프로젝트 완료 보고서

게임명 : 우주에서 살아남기

시현 영상 링크 https://youtu.be/Y1aevWBQUtc

깃허브 주소 https://github.com/kjyook/raspberrypi.git

플레이 방법

- 조이스틱을 움직여 나의 캐릭터를 위 아래로 움직일 수 있다
- 조이스틱을 오른쪽으로 움직이면 캐릭터를 향해 장애물, 아이템들이 빠르게 날라온다
- 조이스틱을 왼쪽으로 움직이면 캐릭터를 향해 장애물, 아이템들이 느리게 날라온다
- 일정한 시간마다 랜덤한 위치에서 나오는 아이템과 장애물을 획득하고 피하며 점수를 쌓아야 한다
- 장애물과 캐릭터가 부딪한다면 바로 죽는다
- 캐릭터가 아이템 (코인) 을 획득하면 총알을 하나 획득한다
- 가지고 있는 총알의 개수만큼 6번 버튼 (joystick.button_B) 를 누르면 총알을 진행방향 (오른쪽) 으로 발사한다
- 총알이 장애물과 부딪히면 장애물이 파괴되며 점수를 5점 획득한다
- 장애물이 캐릭터와 부딪히지 못하고 화면 밖으로 나가면 점수 1점을 시작한다
- 게임이 시작된 후 1분이 지나고 점수가 80점이 넘지 못했으면 패배, 80점을 넘었다면 게임을 끝 낼 수 있는 황금 키가 등장한다
- 황금키를 획득한다면 (부딪힌다면) → 게임 승리, 획득하지 못하면 → 게임 패배

source code 구조 및 동작 설명

1. **물체들의 움직임**

```
from digitalio import DigitalInOut, Direction
from adafruit_rgb_display import st7789
import board

class Joystick:
    def __init__(self):
        self.cs_pin = DigitalInOut(board.CE0)
        self.dc_pin = DigitalInOut(board.D25)
        self.reset_pin = DigitalInOut(board.D24)
        self.BAUDRATE = 24000000

        self.spi = board.SPI()
        self.disp = st7789.ST7789(
            self.spi,
            height=240,
            y_offset=80,
            rotation=180,
            cs=self.cs_pin,
            dc=self.dc_pin,
            rst=self.reset_pin,
            baudrate=self.BAUDRATE,
        )

        # Input pins:
        #버튼 5,6 + 조이스틱 -> l,r,u,d
        self.button_A = DigitalInOut(board.D5)
        self.button_A.direction = Direction.INPUT

        self.button_B = DigitalInOut(board.D6)
        self.button_B.direction = Direction.INPUT
```

```
        self.button_L = DigitalInOut(board.D27)
        self.button_L.direction = Direction.INPUT

        self.button_R = DigitalInOut(board.D23)
        self.button_R.direction = Direction.INPUT

        self.button_U = DigitalInOut(board.D17)
        self.button_U.direction = Direction.INPUT

        self.button_D = DigitalInOut(board.D22)
        self.button_D.direction = Direction.INPUT

        self.button_C = DigitalInOut(board.D4)
        self.button_C.direction = Direction.INPUT

        # Turn on the Backlight
        self.backlight = DigitalInOut(board.D26)
        self.backlight.switch_to_output()
        self.backlight.value = True

        # Create blank image for drawing.
        # Make sure to create image with mode 'RGB' for color.
        self.width = self.disp.width
        self.height = self.disp.height
```

```
def main:
...

        command = {'move': False, 'up_pressed': False , 'down_pressed': False, 'left_pressed': False, 'right_pressed': False}

        if not joystick.button_U.value:  # up pressed
            command['up_pressed'] = True
            command['move'] = True

        if not joystick.button_D.value:  # down pressed
            command['down_pressed'] = True
            command['move'] = True

        if not joystick.button_L.value:  # left pressed
            command['left_pressed'] = True
            command['move'] = True

        if not joystick.button_R.value:  # right pressed
            command['right_pressed'] = True
            command['move'] = True

        if not joystick.button_B.value:
            if my_circle.bullet > 0:
                bullet = Bullet(my_circle.center, command)
                bullet_list.append(bullet)     #누르면 총알 나가게 할까요 여기서?
                my_circle.bullet -= 1
                print("bullet shout",my_circle.bullet)
                time.sleep(0.05)
            else:
                print("no bullet")
```

