

JAVA를 이용한 총알 피하기 게임 프로그램의 구현

2021 05 18 이배원

대구가톨릭대학교 컴퓨터공학전공

1. 서 론

Java를 이용한 프로그램으로 하나의 게임을 구현을 해보았다.

Unity를 다룬 적이 있기 때문에 게임을 만드는 것이 먼저 떠올랐다.

이 프로젝트에서 방향키를 입력받아 캐릭터(플레이어)를 움직여, 지정된 위치에서 날라 오는 총알을 피하면서 순서대로 나오는 아이템 10개를 모으는 프로그램을 만들어 보려고 한다.

다이얼로그의 카드의 들어가는 이모티콘 이미지를 제외한 모든 이미지를 Steam의 Aseprite로 직접 그려서 모든 것을 제작 하였다.

먼저 관련 연구에서는 캐릭터를 어떻게 움직일지, 캐릭터가 움직임에 따라 이미지를 바꿔게 할지, 그리고 마지막으로 날라 오는 총알과 직접 움직이고 있는 플레이어를 어떻게 충돌 시킬 수 있는지 알아보고

프로그램 설계에서는 이 프로그램의 구상과 구체적인 설계 계획을 설명하고자 한다.

프로그램 구현에서는 실제로 직접 움직이는 플레이어와 날라 오는 총알들을 충돌시키는 코드와 움직이는 오브젝트의 이미지를 어떻게 바꿔주는 코드를 보면서 설명 해보고자 한다.

2. 관련 연구

캐릭터 움직임:

키보드로부터 입력을 받아 프로그램에서 캐릭터를 움직일 때, 보통 받은 키 값들을 하나의 int형으로 통합해서 방향을 지정해 주거나, 각 각의 키 값들을 boolean형으로 체크 해주는 방식이 있는데 하나의 int형으로 받아버리면 나중에 다른 구현을 추가 할 때나 세밀한 작업이 안 될 수 있기 때문에, 본 프로젝트에서는 boolean으로 각 각의 키 값들을 체크하는 방식을 채택 하였다.

boolean으로 체크를 하는 경우 키보드에 키를 누르는 순간 true로 바꿔주어 이동메소드를 실행하여 캐릭터를 움직여 주면 된다. 키보드를 누르는 중인 경우에는 가속도 혹은 입력 받은 시간에 따라 특정 이벤트를 발생 시키거나 할 수 있지만 본 프로젝트에는 캐릭터 움직임만 구현하기 때문에 키보드에서 키를 떼는 순간 false로 바꿔주는 2개의 키 이벤트를 이용하였다.

이미지 움직임:

사실 어느 메소드 중에서 제일 어려운 과제였다. 캐릭터 움직임의 경우 boolean값을 받아 그 true이면 움직여주면 끝이지만, 이미지를 움직이는 것은 매우 어려운 일이었다. 먼저 이미지는 캐릭터가 움직이고 있을 때 이미지 들이 그 상황에 맞는 이미지로 계속 교체 주어야 한다. 처음 코드를 짤 때 캐릭터의 거리를 계산해서 특정 거리를 이동 할 때마다 이미지가 교체되는 메소드로 코드를 짰지만 캐릭터가 이동하고 있을 때는 이미지가 바뀌어서 괜찮지만 캐릭터가 정지된 상태에서 움직이기 시작해서 특정 거리 이상 가지 않았을 때는 이미지가 바뀌지 않는 것이 문제였다. 그래서 특정 거리를 이동 할 때마다가 아닌 자동으로 이미지가 교체 되게 만들어 주었다. 쓰레드에서 이동을 처리하고 있을 때 int값이 증가하면서 특정 값에 도달 하면 이미지가 교체되게 바꾼 것이다. 보장 값으로 boolean으로 각각의 방향키 마다 처음 눌릴 때 이미지를 바로 교체 해주어 딜레이 없이 이미지가 원활히 바뀔 수 있다.

충돌함수:




충돌함수는 크게 2가지로 나뉜다. 각 오브젝트에 x1,x2,y1,y2지점을 이용해 사각형으로 둘러싸 x축과 y축을 각각 높고 낮음을 계산해서 두 사각형이 서로 겹치면 충돌하는 사각형 충돌이 있지만 본 프로젝트에서는 동그란 총알과 DCU 마스크트인 디쿠 이미지를 사용하기 때문에 원 충돌을 이용하였다.



