풀스택서비스네트워킹 프로젝트 최종 보고서

2019102190 컴퓨터공학과
신정아

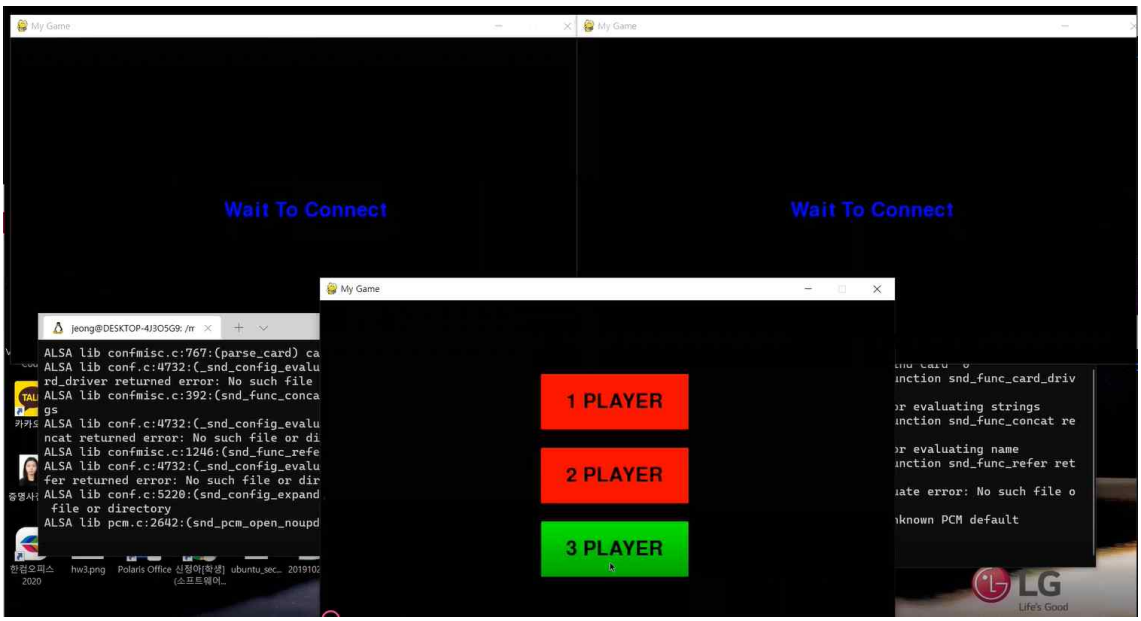
[목차]

1. 개발 결과물 설명

**** 다음의 설명은 3인이 참여하였을 때의 화면들을 기준으로 하였음**

풀스택서비스네트워킹 프로젝트 계획서에서 제출하였다시피, 최대 3인의 사용자가 함께 참여할 수 있는 장애물 피하기 게임을 제작하였다.

1-1. 플레이어 모드 선택



게임을 실행하면 위의 화면과 같이 플레이어 모드를 선택할 수 있게 하였다. 만약 1 PLAYER를 선택하면 바로 실행된다. 그리고 2 PLAYER, 3PLAYER를 선택하였을 경우, 해당 모드에 필요한 클라이언트 수만큼이 접속할 때까지 'Wait To Connect'화면을 띄우고, 만약 해당 모드에 필요한 클라이언트 수만큼이 접속한다면 게임을 실행한다.

1-2. 게임 실행 중 화면

게임이 실행되면 해당 클라이언트의 화면에서 해당 캐릭터는 사자로, 다른 클라이언트들은 펭귄 캐릭터로 나온다. 그리고 각 캐릭터는 각각의 클라이언트들이 움직이는 대로 동일하게 움직인다. 좌측 상단에는 남은 시간을, 우측 상단에는 현재 점수 상태에 따른 중계 멘트를 나타낸다. 그리고 떨어지는 돌을 맞을 때마다 “충돌했습니다” 라는 문구와 함께 현재 점수를 보여준다.

1-3. 일정시간(50초) 후 종료 화면

주어진 시간이 모두 지나면 모든 사용자들의 게임을 중지하고, 각 클라이언트들에게 각자의

점수를 보여준다. 그리고 최종 1,2,3등을 모든 사용자들에게 보여준다.

