Pygame Shooting Project

몬스터를 찾아서

3팀

구 혜령

김 기운

김 현구

이 도규

목차

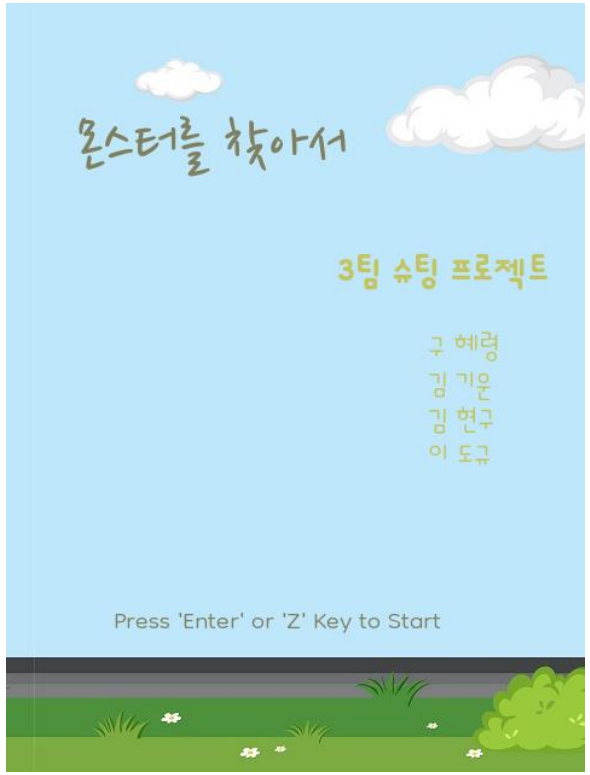
1. 프로젝트 개요

개요

팀원 소개

타임 라인

1. 프로젝트 개요



제목 : 몬스터를 찾아서

플랫폼 : PC

장 르 : 종 스크롤 비행 슈팅 게임

개발 엔진 : Python

기획 목표

- Python으로 기본적인 슈팅 게임을 구현할 수 있다.
- Python 함수 및 클래스 적용 방법들을 연구 체험한다
- 팀 프로젝트에서 코드를 교류하는 방법을 탐방한다.

