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

```
import pygame as pg
from source.main import main

if __name__ == '__main__':
    main()
    pg.quit()
```

2 source

2.1 source/components

2.1.1 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))
```

2.1.2 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()
```

2.1.3 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]
```

2.1.4 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
```

2.1.5 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)
```

2.1.6 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
```

2.1.7 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
```

2.1.8 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
```

2.2 source/data

2.2.1 source/data/maps

2.2.1.1 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
            }
          ]
        }
```

2.2.1.2 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
        }
    ]
```

```
    }
```

2.2.1.3 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
        }
    ]
}
```


2.2.1.4 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
        }
    ]
}
```


2.2.2 source/data/player

2.2.2.1 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
        },
        {
            "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
      }
```

```
}
```

2.3 source/states

2.3.1 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)
```

2.3.2 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
```

2.3.3 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
```

2.4 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]
```

2.5 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()
```

2.6 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"))
```

2.7 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
```

3 resource

3.1 resource/Graphics

一些图片素材

3.2 resource/QuestionProj

一些工程经济学的课程素材，如题目、知识点等等

3.3 resource/Sound

一些音频素材