2. 소스코드 소개
(서버와 클라이언트 간의 데이터 전달을 중심으로 소개함)

2-1. 전체적인 소개

제출한 프로젝트 계획서대로 zmq를 활용하여 구현하였다.
여러 사용자를 한 번에 보여주고(쓰레드로 처리), 데이터 전송과 응답을 비동기적으로 해야 하기 때문에 수업 시간에 배웠던 Dealer-Router pattern with Multi-thread Client에 PUB-SUB 패턴을 추가하여 구현했다.

2-1-1. client.py

```
self.context = zmq.Context()
#DEALER 소켓을 생성
self.socket = self.context.socket(zmq.DEALER)
self.identity = u'%s' % self.id #id값은 사용자가 입력한 값
self.socket.identity = self.identity.encode('ascii')
#DEALER를 CONNECT함
self.socket.connect('tcp://localhost:5570')

print('Client %s started' % (self.identity))
self.poll = zmq.Poller()
self.poll.register(self.socket, zmq.POLLIN)

self.subscriber=self.context.socket(zmq.SUB)
self.subscriber.setsockopt(zmq.SUBSCRIBE,b'')
self.subscriber.connect("tcp://localhost:5557") # 클라이언트이므로 bind가 아니라 connect

clientThread = threading.Thread(target=self.recvHandler)
clientThread.daemon = True
clientThread.start()
```

각 클라이언트들의 정보를 서버(라우터)에 보낼 수 있게 DEALER 소켓을 만들어주고, 서버에서 모든 클라이언트들에게 뿌리는 정보를 받아올 수 있도록 SUB 소켓을 만들어주었다.

그리고 클라이언트들의 동작들을 비동기적으로 처리할 수 있도록 쓰레드로 만들었다. 메인 쓰레드가 종료되면 함께 종료될 수 있게 daemon을 True로 설정해주었다.

또한 게임 동작을 하면서 서버로부터 응답을 받아올 수 있도록 recvHandler를 쓰레드로 동작할 수 있게 했다.

```
while True:
    if(self.subscriber.poll(100)&zmq.POLLIN):
        message=json.loads(self.subscriber.recv())
```

위 사진은 recvHandler의 일부분으로, 서버에서 데이터를 받아 json으로 처리하는 부분이다. (.poll(time-out 제한 시간), zmq.POLLIN을 통해 해당 소켓에 데이터가 있는 경우에만 동작하도록 한다.)

2-1-2. server.py

```
class ServerTask(threading.Thread):
    """ServerTask"""
    def __init__(self, num_server):
        threading.Thread.__init__ (self)
        self.num_server = num_server

    def run(self):
        context = zmq.Context()
        frontend = context.socket(zmq.ROUTER)
        frontend.bind('tcp://*:5570')

        backend = context.socket(zmq.DEALER)
        backend.bind('inproc://backend')

        publisher=context.socket(zmq.PUB)
        publisher.bind("tcp://*:5557")

        workers = [] #worker들
        for i in range(self.num_server): #num_server는 사용자가 입력한 숫자
            worker = ServerWorker(context, i,publisher) #각 woker에 serverWorker가 들어가는 거임
            worker.start() #serverworker클래스의 run함수 동작
            workers.append(worker)
```

클라이언트들의 DEALER에서 보내는 데이터들을 받기 위해서 ROUTER소켓을 만들어준다. 그리고 worker들과 연결되기 위해서 DEALER 소켓을 만든다. 마지막으로 각 클라이언트들에게 동일한 정보들을 뿌려주기 위해서 PUB 소켓을 만들어주었다. worker들의 수는 사용자가 입력한 만큼 만들어질 수 있게 했다.

```
class ServerWorker(threading.Thread):
    """ServerWorker"""
    def __init__(self, context, id,publisher):
        threading.Thread.__init__ (self)
        self.context = context
        self.id = id
        self.publisher=publisher

    def run(self):
        global clientNum
        worker = self.context.socket(zmq.DEALER)
        worker.connect('inproc://backend')
        print('Worker#{0} started'.format(self.id))
        global stage_data
        global first
        global second
        global third
        global relay_info
        global is_prevment
        global prevment
        global count
        global rank_2
        global rank_3

        while True:
            ident, msg = worker.recv_multipart() #클라이언트로부터 받은 메시지들
```

위의 사진은 worker를 구현하기 위한 코드의 일부분이다. 여러 개의 worker가 동시에 비동기적으로 동작할 수 있도록 하기 위해 Thread로 구현했다. 그리고 worker에서 작업할 데이터를 받아오기 위해 DEALER 소켓을 만들고, 위에서 설명했던 또다른 DEALER 소켓이었던 backend에 연결하였다. 그리고 recv_multipart()를 통해 각 클라이언트들로부터 들어온 메시지를 받아온다.

