Run! Starfish!

刘海洋 黄思彧 金晨忻 孙泽宇

1 Introduction

1.1 The background of platform games

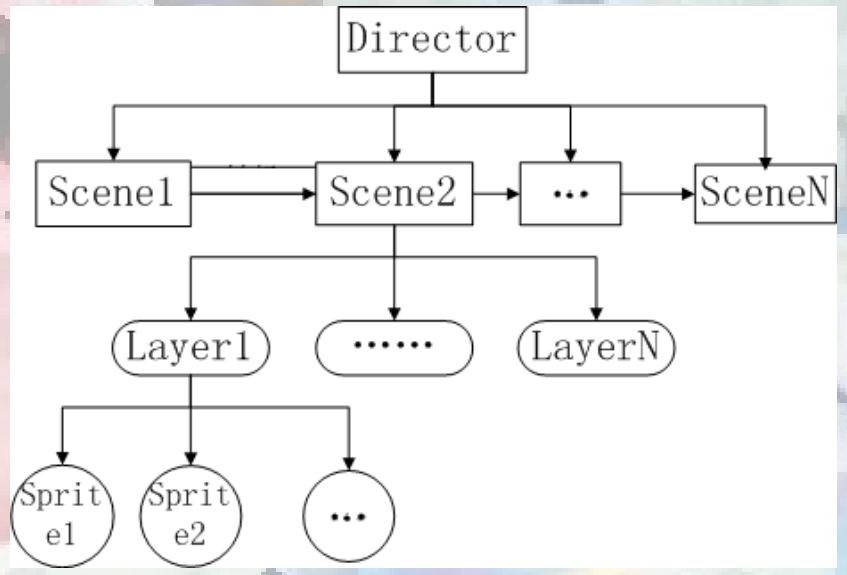
1.2 The libraries we have used

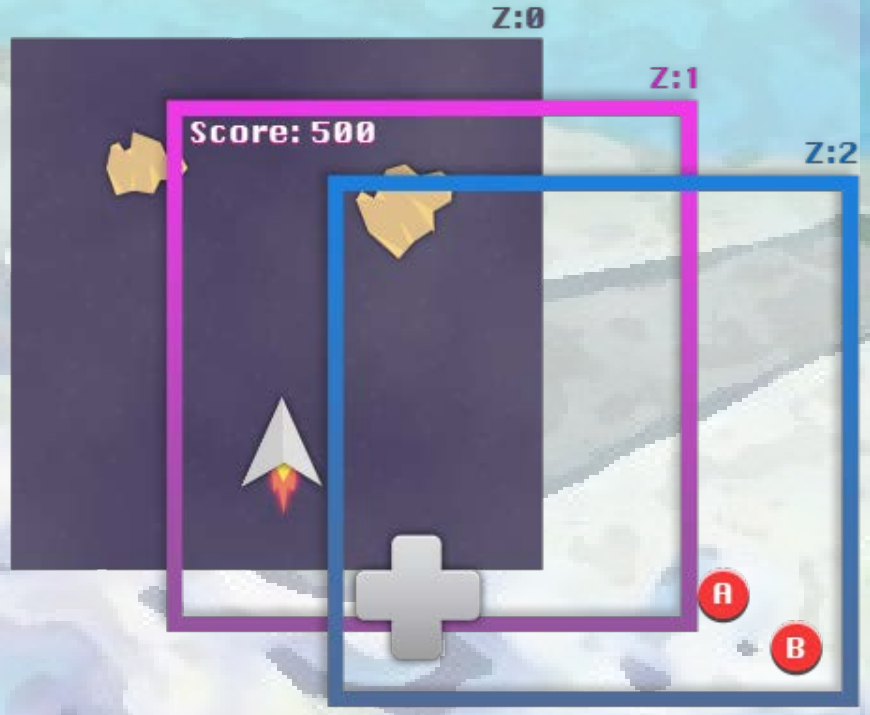
1.1 The background of platform games

- A video game genre and subgenre of action game.
- Control a character or avatar to jump between suspended platforms and avoid obstacles.
- Other acrobatic maneuvers like swinging or bouncing.
- Endless running games----get as far as possible.