원 충돌은 두 원의 중심 사이의 거리만 판단하면 된다. 이 거리가 두 원의 반지름 보다 작으면 두 원은 충돌이 일어난다.

피타고라스 정리의 의해서 $a^2+b^2 = c^2$ 을 이용한다.

java에서 기본적으로 제공하는 Math클래스에 sqrt메소드를 이용하면 보다 쉽게 충돌함수를 구현할 수 있다.


3. 프로그램 설계

게임 이름 : DCU의 총알피하기 


게임 장르 : 총알피하기  

플랫폼 : PC 소프트웨어, 600 x 830(해상도)

스타일 : 2D 픽셀아트, Top View

주제 : 총알을 피하면서 10개의 동전 모으기 

게임의 구성: 캐릭터(플레이어), 총알, 캐릭터 이미지, 총알 이미지, 각종 UI이미지, 배경음악, 각종효과음

모든 이미지를 제작, 수정 툴 프로그램(도트) : Steam - Aseprite 

배경이미지 제작 : Unity에셋 - Tileset_Sprite_Sheet(무료 에셋)
원하는 타일들을 잘라서 맵 제작

배경음악(BGM) : 카트라이더 대저택BGM(대저택 댄스 배틀)

효과음 : Unity에셋 - CasualGameSounds(무료 에셋)

클릭효과음 — DM-CGS-32.wav

플레이어_적 충돌효과음 — DM-CGS-42.wav

플레이어_동전 충돌효과음 — DM-CGS-28.wav

게임클리어 효과음 — DM-CGS-18.wav

게임오버 효과음 — DM-CGS-43.wav

세부사항 : 모든 이미지(플레이어, 총알, 동전, UI, 맵) 직접 제작(디쿠 이모티콘 제외)

<다이얼로그>

-게임 클리어, 게임오버시 재 시작 구현

-시작 버튼과 재 시작 버튼 이미지 구분

-처음 게임 클리어일 때와 그 후 재시작한 클리어엔딩은 다름

-게임 클리어일 때 엔딩 다름(디쿠 이모티콘은 편집 후 카드로 제작)

-게임 오버일 때 동전 개수의 따라 엔딩 다름

-일시정지 가능(esc)

-버튼은 클릭 혹은 스페이스 키로 실행 구현

<총알(적)>

- 위, 아래, 왼쪽, 오른쪽, 대각선, 5방향에서 스폰
- 동전을 획득함에 따라 총알 개수가 5개씩 늘어남(난이도 조절)
- 느린 총알, 빠른 총알 이미지 구분

<동전(아이템)>

- 1개만 생성 습득 후 랜덤한 위치에 재생성

<플레이어>

- 이동시 이미지 움직임 적용
- 총알에 맞은 직후 잠깐 동안 무적, 이미지 구현

클래스 : Audio, Object, Item, Enemy, EnemySpawn, Player, PlayerController, Manual, GameManager, Main(총 10개)