위 코드로 joystick 의 입력을 받아 command에 저장하고 버튼 B의 입력도 입력받아 구동한다

캐릭터

```
class Character:
    def __init__(self, width, height):
        ...
        self.position = np.array([width/2 - 20, height/2 - 20, width/2 + 20, height/2 + 20])
        ...

    def move(self,command = None):
        if command['move'] == True:
            if command['up_pressed']:     # moverspeed만큼이 아닌 라인만큼 이동
                self.position[1] -= 5
                self.position[3] -= 5

            if command['down_pressed']:
                self.position[1] += 5
                self.position[3] += 5

        self.center = np.array([(self.position[0] + self.position[2]) / 2, (self.position[1] + self.position[3]) / 2])
```

main 함수로부터 joystick의 입력상태가 기록된 command를 넘겨 받아 command에 따라 위로 가야하면 캐릭터의 y좌표인 position의 1번째, 3번째 숫자를 감소시킨다 (y=0 이 가장 위, y= 240 이 가장 밑이다), x좌표로는 움직이지 않고 아래로 가야할때는 위와 같이 y좌표를 증가시키기만 한다.

아이템(코인, 황금키), 장애물

```
class Obstacle:
    def __init__(self, spawn_position):
        ...
        self.position = np.array([spawn_position[0] - 25, spawn_position[1] - 25, spawn_position[0] + 25, spawn_position[1] + 25])
        self.speed = 3
        ...

    def move(self, command = None):
        if command['move'] != False:
            if command['left_pressed']:
                self.speed = 2

            if command['right_pressed']:
                self.speed = 4

        self.position[0] -= self.speed
        self.position[2] -= self.speed
```

장애물의 코드로 위 객체들의 움직임 설명은 같이한다. → 코드가 같기 때문

캐릭터와 같이 main함수에서 받은 command를 활용한다. 하지만 장애물은 왼쪽 방향 (-X 방향)으로만 이동하고, 사용자가 왼쪽으로 조이스틱을 이동하면 천천히, 오른쪽으로 이동하면 빠르게 이동해야 하므로 객체의 속도를 정의한 후 조이스틱의 상태에 따라 속도를 결정하여주고 해당 속도만큼 -x 방향으로 움직이면 된다.

총알

```
class Bullet:
    def __init__(self, position, command):
        ...
        self.speed = 8
        ...

    def move(self):
        self.position[0] += self.speed
        self.position[2] += self.speed
```

총알은 사용자가 버튼을 누르면 캐릭터의 위치에 따라 생성되어 오른쪽으로만 움직이면 된다(+X 방향) 따라서 위와같이 코드를 작성한다

2. **물체들의 충돌 확인**

   캐릭터 - 아이템 (코인 , 황금키) , 장애물 과의 충돌 확인

```
for enemy in enemy_list:
            enemy.move(command)
            my_character.collision_check(enemy)

            if enemy.position[2] <= 0:
                enemy.state = 'die'

            if enemy.state == 'die':
                score += 1
                print(score)
                enemy_list.remove(enemy)
```

main 함수에서 game이 실행되는 while문 내의 코드중 일부이다.
enemy는 화면을 벗어나는 순간 → position[2] 값이 0보다 작아지면 삭제해주고 점수를 1점 올려준다
위 코드에 의해 enemy_list에 저장되어 있는 모든 enemy들은 하나씩 my_character 와 충돌 검사를 하게된다.

```
def collision_check(self, object):
        if object.state == 'alive':
            collision = self.overlap(object.position)

            if collision:
                object.state = 'die'

                if isinstance(object, Item):
                    self.bullet += 1

                if isinstance(object,Obstacle):
                    self.alive = 'die'

                if isinstance(object, Finish):
                    self.alive = 'win'



    def overlap(self, object_position):
        top = max(self.position[1], object_position[1])
        bottom = min(self.position[3], object_position[3])
        left = max(self.position[0], object_position[0])
        right = min(self.position[2], object_position[2])

        if bottom > top and left < right:
            return True

        else:
            return False
```

위는 Character 클래스 내부의 collision_check 함수와 overlap 함수이다.
overlap 함수 → 입력 받은 물체가 지금 나의 캐릭터와 겹치는 상태 (충돌) 인지 확인하여 겹치는 상태라면 True 를 겹치지 않는 상태라면 False를 반환하는 함수이다.
collision_check 함수 → 입력받은 물체가 overlap 함수에 의해 충돌 상태라고 반환받으면 그 물체의 상태를 'die'로 바꿔주고 그 물체의 클래스 (어떤 물체인지) 확인하여 아이템 (코인) 이면 캐릭터의 총알을 하나 추가, 장애물이라면 캐릭터의 alive 를 'die'로 , 골드키라면 캐릭터의 alive 를 'win'으로 바꿔준다