1.2 The libraries we have used----cocos2d

- An open source software framework.
- To build games, apps and other cross platform GUI based interactive programs.
- Scene graphs
- Layers
- Sprites
- Director





1.2 The libraries we have used----PyAudio

- PyAudio provides Python bindings for PortAudio, the cross-platform audio I/O library.

- Can easily use Python to play and record audio on a variety of platforms.

2 Rules & Objectives

Rules

1. When the game starts, a starfish falls onto the first sponge.
2. The player makes sounds to control the starfish.
3. When the sound reaches lever 1 (the lower lever), the starfish will move forward.
4. When the sound reaches level 2 (the higher level), the starfish will jump up and then drop down.
5. The player cannot make sounds to make the starfish jump higher again when it’s in the air.
6. When it falls off the sponge, the game is over.

Objective

- The objective of the game is to get as many scores as possible.
- The high score will be recorded on the interface.

3 Game Demo

Game demo

4 Development process

4.1 Sponge

4.2 Starfish

4.3 Main game

Sponge

```
class Sponge(cocos.sprite.Sprite):
    def __init__(self, x):
        super().__init__('sponge.png')
        self.image_anchor = 0, 0
        # The first sponge:
        if x == 0:
            self.position = 0, 0
            # Additional horizontal-only scale of the sprite
            self.scale_x = 5
            # Additional vertical-only scale of the sprite
            self.scale_y = 1
        # Others:
        else:
            self.position = x + 50 + random.random() * 200, 0
            self.scale_x = 0.5 + random.random() * 1.5# between 0.5 and 2
            self.scale_y = 0.5 + random.random()  # between 0.5 and 1.5
```

- **Subclass Sprite as Sponge.**
- **Create sponges with random position and size except the first one.**

Starfish

```python
class StarFish(cocos.sprite.Sprite):
    def __init__(self):
        super().__init__('starfish.png')
        self.can_jump = False
        self.speed = 0 # the speed of falling.
        self.image_anchor = 0, 0
        self.position = 100, 300 # the initial position.
        self.schedule(self.update) # update
```

- Subclass Sprite as Starfish.
- Set the initial position and falling speed and it is set to update automatically.
- Can_jump gets Boolean value, indicating whether it can jump.

Jump & Land

```
def jump(self, h):
    if self.can_jump:
        self.y += 0.5
        self.speed -=  min(h, 8)
        self.can_jump = False

def land(self, y):
    if self.y > y - 30:
        self.can_jump = True
        self.speed = 0
        self.y = y
```

- Define two functions for starfish to jump and land.
- Change the speed and Boolean value of can_jump accrodingly

Update & Reset

```python
def update(self, dt):
    self.speed += 9.8 * dt
    self.y -= self.speed
    if self.y < -80:
        self.reset()

def reset(self):
    self.parent.reset()
    self.can_jump = False
    self.speed = 0
    self.position = 100, 300
```

- These two functions are used to update the starfish per frame(帧).
- When it fall off the sponge, game is over and then it will be reset.

Main game

```
class Game(cocos.layer.Layer):
    def __init__(self):
        super().__init__()

        self.bgm = Sound('bgm.wav')
        self.bgm.play(-1)

        self.bg = cocos.sprite.Sprite('background.png')
        self.bg.position = 320, 240
        self.add(self.bg)
```