요구 사항 분석 → 개발 환경 구성 → 코드 구조화 → 페이즈 테스트 → 개발 확인

슈팅게임 요소 확인

매개변수 / 함수 / 클래스 구성에 대한 회의 및 연구

1. 프로젝트 개요 - 팀원소개

구 혜 령

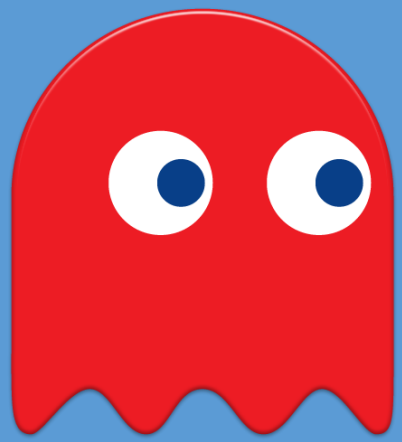
시나리오 담당
게임 컨셉 담당
보스 담당

김 현 구

Phase 구상 담당
리소스 적용
테스트 담당

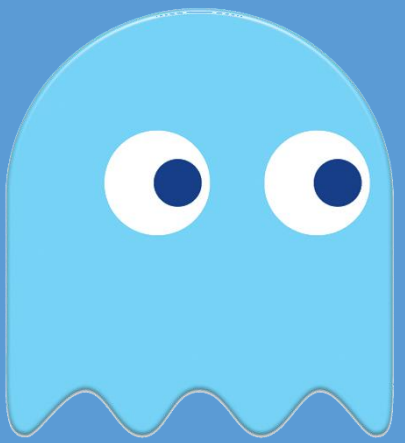
김 기 운

UI 인터페이스 담당
리소스 탐색 담당

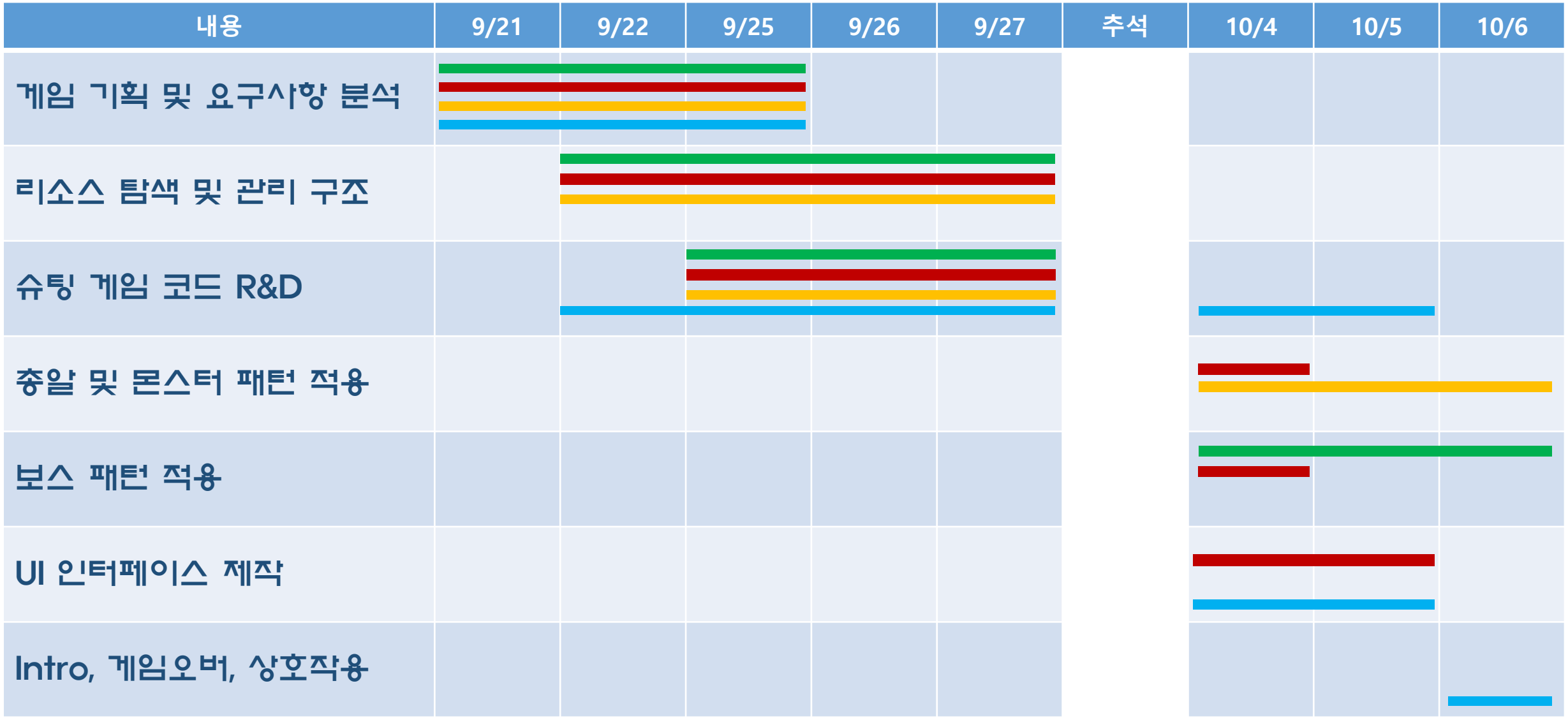


이 도 규

팀장역
전체 코드 구조 담당
클래스 구조화 담당



1. 프로젝트 개요 – 타임라인



2. 게임 구성

개요

