(),

c.LOAD\_SCREEN: load\_screen.LoadScreen(),

c.LEVEL: level.Level(),

c.GAME\_OVER: load\_screen.GameOver(),

c.TIME\_OUT: load\_screen.TimeOut()}

game.setup\_states(state\_dict, c.MAIN\_MENU)

game.main()

## setup.py

import os

import pygame as pg

from . import constants as c

from . import tools

pg.init()

pg.event.set\_allowed([pg.KEYDOWN, pg.KEYUP, pg.QUIT])

pg.display.set\_caption(c.ORIGINAL\_CAPTION)

SCREEN = pg.display.set\_mode(c.SCREEN\_SIZE)

SCREEN\_RECT = SCREEN.get\_rect()

GFX = tools.load\_all\_gfx(os.path.join("resources", "Graphics"))

MY\_GFX = tools.load\_all\_gfx(os.path.join("resources", "QuestionProj"))

SFX = tools.load\_all\_sfx(os.path.join("resources", "Sound"))

## tools.py

import os

from abc import abstractmethod

import pygame as pg

# import pyaudio

# import wave

# import numpy as np

# tmp = []

# TODO: 分贝控制

keybinding = {

'action': pg.K\_s,

'jump': pg.K\_a,

'left': pg.K\_LEFT,

'right': pg.K\_RIGHT,

'down': pg.K\_DOWN

}

class State():

def \_\_init\_\_(self):

self.start\_time = 0.0

self.current\_time = 0.0

self.done = False

self.next = None

self.persist = {}

@abstractmethod

def startup(self, current\_time, persist):

'''abstract method'''

def cleanup(self):

self.done = False

return self.persist

@abstractmethod

def update(sefl, surface, keys, current\_time):

'''abstract method'''

class Control():

def \_\_init\_\_(self):

self.screen = pg.display.get\_surface()

self.done = False

self.clock = pg.time.Clock()

self.fps = 60

self.current\_time = 0.0

self.keys = pg.key.get\_pressed()

self.state\_dict = {}

self.state\_name = None

self.state = None

def setup\_states(self, state\_dict, start\_state):

self.state\_dict = state\_dict

self.state\_name = start\_state

self.state = self.state\_dict[self.state\_name]

def update(self):

self.current\_time = pg.time.get\_ticks()

if self.state.done:

self.flip\_state()

self.state.update(self.screen, self.keys, self.current\_time)

def flip\_state(self):

previous, self.state\_name = self.state\_name, self.state.next

persist = self.state.cleanup()

self.state = self.state\_dict[self.state\_name]

self.state.startup(self.current\_time, persist)

def event\_loop(self):

for event in pg.event.get():

if event.type == pg.QUIT:

self.done = True

elif event.type == pg.KEYDOWN:

self.keys = pg.key.get\_pressed()

elif event.type == pg.KEYUP:

self.keys = pg.key.get\_pressed()

def main(self):

# CHUNK = 512

# FORMAT = pyaudio.paInt16

# CHANNELS = 1

# RATE = 48000

# RECORD\_SECONDS = 5

# WAVE\_OUTPUT\_FILENAME = "cache.wav"

# p = pyaudio.PyAudio()

# stream = p.open(format=FORMAT,

# channels=CHANNELS,

# rate=RATE,

# input=True,

# frames\_per\_buffer=CHUNK)

# print("开始缓存录音")

# frames = []

while not self.done:

self.event\_loop()

self.update()

pg.display.update()

self.clock.tick(self.fps)

# for i in range(2):

# data = stream.read(CHUNK)

# frames.append(data)

# audio\_data = np.fromstring(data, dtype=np.short)

# temp = np.max(audio\_data)

# tmp.append(temp)

# print('监听麦克风音量：',tmp[-1])

# stream.stop\_stream()

# stream.close()

# p.terminate()

# wf = wave.open(WAVE\_OUTPUT\_FILENAME, 'wb')

# wf.setnchannels(CHANNELS)

# wf.setsampwidth(p.get\_sample\_size(FORMAT))

# wf.setframerate(RATE)

# wf.writeframes(b''.join(frames))

# wf.close()

def get\_image(sheet, x, y, width, height, colorkey, scale):

image = pg.Surface([width, height])

rect = image.get\_rect()

image.blit(sheet, (0, 0), (x, y, width, height))

image.set\_colorkey(colorkey)

image = pg.transform.scale(image,

(int(rect.width \* scale),

int(rect.height \* scale)))

return image

def load\_all\_gfx(directory, colorkey=(0, 0, 0), accept=('.png', '.jpg', '.bmp', '.gif')):

graphics = {}

for pic in os.listdir(directory):

name, ext = os.path.splitext(pic)

if ext.lower() in accept:

img = pg.image.load(os.path.join(directory, pic))

if img.get\_alpha():

img = img.convert\_alpha()

else:

img = img.convert()

img.set\_colorkey(colorkey)

graphics[name] = img

return graphics

def load\_all\_sfx(directory, accept=('.wav', '.mpe', '.ogg', '.mdi', '.mp3')):

effects = {}

for fx in os.listdir(directory):

name, ext = os.path.splitext(fx)

if ext.lower() in accept:

effects[name] = pg.mixer.Sound(os.path.join(directory, fx))

return effects

# resource

## resource/Graphics

一些图片素材

## resource/QuestionProj

一些工程经济学的课程素材，如题目、知识点等等

## resource/Sound

一些音频素材