- Subclass Layer as Game
- Set the background picture and music

Score boards

- Create two text labels to record score and high score.

```
BLACK = (0, 0, 0, 255)
```

```
# Record score
self.score = 0
# Create a Text Label to record score.
self.txt_score = cocos.text.Label(u'Score: 0',
                                  font_name='Times New Roman',
                                  font_size=24,
                                  color=BLACK)
self.txt_score.position = 460, 400
self.add(self.txt_score)
# Record the high score.
self.high_score  = 0
# Create a Text Label to record high score.
self.txt_high_score = cocos.text.Label(u'High Score: 0',
                                  font_name='Times New Roman',
                                  font_size=24,
                                  color=BLACK)
self.txt_high_score.position = 386, 440
self.add(self.txt_high_score)
```

Create sponges

- Create a CocosNode floor.
- Create a row of sponges and add them to floor,

```
# Create a CocosNode floor.
self.floor = cocos.cocosnode.CocosNode()
self.add(self.floor)


# Create Sponge instances and add them to floor.
x = 0
for i in range(100):
    b = Sponge(x)
    self.floor.add(b)
    x = b.x + b.width
```

Volume bar & Starfish

```
self.volumebar = cocos.sprite.Sprite('black.png')
self.volumebar.image_anchor = 0, 0
self.volumebar.position = 20, 450
self.volumebar.scale_y = 0.1
self.add(self.volumebar)

self.starfish = StarFish()
self.add(self.starfish)
```

- **Create volume bar and starfish on the interface.**

Voice input

```
NUM_SAMPLES = 1000
```

```
pa = PyAudio()
# store the stream input.
self.stream = pa.open(format=paInt16,
                      channels=1,
                      rate=44100,
                      input=True,
                      frames_per_buffer=NUM_SAMPLES)
# Parameters:
# format - Sampling size and format.
# channels - Number of channels
# rate - Sampling rate
# input - Specifies whether this is an input stream
# frames_per_buffer - Specifies the number of frames per buffer.

# Update the stream.
self.schedule(self.update)
```

- Get the voice input
- Update the voice stream automatically

Voice translation

- Get the max volume k from the stream.
- Change the scale of volume bar
- According to k, decide whether move forward and jump.

```
LEVEL_1 = 2000
LEVEL_2 = 10000
```

```
def update(self, dt): # dt=1/fps (seconds)
    # Read samples from the stream.
    # num_frames - The number of frames to read.
    string_audio_data = self.stream.read(NUM_SAMPLES)
    # struct.unpack(fmt, buffer):
    # Unpack from the buffer according to the formal string fmt,
    # Return a tuple.
    k = max(struct.unpack('1000h', string_audio_data)) #
    self.volumebar.scale_x = k / 10000.0
    # Determine moving and jumping
    if k > LEVEL_1:
        self.floor.x -= 150 * dt
    if k > LEVEL_2:
        self.starfish.jump((k - LEVEL_2) / 1000.0)
    self.collide()
```

Collision

- Decide whether starfish land on the sponge.
- Update score and high score

```
def collide(self):
    L = []
    px = self.starfish.x - self.floor.x
    # get_children: Return a list with the node's children
    for s in self.floor.get_children():
        L.append(s)
        if s.x <= px + self.starfish.width  and px <= s.x + s.width:
            if self.starfish.y < s.height:
                self.starfish.land(s.height)
                self.score = L.index(s)
                self.txt_score.element.text = u'Score: %d' % self.score
                if self.score > self.high_score:
                    self.high_score =self.score
                    self.txt_high_score.element.text = u'High Score: %d' % self.high_score
                break
```