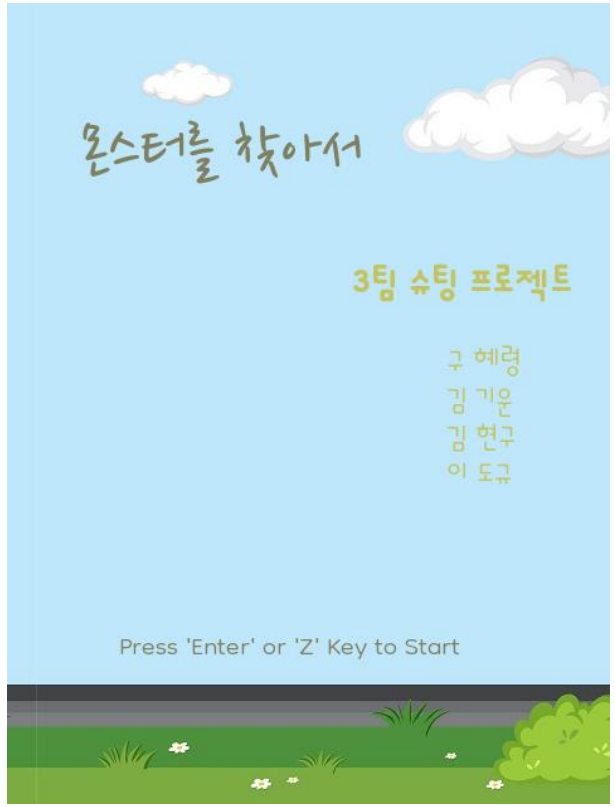
캐릭터

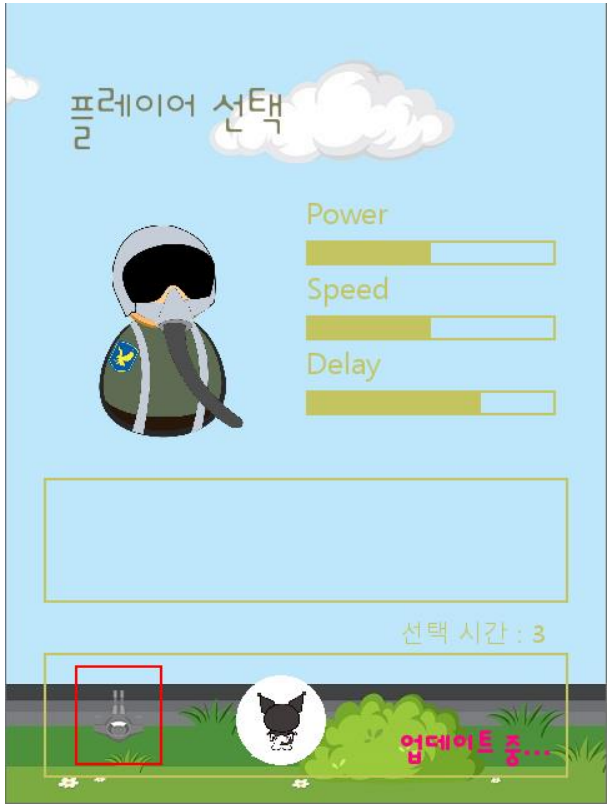
스테이지

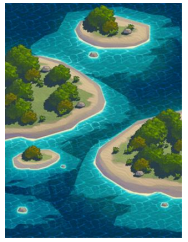
기타 요소

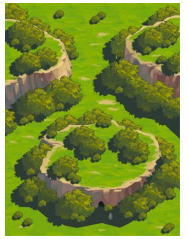
2. 게임 구성 - 개요

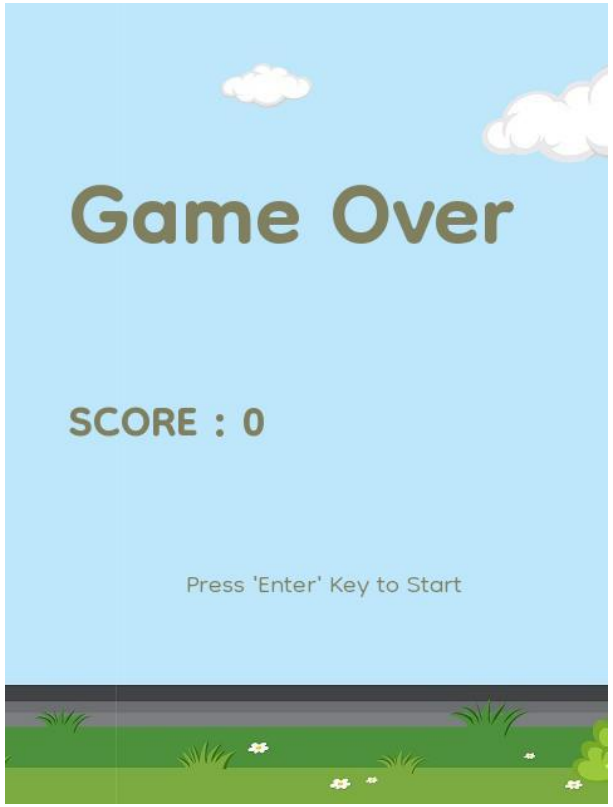
Z : 발사 / X : 필살기 / 화살표 : 움직임











용병이 사막, 바다, 산악에 있는 다양한 몬스터들을 찾는다

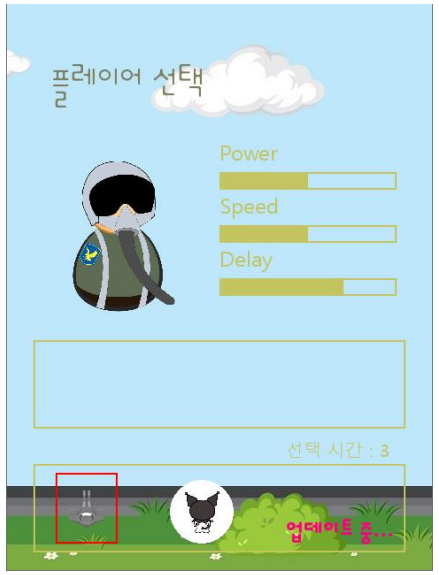
- 비행을 통해 2차원 공간 내에서 자유로운 종횡주행 및 몬스터 파괴
- 3단계 스테이지 구성 및 보스 구현