💡 캐릭터의 alive는 'alive' 가 초기 상태이다.
'die'라면 캐릭터가 죽은 상태이다 → 게임 종료(패배)
'win'이라면 게임을 이긴 상태 → 게임 종료(승리)

캐릭터와 장애물과의 충돌 확인 설명으로 아이템과 캐릭터의 충돌 확인 설명은 대체한다

총알과 장애물의 충돌 확인

총알과 장애물의 충돌 확인은 총알 객체에서 해준다.

```
for bullet in bullet_list:
            bullet.collision_check(enemy_list)
            bullet.move()

            if bullet.position[0] >= 240:
                bullet_list.remove(bullet)

            if bullet.position[0] < 240 and bullet.state == 'hit':
                score += 5
                print(score)
                bullet_list.remove(bullet)
```

아래는 Bullet 클래스의 collision_check 함수와 overlap 함수이다

```
def collision_check(self, enemys):
        for enemy in enemys:
            collision = self.overlap(enemy.position)
```

```
            if collision:
                enemy.state = 'die'
                self.state = 'hit'

    def overlap(self, other_position):
        top = max(self.position[1], other_position[1])
        bottom = min(self.position[3], other_position[3])
        left = max(self.position[0], other_position[0])
        right = min(self.position[2], other_position[2])

        if bottom > top and left < right:
            return True

        else:
            return False
```

위의 캐릭터와 장애물의 충돌 검사와 같은 알고리즘으로 충돌 여부를 검사한다.
충돌이 된 상태라면 충돌된 enemy의 state 는 'die'로 총알의 state는 'hit'로 바꿔준다
위 둘 역시 main함수에서 state가 각각 'die'와 'hit'이면 list에서 제거된다.

3. **화면에 그림 그리기**

화면에 그림을 그리는건 PIL 라이브러리의 Image 클래스를 활용하였다

```
my_image = Image.new("RGBA", (joystick.width, joystick.height))

image_charcter = Image.open("/home/kau-esw/TA-ESW/gameProject/dk_deft.png")
background = Image.open("/home/kau-esw/TA-ESW/gameProject/background.png")

my_image.paste(background,(0,0))

my_image.paste(image_charcter,(int(my_character.position[0]), int(my_character.position[1])))

joystick.disp.image(my_image)
```

위는 main함수 내의 가장 기본적인 배경 위의 캐릭터를 그리는 코드를 일부 가져온 것이다.
사진이 나오게 되는 방법은 미리 rasp에 옮겨놓은 사진 파일을 Image.open을 해 변수에 저장해 둔 후, 내가 띄워줄 그림인 my_image에 paste해준다.

> 💡 Image.paste(사진,위치) 이므로 사진에는 내가 붙여줄 사진, 위치에는 그 사진이 나와야 할 위치 (사각형의 왼쪽 위 점의 좌표) 를 입력해준다.

나머지 객체들은 위와 같은 방법으로 그려준다

4. **게임 진행**

아이템과 장애물의 생성은 threading 모듈을 활용하여 구현하였습니다.
threading.Timer()를 활용하여 일정 시간마다 반복하거나, 일정 시간 후에 행동을 하도록 명령하였습니다.

```
def make_object():
        sample_list = [25, 73, 121, 169, 215]
        spawn_list = random.sample(sample_list,2)

        temp_enemy = Obstacle((230,spawn_list[0]))
        enemy_list.append(temp_enemy)

        temp_item = Item((230,spawn_list[1]))
        item_list.append(temp_item)

        threading.Timer(2.5,make_object).start()
```

main함수 내부의 아이템 (코인) 과 장애물을 동시에 만드는 make_object() 함수입니다.
위 함수를 실행하게 되면 장애물과 아이템을 생성하고 각각을 장애물 리스트에 담은 후, 2.5초 후 같은 행동을 하는 본인 함수를 다시 실행시킵니다.

```
threading.Timer(61.5,end_game).start()

def end_game():
        if score < 85:
            my_image.paste(image_die,(0,0))
            joystick.disp.image(my_image)
            quit()  #점수 기준치 미달시 게임 패배
        else:
            temp_finish = Finish((230,115))
            finish_list.append(temp_finish)
```

main함수에서 맨 위의 코드를 실행하면 61.5초 후에 end_game() 이라는 함수를 실행하여 score가 85점을 넘지 못하면 게임 실패 화면을 띄워주고 게임을 종료하고, 85점을 넘었다면 Finish 객체를 생성하여 finish_list에 담아줍니다.