Game over

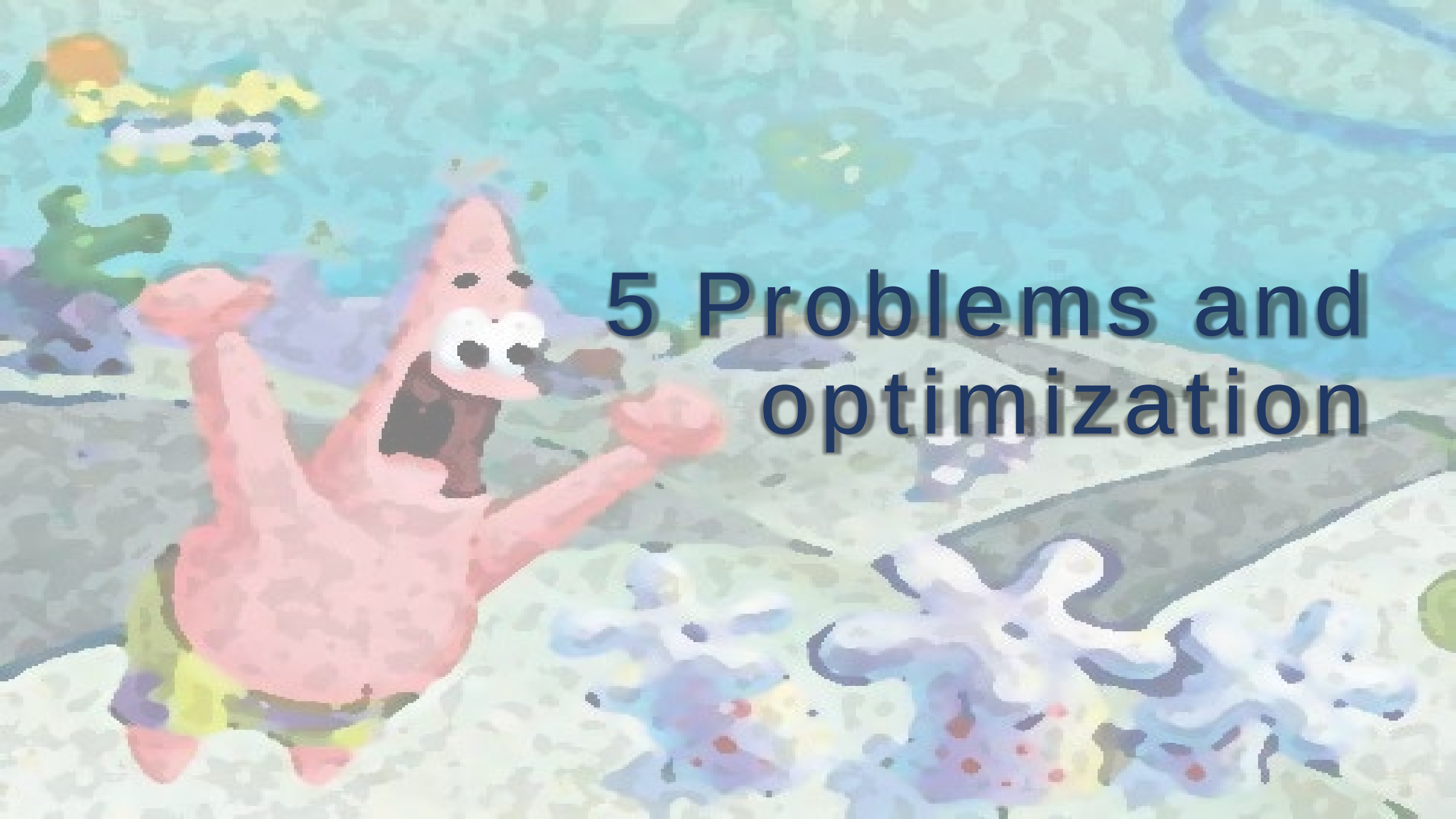
- Reset the game
- Change background music
- Suspend the picture

```
def reset(self):
    self.bgm.stop()
    game_over = Sound('over.wav')
    game_over.play()
    sleep(3) # pause for 3 seconds.
    game_over.stop()
    self.bgm.play(-1)
    self.floor.x = 0
    self.score = 0
    self.txt_score.element.text = u'Score: 0'
```

Run the game

- Initialize director and mixer
- Add Game to a scene
- Run the game

```
# Initialize the Director (which create a 640*480 window)and Mixer.
cocos.director.director.init(caption="Run! Starfish")
mixer.init()
# Then we create a Scene that contains the Game layer as a child.
main_scene = cocos.scene.Scene(Game())
# And finally we run the scene.
cocos.director.director.run(main_scene)
```