Audio : 배경음악, 효과음을 담고 있는 클래스

Object : Item, Enemy의 부모 클래스, 기본적인 설정(위치 값, 콜라이더, 이미지...등등)을 가지고 있음

Item : 동전의 관한 클래스

Enemy : 총알의 관한 클래스

EnemySpawn : Enemy를 만들어 내는 클래스, 모든 총알을 이 클래스에서 관리함

Player : 플레이어의 관한 클래스

PlayerController, : 플레이어의 움직임, 이미지를 관리하는 클래스

Manual : 게임의 대한 설명, 재 시작, 일시정지, 엔딩

GameManager : 모든 클래스를 총괄하는 클래스, 게임의 전반적인 설정, 관리

Main : 메인 클래스

4. 프로그램 구현

플레이어와 총알 충돌 (소스코드) :

```
//충돌함수
public void Clash(Player player) {
    for(int i = 0; i < this.enemys.size(); i++) {
        deltaX = (player.xpos - enemys.get(i).xpos);
        deltaY = (player.ypos - enemys.get(i).ypos);
        dis1 = Math.sqrt( deltaX * deltaX + deltaY * deltaY );
        dis2 = player.collider + enemys.get(i).collider;
        if(player.isInvincible) {
            player.nvincibleCount++;
            if(player.nvincibleCount == 500) {
                player.nvincibleCount = 0;
                player.isInvincible = false;
            }
        }
        //충돌
        if(dis1 < dis2) {
            if(!(player.isInvincible)) {
                player.Health--;
                ClashSound(); //충돌 사운드
            }
            player.isInvincible = true;
            randomPoint = (int)(Math.random() * 31) + 0;
            enemys.get(i).setRespawnPosition(randomPoint);
            enemys.get(i).Respawn(randomPoint);
        }
    }
}
```

플레이어와 총알 충돌 구현 (설명) : 충돌 함수에 플레이어를 매개변수로 받아드려 플레이어의 현재 위치정보를 가져올 수 있게 한다. for문을 돌려 현재 생성된 모든 총알들을 플레이어의 위치에서 뻗 값을 deltaX, deltaY로 저장한 뒤 직각 삼각형 공식을 사용하여 빗변의 길이를 구할 수 있는 Math.sqrt메소드를 이용하여 그 값을 dis1에 저장을 한다.

dis2에는 플레이어의 콜라이더, 즉 플레이어의 반지름과 총알의 콜라이더를 더한 값을 저장한다.

dis1에 저장되어있는 총알과 플레이어 사이의 거리와 dis2의 거리를 계산하여 dis2가 더 크다면 충돌이 일어나는 것이다. 충돌된 총알은 다시 랜덤함수를 돌려 새로 리스폰 된다. 아이템 충돌도 총알과 마찬가지로 방식 사용한다.

플레이어 이미지 (소스코드) :

```
public void PlayerImage() {
    //무적 중일때 깜빡임
    if(player.isInvincible) {
        if(keyUp) {
            if(playerImageCount >= (playerSpeed / 2)) {
                player.playerImage = player.playerImgInvincible;
            }
            if(playerImageCount >= playerSpeed) {
                if(curPlayerImage == 4)
                    curPlayerImage = 0;
                player.playerImage = player.playerImgBack[curPlayerImage++];
                playerImageCount = 0;
            }
            playerImageCount++;
        }
        (생략 keyDown, keyLeft, keyRight)
        .
        .
        .
        else {
            if(playerImageCount >= (playerSpeed / 2)) {
                player.playerImage = player.playerImgInvincible;
            }
            if(playerImageCount >= playerSpeed) {
                switch(lastKey) {
                    case 0:
                        player.playerImage = player.playerImgBack[0];
                        break;
                    case 1:
                        player.playerImage = player.playerImgFront[0];
                        break;
                    case 2:
                        player.playerImage = player.playerImgLeft[0];
                        break;
                    case 3:
                        player.playerImage = player.playerImgRight[0];
                        break;
                }
                playerImageCount = 0;
            }
        }
    }
}
```

```

        playerImageCount++;
    }
    //이미지(움직임)
else if(keyUp) {
    if(isFirstKeyUp) {
        player.playerImage = player.playerImgBack[0];
        isFirstKeyUp = false;
    }
    if(playerImageCount >= playerSpeed) {
        if(curPlayerImage == 4)
            curPlayerImage = 0;
        player.playerImage = player.playerImgBack[curPlayerImage++];
        playerImageCount = 0;
    }
    playerImageCount++;
}
(생략 keyDown, keyLeft, keyRight)
.
.
.
//무적 이미지 풀어주기(플레이어 사라짐 방지)
else if(!player.isInvincible) {
    if(keyUp) {
        player.playerImage = player.playerImgBack[0];
    }
    else if(keyDown) {
        player.playerImage = player.playerImgFront[0];
    }
    else if(keyLeft) {
        player.playerImage = player.playerImgLeft[0];
    }
    else if(keyRight) {
        player.playerImage = player.playerImgRight[0];
    }
    else {
        switch(lastKey) {
            case 0:
                player.playerImage = player.playerImgBack[0];
                break;
            case 1:
                player.playerImage = player.playerImgFront[0];