2-2. 데이터 주고 받으며 처리하기

2-2-1. 클라이언트에서 데이터 보내기

```
self.socket.send_string('s'+stage) #서버에 선택한 stage정보 보냄
```

서버에 해당 클라이언트가 선택한 모드(한 명인지, 2명인지, 3명인지) 데이터를 보낸다

```
#게임 캐릭터의 움직인 위치
character_x_pos+=to_x*dt
self.socket.send_string(str(character_x_pos))
```

서버에 각 클라이언트들의 캐릭터들의 움직인 위치 정보를 보낸다

```
#충돌 체크 - colliderect : 사각형을 기준으로 충돌이 있는지 | 확인
if character_rect.colliderect(rock1_rect):
    rock1_y_pos=0
    rock1_x_pos=random.randint(0,screen_width-rock1_width)
    is_collision1=True
    score-=3
    self.socket.send_string('j'+str(score)) #충돌될 때마다 업데이트되는 점수 정보 보냄
elif(character_rect.colliderect(rock2_rect)):
    rock2_y_pos=0
    rock2_x_pos=random.randint(0,screen_width-rock1_width)
    is_collision2=True
    score-=2
    self.socket.send_string('j'+str(score))
elif(character_rect.colliderect(rock3_rect)):
    rock3_y_pos=0
    rock3_x_pos=random.randint(0,screen_width-rock1_width)
    is_collision3=True
    score-=5
    self.socket.send_string('j'+str(score))
```

캐릭터들이 각 돌들에 충돌될 때마다 업데이트되는 점수를 서버에 보낸다

2-2-2. 서버에서 데이터 처리하기

**** 모든 데이터들을 처리후 json형식으로 클라이언트들에게 뿌림**

```
if(msg.decode()[0]=='s'): #s로 시작하면 stage 정보임
    if(msg.decode()=='s1'):
        stage_data['s1']+=1
    elif(msg.decode()=='s2'):
        stage_data['s2']+=1
    elif(msg.decode()=='s3'):
        stage_data['s3']+=1
    self.publisher.send(json.dumps(stage_data).encode())
```

전달받은 클라이언트들의 모드 정보에 따라 현재 해당 모드에 몇 명의 클라이언트가 있는지 업데이트한다.

```
elif(msg.decode()[0]=='j'): #j로 시작되면 점수 정보임 #one player모드에서는 점수 비교 필요 없음-점수 관련 정보는 publisher로 뿌림
    if(stage_data['s2']==2): #2 player mode 시작이면
        if(str(ident.decode())=='1'):
            score_data_2['1']=int(msg.decode()[1:])
        elif(str(ident.decode())=='2'):
            score_data_2['2']=int(msg.decode()[1:])
        first=(max(score_data_2, key=score_data_2.get))
        second=(min(score_data_2, key=score_data_2.get))
        rank_2['first']=first
        rank_2['second']=second
        if((score_data_2[first]-score_data_2[second])>=30):
            text1=str("{}플레이어님, 분발하세요!".format(second))
            if(prevment !=text1):
                prevment=text1
                relay_info['relay_info']=text1
                self.publisher.send(json.dumps(relay_info).encode())
        elif((score_data_2[first]-score_data_2[second])>=20):
            text2=str("{}플레이어님이 앞서고 있습니다!".format(first))
            if(prevment != text2):
                prevment=text2
                relay_info['relay_info']=text2
                self.publisher.send(json.dumps(relay_info).encode())
        elif((score_data_3[first]-score_data_3[second])<=10):
            text3=str('아직 1등이 확실하지 않습니다!')
            count+=1#충돌될 때마다 클라이언트에서 점수를 보내는데, 1등이 확실하지 않다는 멘트를 충돌이 적어도 3번보다 많을 때 중계 시작
            if(prevment !=text3 and count>3):
                prevment=text3
                relay_info['relay_info']=text3
                self.publisher.send(json.dumps(relay_info).encode())
        self.publisher.send(json.dumps(rank_2).encode())
```

점수를 전달 받고 해당 점수에 따라 등수 정보를 업데이트한다. 그리고 점수를 전달 받을 때마다 비교해서 점수 차에 따라 클라이언트에 보내는 중계 멘트를 다르게 해주었다. stage가 3인 경우도 똑같으므로 해당 보고서에서는 생략한다.

```
else: #위치정보임
    pos_data={"ident":ident.decode(),"msg":msg.decode()}
    payload=json.dumps(pos_data).encode()
    self.publisher.send(payload)
```

캐릭터들의 위치 정보를 받으면(메시지가 특정한 글자로 시작하지 않으면) 각 클라이언트별 위치정보를 json형식으로 만들어준 다음 클라이언트 측에 보낸다.