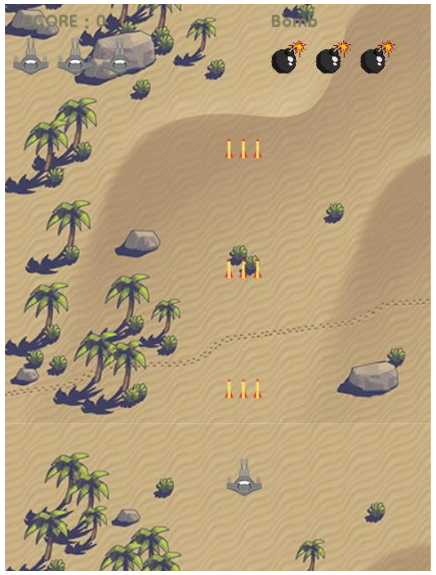
화살표 : 움직임

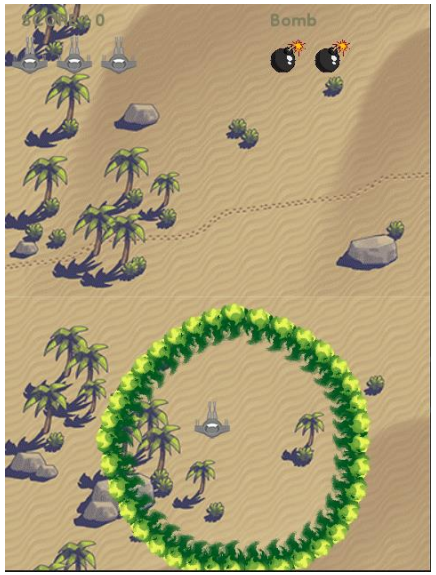
Z : 발사

X : 필살기

2. 게임 구성 – 캐릭터



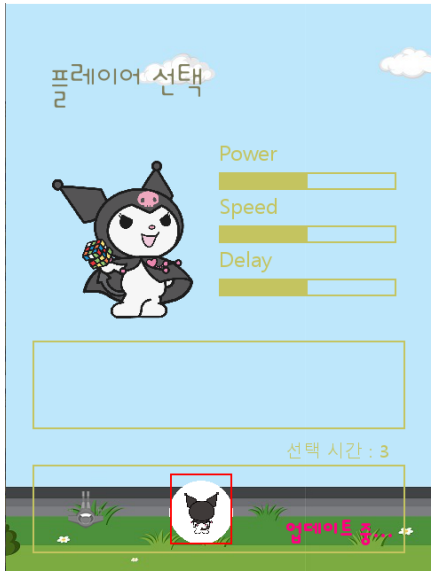




Player 1

일반공격 : 직선으로 나가는 총알

특수공격 : 36방향으로 나가는 총알



Player 2

일반공격 : 3방향으로 나가는 총알

특수공격 : 화면 클리어하기

[화면 클리어 이펙트 미구현]