5 Problems and optimization

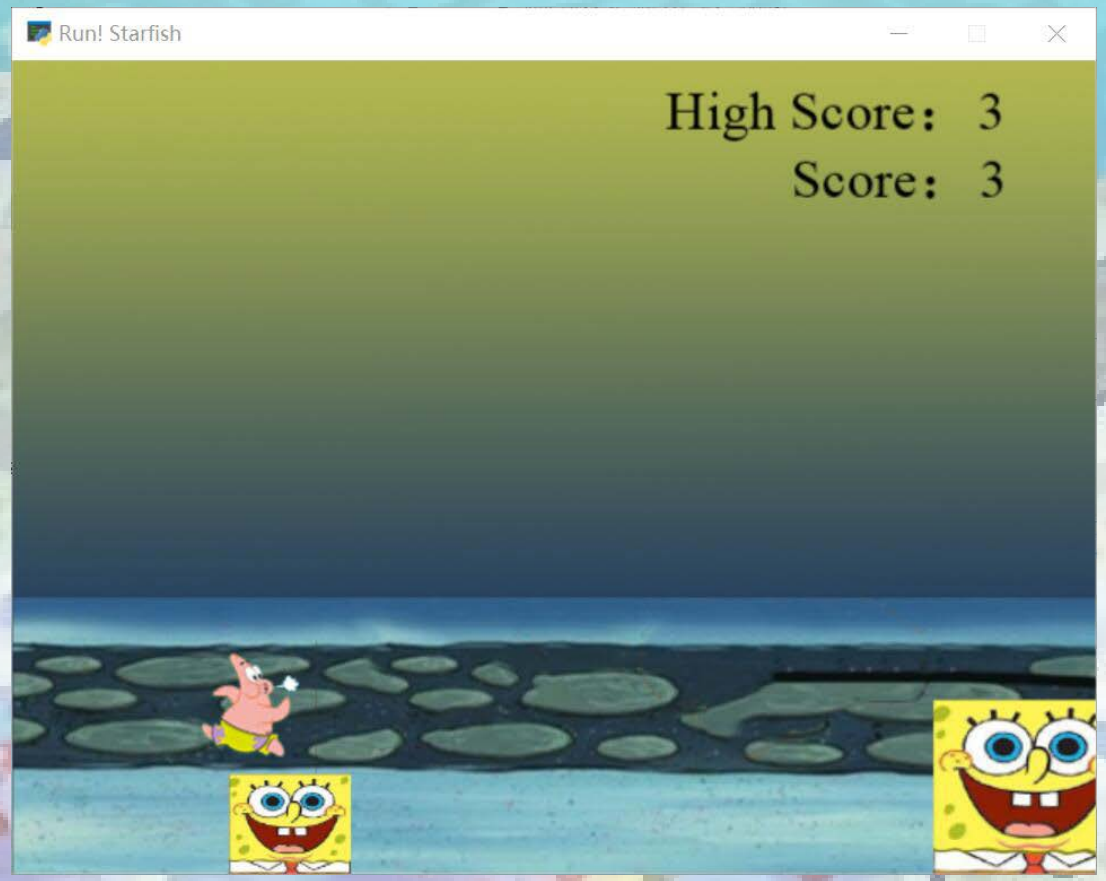
Problems and optimization

- Distance tested time and again.
- Never be too accurate.
- Details (prohibition of the second jump in the air, volume setting in different situations, etc.).
- User-friendly interaction environment

6 Future Development

Future development

- **Inappropriate distance difficult to cover**



Future development

- Interference of noise
- Potential of improvement: Machine Learning (Classification)

Future development

- Level up: Interaction with others
- Level up: Time limitation & Length Limitation

Thanks for Watching