개발 이력

1. 수업 시간에 한 예제 코드를 통한 캐릭터와 장애물의 가장 기초적인 움직임 구현
2. 캐릭터, 장애물, 아이템, 배경 그림 그리기 후 이미지 rasp로 옮기기
3. 캐릭터와 객체들 그리고 총알과 장애물의 충돌 구현 후 리스트에서 삭제까지
4. 객체들의 이동 속도를 게임에 몰입할 수 있는 적당한 수준으로 조절
5. 버튼 누르면 캐릭터의 위치에서 총알이 날라가는 것 구현
6. 배경 - 캐릭터, 리스트에 남아있는 객체들 순서로 그리기 구현
7. 게임 끝내기 → 골드 키를 이용한 게임 끝내기 혹은 게임 패배 구현

주요 issue 및 해결책

1. **rasp 연결 문제**
   a. wifi 문제인 경우가 대다수 → wifi를 끄고 연결 후 연결에 성공하면 wifi랑 연결해도 된다.
   b. 프로세스에 없는 파이프에 연결하려고 하였습니다, ssh 무한 로딩
   → 해결 방법을 명확하게는 모르겠으나 시도하여 성공했던 방법들은
   1. .ssh파일의 known_hosts 파일에 적힌 글자를 모두 지운 후 저장, 그리고 vscode에서 ssh를 종료 후 연결 서버 제거 (kill 기능이 있음) 한 후 다시 연결 시도
   2. ssh ip주소를 터미널에서 복사하여 cmd에서 wget을 통해 rasp 서버? 를 다운 받는다

2. **python 파일 import가 안됨**
   이유를 모르겠지만 파이썬을 정상적으로 설치했음에도 파일이 import가 안되는 경우가 있었음
   → 파이썬을 삭제 후 다시 설치하니 해결 됨

3. **image를 배경을 지운 후 그렸는데도 rasp에 와서 paste하면 배경이 생김**
   사실 해결을 못했다. image 를 RGBA로 바꾸면 원래 A → 투명도로 투명하게 만들면 안보여야 하는데 위 코드가 정상적으로 수행되지 않음. rgb 값들을 바꾸어 검은색을 흰색으로 하는 등 여러가지는 가능했지만 투명도는 설정해도 바뀌는 것이 없었다 ㅠ_ㅠ

4. **충돌 확인 메커니즘 오류**

두 개의 사각형이 겹치는지 확인하는 하는 방법을 사용, 허나 분명 겹치는데 안 겹친다고 리턴값이 나옴 → python의 그림판은 y축이 우리의 생각과는 반대 방향으로 이루어졌다.

5. **일정한 시간 마다의 객체 생성, 그리기**
   a. 객체의 생성을 while문 안에서 해주게 되면 내가 위에서 써준 코드가 while문에서 계속해서 새로 호출되는 상황이 생기고 while문의 주기가 너무나도 빠르기 때문에 while문 밖에서 써줘야 한다. 그리고 while문 안에서 생성만 하게되면 다시 while문이 실행될 때 객체가 사라짐
   b. 객체를 생성하게되면 while문 밖에서 생성하는데 수 많은 객체들을 확인하려면 리스트에 담아준 후 그 리스트를 반복하며 확인해줘야 한다.

6. **이미지 전달, 이미지 오픈**

   컴퓨터 → rasp 이미지 전달은 samba 아니면 usb를 통해 vnc로 파일을 옮기는 두가지 방법

   이미지 오픈은 파이썬에서는 원래 같은 폴더에 있으면 Image.open('사진 이름.png')를 통해 열려야 하는데 열리지 않음 → 그냥 경로 전부를 입력해줌

배운점

파이썬에서 게임 만들기 → 정확히 말하면 이미지를 다루는 법을 많이 익히고 배웠습니다.

이미지 - 캐릭터를 원하는 위치에 그리는 방법, 내가 원하는 이미지를 입히는 방법을 배웠습니다.

사실 rasp를 사용하면서 정말 많은 기기 오류가 생겼었는데 이 오류들을 직접 검색하고 해결하면서

코딩을 하려면 역시 온라인에 있는 많은 자료들을 잘 해결해야 한다는 사실을 다시 배웠습니다.