2. 게임 구성 – 스테이지

Stage 1 - 사막의 폭풍

Phase 1 : 왼쪽에서 내려오기

Phase 2 : 오른쪽에서 내려오기

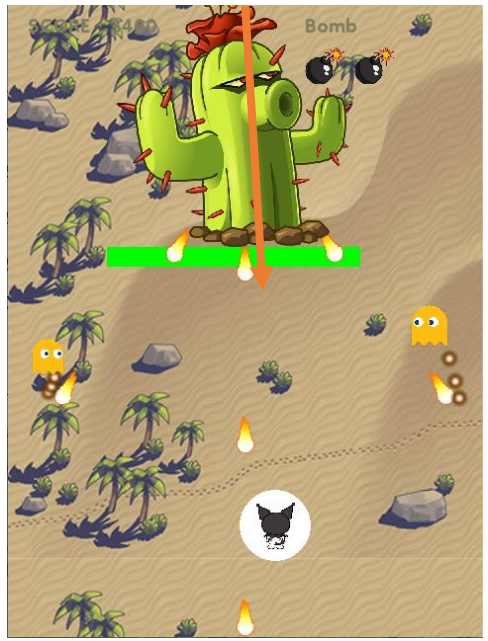
Phase 3 : 양방향에서 랜덤으로 내려오기

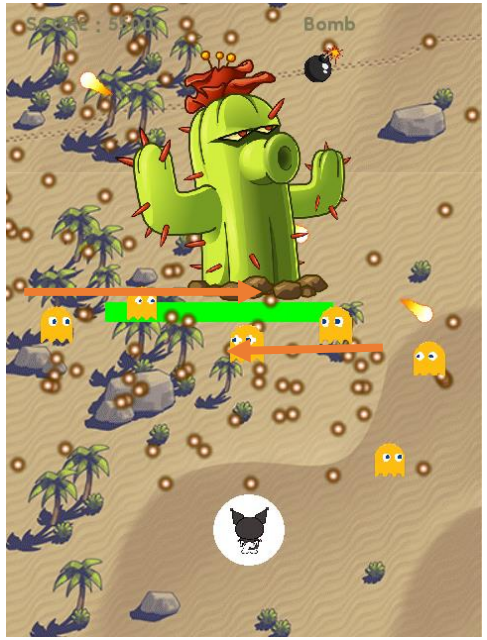
Phase 4 : 보스 등장하기

보스 Phase 1 : 아래로 총알 발사

보스 Phase 2: 사방으로 총알 발사

- 유도탄을 날리는 몬스터 스폰





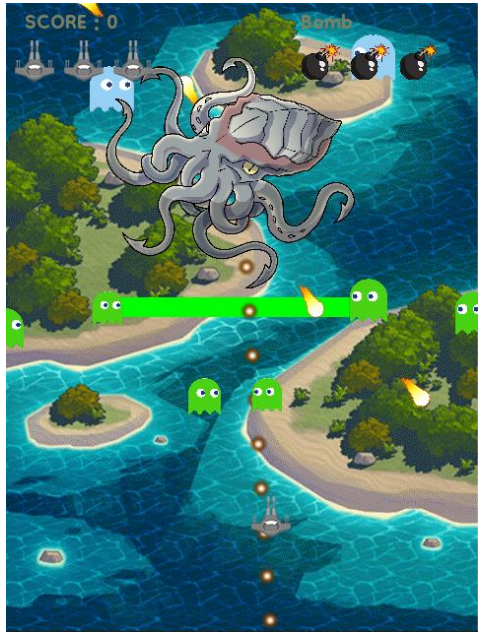
2. 게임 구성 – 스테이지

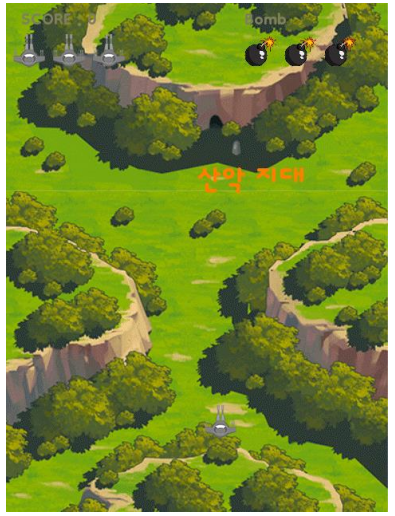
Stage 2 – 바다 괴물

보스 Phase 1 : 아래로 총알 발사

보스 Phase 2 : 4방향 몬스터 스폰

- 2방향으로 총알 발사
- 중심에서 유도탄 발사



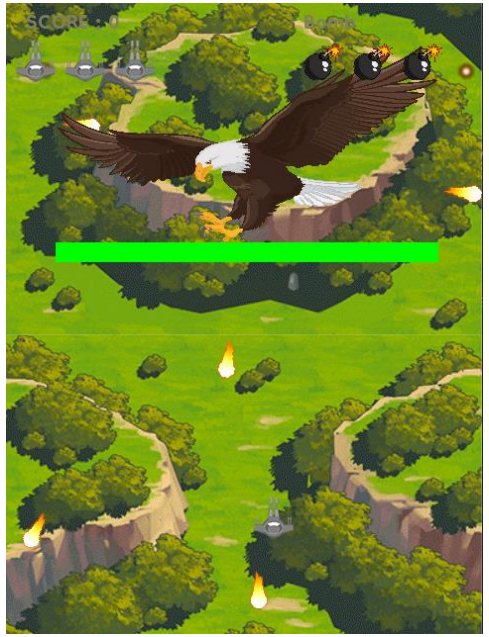


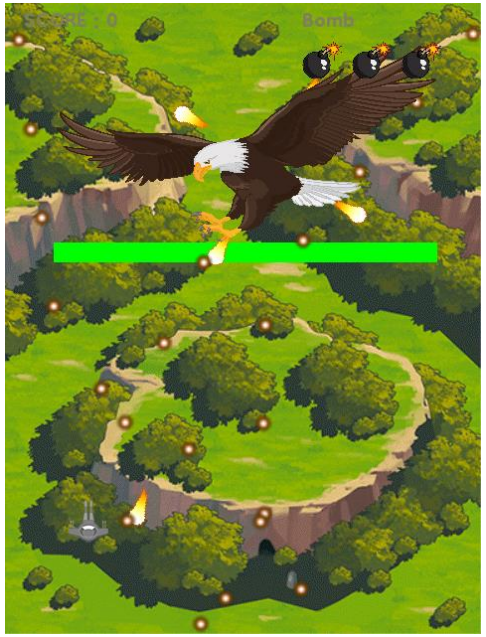
Stage 2 – 산악 지대

보스 Phase 1 : 아래로 총알 발사

보스 Phase 2 :

- 4방향으로 총알 발사
- 3곳에서 유도탄 발사





2. 게임 구성 - 요소

디버그 키
F1 : Intro / F2 : Stage 1 / F3 : Stage 2 / F4 : Stage 3
F5 : Outro / F6: 일반모드 / F7: 무적모드