2-2-3. 서버에서 처리된 데이터 클라이언트가 받기

```
while True:
    if(self.subscriber.poll(100)&zmq.POLLIN):
        message=json.loads(self.subscriber.recv())
        if('clientNum' in message):
            global clientNum
            clientNum=message['clientNum']
        elif ('s1' in message):
            global stage_data
            stage_data[0]=message['s1']
            stage_data[1]=message['s2']
            stage_data[2]=message['s3']
        elif('relay_info' in message):
            global relay_text
            relay_text=message['relay_info']
        elif('third' in message):
            rank['first']=message['first']
            rank['second']=message['second']
            rank['third']=message['third']
        elif('second' in message):
            rank['first']=message['first']
            rank['second']=message['second']
```

게임 모드 정보, 중계멘트, 등수 정보 등을 키를 통해 구분해서 받아 처리한다.

```
else:
    if(message['ident'] != self.identity): #해당 클라이언트가 아닌 다른 클라이언트의 위치 정보임
        if(is_enemy == False):
            ident_standard=message['ident']
        if(message['ident']==ident_standard):
            is_enemy=True
            enemy=pygame.image.load('./picture/peng.png')
            enemy_size=enemy.get_rect().size
            enemy_width=enemy_size[0]  #가로
            enemy_height=enemy_size[1] #세로크기
            enemy_x_pos=(screen_width/2)-(enemy_width/2)+20 #화면 가로의 절반에 해당하는 곳에 위치 #x좌표
            enemy_y_pos=screen_height-enemy_height-6 #화면은 아래쪽으로 갈 수록 +임 #y좌표
            enemy_x_pos=float(message['msg'])
        elif(message['ident']!=ident_standard):
            is_enemy2=True
            enemy2=pygame.image.load('./picture/peng.png')
            enemy2_size=enemy2.get_rect().size
            enemy2_width=enemy2_size[0] #가로크기
            enemy2_height=enemy2_size[1] #세로크기
            enemy2_x_pos=(screen_width/2)-(enemy2_width/2)+20
            enemy2_y_pos=screen_height-enemy_height-6
            enemy2_x_pos=float(message['msg'])
```

위치 정보를 받았을 경우, 각 클라이언트별로 위치 정보를 구분해서 업데이트하고 캐릭터의 위치 정보로 넣어준다.

3. 데모 영상 소개 및 시작에 대한 설명

*첨부된 2019102190_1player 영상은 1player 모드로 게임에 참여하였을 때의 영상이다.

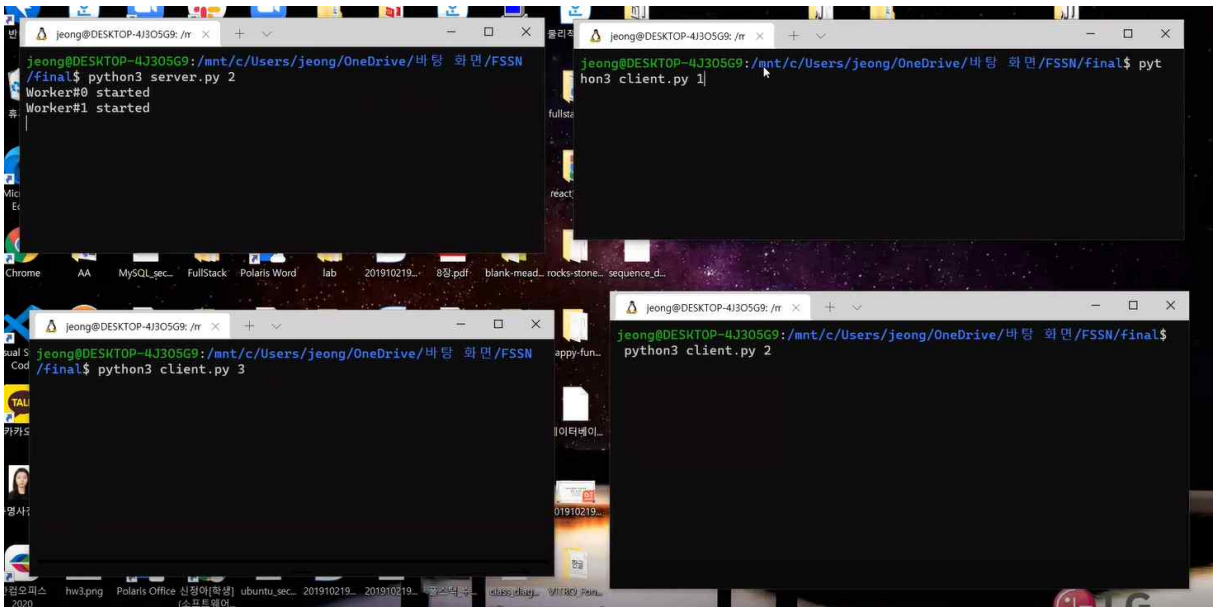
*첨부된 2019102190_2player 영상은 2player 모드로 게임에 참여하였을 때의 영상이다.