플레이어 기체




미사일





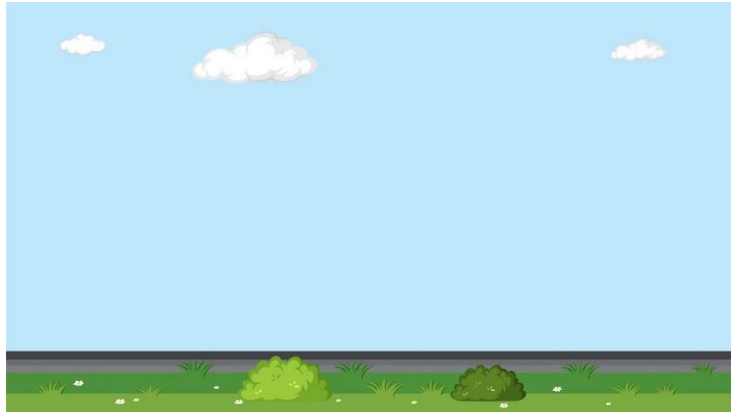
적 기체




보스



배경




3. 기능 상세

클래스 구조도

이미지 및 객체 관리

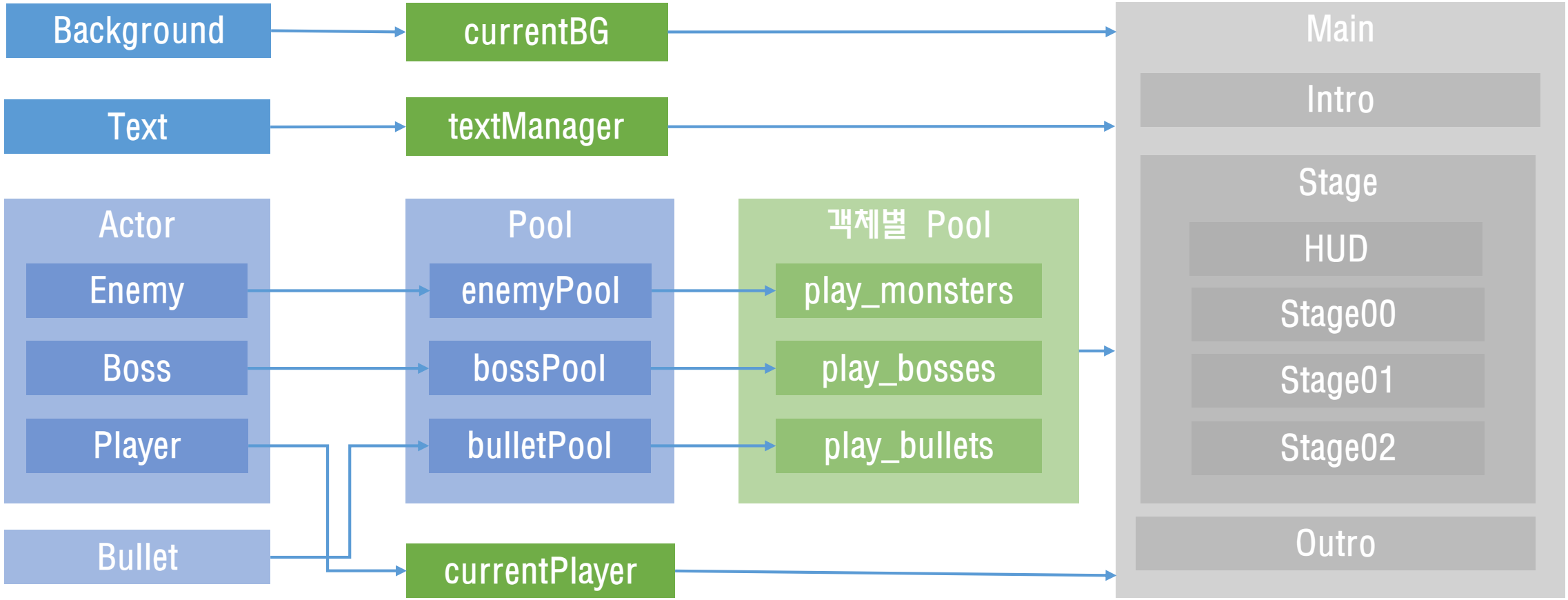
Background

Text

Actor

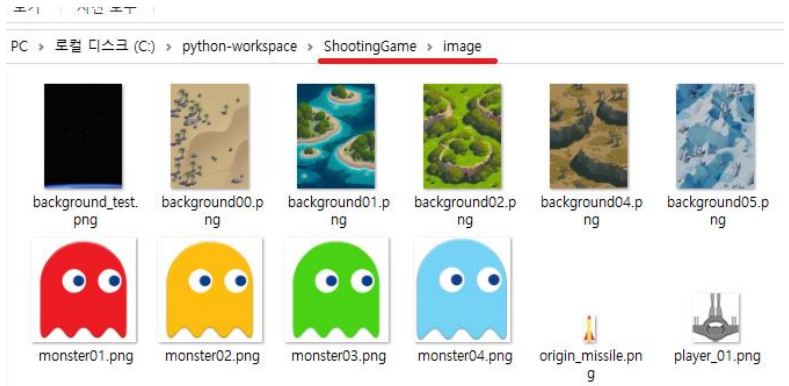
Pool

3. 기능 상세 – 클래스 구조도



3. 기능 상세 _ 이미지 및 객체 관리 구성

1. 이미지 파일 읽기 – 경로 이미지 가져오기




```python
def readGameResource():
    global image_backgrounds, image_player_profiles, image_players , image_monsters , image_boss

    current_path = os.path.dirname(__file__)

    image_path = os.path.join(current_path, "image")

    # 팩맨 빨강
    image_monsters["1"] = (pygame.image.load(os.path.join(image_path,"monster01.png")))
    image_monsters["1"] = pygame.transform.scale(image_monsters["1"],(48,48))
    image_monsters["2"] = pygame.transform.flip(image_monsters["1"], True, False)

    image_monsters["3"] = (pygame.image.load(os.path.join(image_path,"monster01.png")))
    image_monsters["3"] = pygame.transform.scale(image_monsters["3"],(48,48))
    image_monsters["4"] = pygame.transform.flip(image_monsters["3"], True, False)
```

2. 클래스 객체로 데이터 딕셔너리로 저장하기

```python
data_players = {}
data_monsters = {}
data_bosses = {}
data_bullets = {}
```

```python
def saveResourceData():
    global image_backgrounds, image_player_profiles, image_players , image_monsters
    global data_backgrounds, data_players, data_monsters, data_bosses, data_bullets,

    data_monsters["1"] = actor.Enemy("1", image_monsters["1"], 1,5,5,0.5)
    data_monsters["2"] = actor.Enemy("2", image_monsters["2"], 1,5,5,0.5)
    data_monsters["3"] = actor.Enemy("3", image_monsters["3"], 1,5,5,0.5)
    data_monsters["4"] = actor.Enemy("4", image_monsters["4"], 1,5,5,0.5)
```

3. Pooling에 딕셔너리로 데이터 넣기

```python
play_players = {}
play_monsters = {}
play_bosses = {}
play_bullets = {}
```

```python
def makePooling():
    global data_backgrounds, data_players, data_monsters, data_bosses,
    global play_monsters, play_bullets, play_bosses, play_effects

    play_monsters["1"] = pool.enemyPool(data_monsters["1"])
    play_monsters["2"] = pool.enemyPool(data_monsters["2"])
    play_monsters["3"] = pool.enemyPool(data_monsters["3"])
    play_monsters["4"] = pool.enemyPool(data_monsters["4"])
```

3. 기능 상세 – Background 클래스

1. 배경 데이터 세팅 / 클래스 선언과 함께 매개변수 적용

```
# 배경
data_backgrounds["Intro"] = background.Background(image_backgrounds["Intro"], 1, 0)

data_backgrounds["Desert"] = background.Background(image_backgrounds["Desert"], 0, 3)
data_backgrounds["Sea"] = background.Background(image_backgrounds["Sea"], 0, 3)
data_backgrounds["Mountain"] = background.Background(image_backgrounds["Mountain"], 0, 3)
```

```
class Background:
    def __init__(self, *args):

        self.shape = args[0]        # 배경 이미지
        self.xSpeed = args[1]       # 배경 x 좌표 속도
        self.ySpeed = args[2]       # 배경 y 좌표 속도

        self.width = self.shape.get_rect().size[0]  # 가로 크기
        self.height = self.shape.get_rect().size[1] # 세로 크기
```

2. 현재 실행중인 변수에 배경 데이터 적용하기

```
if scene == 1:
    if stage == 0:
        currentBG = data_backgrounds["Desert"]
    elif stage == 1:
        currentBG = data_backgrounds["Sea"]
    elif stage == 2:
        currentBG = data_backgrounds["Mountain"]
```

3. 배경 움직이기

```
currentBG.Move()
```

```
def Move(self):
    self.xPos += self.xSpeed
    self.xPosSub += self.xSpeed

    if self.xSpeed > 0:
        if self.xPos >= self.width:
            self.xPos = self.xPosSub - self.width
        if self.xPosSub >= self.width:
            self.xPosSub = self.xPos - self.width
    elif self.xSpeed < 0:
        if self.xPos <= -self.width:
            self.xPos = self.xPosSub + self.width
        if self.xPosSub <= -self.width:
            self.xPosSub = self.xPos + self.width
```

3. 기능 상세 – Text 클래스

1. 매니저에 텍스트 데이터 동적으로 넣기

```python
# 텍스트 출력
def SetText(self, id, xPos, yPos, msg, fontSize, color = (255,255,255), alpha = 255):
    if self.move_textList.get(id) == None:
        self.move_textList[id] = Text(id, xPos, yPos, xPos, yPos, 0, msg, fontSize, color, alpha, False)
    else:
        self.move_textList[id].__reset__(xPos, yPos, xPos, yPos, 0, msg, fontSize, color, alpha, False)

    self.move_textList[id].isAlive = True
```

2. 기능별로 함수 구성하기

```python
# 텍스트 출력
def SetText(self, id, xPos, yPos, msg, fontSize, color = (255,255,255), alpha = 255):…

# 블링크 텍스트 출력
def SetTextBlink(self, id, xPos, yPos, msg, fontSize, color = (255,255,255), alpha = 0):…

# 알파 텍스트 출력
def SetTextAlpha(self, id, xPos, yPos, msg, fontSize, color = (255,255,255), alpha = 0):…

# 움직이는 텍스트 출력
def SetTextMove(self, id, startPosX, startPosY, endPosX, endPosY, msg, fontSize, speed = 3, color = (255,255,255), alpha = 255):…

# 폰트 바꾸기
def SetFont(self, id, font):…
```

```python
class Text:
    def __init__(self, *args):
        self.id = args[0]           # id

        self.xPos = args[1]         # 시작위치 X
        self.yPos = args[2]         # 시작위치 Y
        self.xEndPos = args[3]      # 종료위치 X
        self.yEndPos = args[4]      # 종료위치 Y
        self.speed = args[5]        # 움직임 속도
        self.msg = args[6]          # 문자
        self.size = args[7]         # 폰트 크기
        self.color = args[8]        # 색
        self.alpha = args[9]        # 알파
        self.moving = args[10]       # 움직임 상태
        self.isAlive = False

        self.check_custom_font = False
        self.font = ""
```