*첨부된 2019102190_3player 영상은 3player 모드로 게임에 참여하였을 때의 영상이다.

우선 client,server 코드 모두 명령행 인자를 받도록 구현하였다. client 경우에는 client id 값을 입력하고, server의 경우에는 worker를 몇 개 만들지 입력하는 것이다.

먼저 [python3 server.py 2]를 입력해서 서버를 구동한다. (worker를 2개 띄운다)

그리고 클라이언트 id값을 1,2,3으로 준다.



예를 들어 1 player 모드로 참여하고 싶다면 [python3 client.py 1]을 입력해서 시작하고 1 player를 선택해서 게임을 시작한다.

2 player모드로 참여하고 싶다면, [python3 client.py 1], [python3 client.py 2]를 입력하고 각각의 클라이언트에서 2player를 선택한다.

3 player 모드로 참여하고 싶다면 [python3 client.py 1], [python3 client.py 2] ,[python3 client.py 3]를 입력하고 각각의 클라이언트에서 3player 모드를 선택한다.

모드 선택 및 게임 실행, 종료 화면은 위의 [1. 개발 결과물 설명]과 동일하므로 생략한다.

(+) 3 player 모드를 실행하고 녹화 프로그램까지 실행하니 게임 구동 속도가 급격하게 느려지는 현상이 발생하였습니다. 3player 모드의 녹화 영상으로는 3player에서도 잘 실행된다는 정도로만 알아주시고, 실제적인 속도와 중계 기능 등은 2 player모드의 녹화 영상을 참고해주십시오.

4. 느낀점 및 후기

윈도우 환경에서 ZeroMQ를 사용하기 위해서 WSL2를 설치하고 사용해보았는데, 예상했던 것보다 자료가 많아 개발 환경 설정은 수월하였다. 다만, WSL2에는 기본적으로 디스플레이 관련 기능이 포함되어 있지 않아서 GUI 기반 어플리케이션을 사용할 수 없었기 때문에 별도로 X server를 설치하여야 했는데, 이 과정에서 방화벽 설정 및 DISPLAY 환경 변수 설정에서 조금 어려움이 있었다. 하지만 성공 후, WSL2을 통해 윈도우에서도 리눅스 개발환경(우분투)으로 자유롭게 개발할 수 있다는 것을 체감하면서 WSL2의 위력을 느낄 수 있었다.

그리고 ZeroMQ를 통해 개발하면서, application level에서 라이브러리 형태로 통신 프로토콜을 불러올 수 있다는 게 정말 편리하다는 것을 실감했다. 특히 자신이 원하는 기능에 적절한 패턴이 있다면 자유롭게 패턴별로 소켓을 불러와 기능들을 합칠 수도 있었고, 공식 문서에서 설명하는 API를 참고해서 구현할 수 있었다. 그리고 ZeroMQ는 기존의 socket 프로그래밍을 기반으로 하기 때문에 진입 장벽이 생각보다 매우 낮은 것도 큰 장점이라고 생각했다.

마지막으로 게임 구현에 대해서 조금 아쉬운 점이 있다. 게임을 계획했을 때는 플레이어가 선택한 모드에 따라 화면에 다른 플레이어의 캐릭터가 동시에 보이면 좋을 것 같았는데, 막상 구현해보니 굳이 다른 플레이어가 화면에 동시에 보여질 필요가 있었나하는 의문이 들었다. 나중에 시간 및 기회가 된다면 화면에 동시에 나타나는 다른 플레이어와의 인터랙션 부분을 강화해서 추가하고 싶다.