3. 기능 상세 – Actor 클래스

1. 출현 객체(Actor)가 가지고 있는 공통적인 변수 세팅하기

2. 상속 변수 생성하기 – 필요에 따른 매개변수 추가하기

```
class Enemy(Actor):…

class Player(Actor):…

class Boss(Actor):…
```

```
class Player(Actor):
    def __init__(self, *args):
        super().__init__(*args)
        self.profile = args[6]
        self.initLife = args[7]
        self.initBomber = args[8]
        self.shieldAlive = False
```

3. 데이터 선언하기

```
data_monsters["1"] = actor.Enemy("1", image_monsters["1"], 1,5,5,0.5)
data_players["1"] = actor.Player("1", image_players["1"], 10, 5, 5, 0.3, image_player_profiles["1"],3,3)
data_bosses["1"] = actor.Boss("1", image_bosses["1"], 1000, 5,5,0.5)
```

4. 필요에 따라 함수 사용하기

```
if play_monsters["5"].SpawnDelay(1.0) == True:
    monster = play_monsters["5"].SpawnObj()
    monster.MoveSpawn(-100, 300, 3, -20 + random.randrange(0,40), "Enemy")
    monster.ChangeScale(128,128)
```

```
class Actor:
    def __init__(self, *args):
        self.id = args[0]           # ID
        self.shape = args[1]        # 모양
        self.hp = args[2]           # 체력
        self.power = args[3]        # 파워
        self.speed = args[4]        # 스피드
        self.shotDelay = args[5]    # 총알 딜레이
```

```
# 총알 발사 딜레이
def ShotDelay(self):…

# 크기 변환
def ChangeScale(self, width, height):…

# 방향 변환
def ChangeRotation(self, renderAngle):…

# 움직임 스폰
def MoveSpawn(self, xPos, yPos, speed, angle, tag):…

# 등장 스폰
def AppearSpawn(self, xPos, yPos, tag):…

# 목적지 도달 스폰
def MoveDestination(self, xPos, yPos, play_timer):…

# 데미지
def Hit(self, damage):…
```

3. 기능 상세 – Pool 클래스

※ Pool : 복제 또는 추가되는 객체 데이터를 관리하는 공간

1. Pool에 원본 데이터 넣어두기 및 초기 세팅하기

```
play_monsters["1"] = pool.enemyPool(data_monsters["1"])
play_bosses["1"] = pool.bossPool(data_bosses["1"])
play_bullets["1"] = pool.bulletPool(data_bullets["1"])
```

2. 복제 타이밍에 맞춰서 원본 복제하기

```
if play_monsters["6"].SpawnDelay(1.0) == True:
    monster = play_monsters["6"].SpawnObj()
```

※ 상속받은 데이터는 원본 및 Pool 에 담아놓는 데이터가 제각각 다르다

```
class enemyPool(Pool):
    def __init__(self, *args):
        super().__init__(*args)

        newActor = actor.Enemy(self.objParent.id, self.o
        self.pool.append(newActor)
```

```
class bulletPool(Pool):
    def __init__(self, *args):
        super().__init__(*args)

        newActor = bullet.Bullet(self.objParent.id, self.
        self.pool.append(newActor)
```

```
class Pool:
    def __init__(self, *args):

        self.pool = []              # 담아놀 객체 데이터
        self.objParent = args[0]    # 원본
```

```
class enemyPool(Pool):…

class bulletPool(Pool):…

class bossPool(Pool):…
```

```
# 복제 타이밍에 딜레이 주기
def SpawnDelay(self, delayTime):…

# 원본 복제하기
def SpawnObj(self):…

# 복제된 모든 데이터 죽이기
def AllDead(self):…
```

4. 마무리

DEMO

Q/A

소감

DEMO

Q/A

4. 마무리 - 소감

구 혜 령

하나부터 열까지 모르는 것투성이었다.

김 기 운

코딩을 배우면서 처음으로 만들어 본 게임이라 다소 어려웠지만 배운 내용을 기반으로 코드를 수정해 나가면서 하나하나 만들어 보면서 직접 코딩을 짜보는 것이 얼마나 중요한 것인지 알게 되었습니다.
특히 객체와 클래스 등을 이용하여 수정, 가시성 등을 용이하게 할 수 있고, 중복의 최소화를 통해 코드를 보기 좋게 짤 수 있는 것을 깨달았습니다.
또한 UI를 제작해보면서 코딩을 통한 화면구성이 생각보다 까다롭고 세밀한 작업이지만 익숙해질수록 동작구현이나 디자인하는데 오히려 코드를 쓰면 더 편리한 부분도 많다고 느껴지었습니다.

김 현 구

어떻게 시작해야 할지 막막했는데 팀장님과 팀원분들이 알기 쉽게 설명해주셔서 부족한 부분을 채워나가면서 진행할 수 있었습니다.

이 도 규

프로젝트를 진행하면서, 팀원들이 원활히 프로젝트를 진행할 수 있는 환경을 만드는 것을 중심으로 개발을 진행하였습니다.
함수를 구성하고, 클래스를 구성하면서, 의외의 장소에서 문제가 발생하였고, 이를 해결하기 위한 방법들을 모색하는 시간이 많았던 것으로 보입니다.
구상한 내용보다는 다소 부족한 면이 있지만, 팀원간의 교류로 만들어진 프로젝트에 만족하였습니다.