```

```

        break;
    case 2:
        player.playerImage = player.playerImgLeft[0];
        break;
    case 3:
        player.playerImage = player.playerImgRight[0];
        break;
    }
}
}
}

```

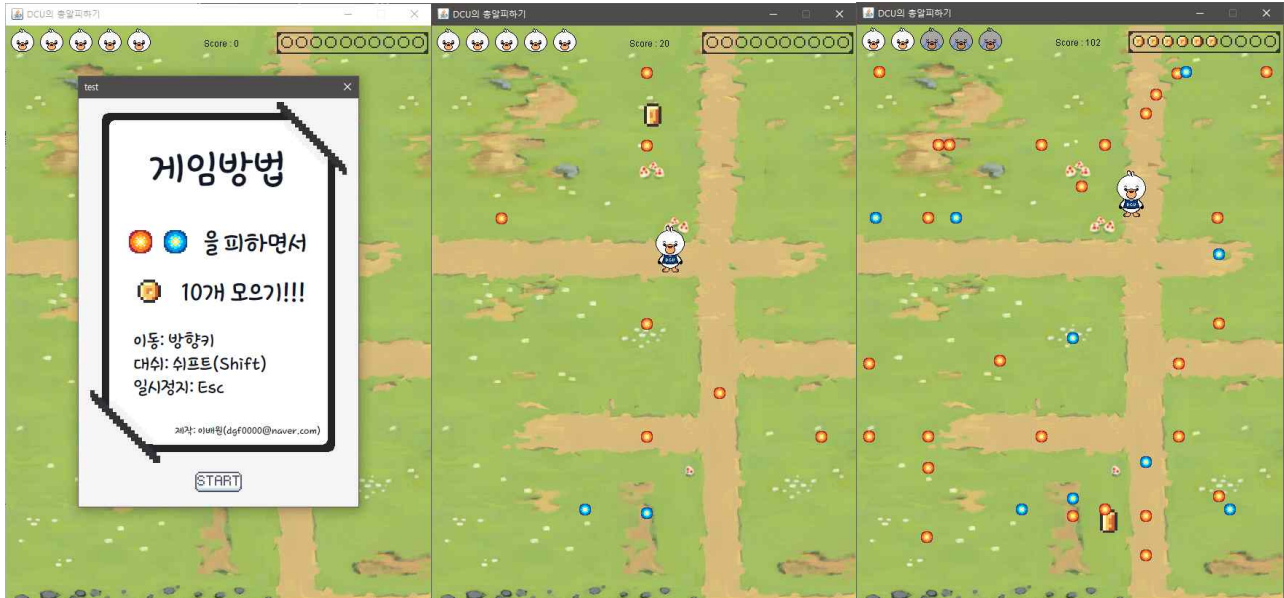
플레이어 이미지 (설명) : 움직임은 크게 3가지로 나눌 수 있다. 플레이어가 총알에 맞아 이미지가 깜빡이는 무적 이미지, 일반 움직임 이미지, 무적이 풀린 후 플레이어가 사라짐을 방지하는 이미지, 이렇게 3가지이다.

무적 이미지를 보기에 앞서 먼저 일반 움직임 이미지를 보면 if문에서 키 이벤트로부터 받아온 boolean형태의 keyUp, keyDown, keyLeft, keyRight들이 참에 해당하면 이미지가 바뀌게 된다. playerImageCount란 스레드가 돌아가면서 playerImageCount가 1씩 증가가 되고 먼저 설정해 놓은 playerSpeed 값보다 크거나 같아지면 이미지가 다음 장면으로 넘어가도록 설정을 하였다. 이미지는 왼발, 중간, 오른발 3가지만 Aeprite(도트 툴)로 제작을 해서 걸어가기 위해서 중간, 왼발, 중간, 오른발 4가지가 필요하기 때문에 중간의 복사해서 추가하고, 현재 이미지 즉, curPlayerImage가 4가 되면 다시 처음으로 돌아가게 만들어 주었다.

무적 이미지는 일반 움직임 이미지에서 if 하나가 더 추가 되었는데 중간, 왼발, 중간, 오른발 4가지 이미지 사이사이에 투명한 이미지를 추가하여 주었다.

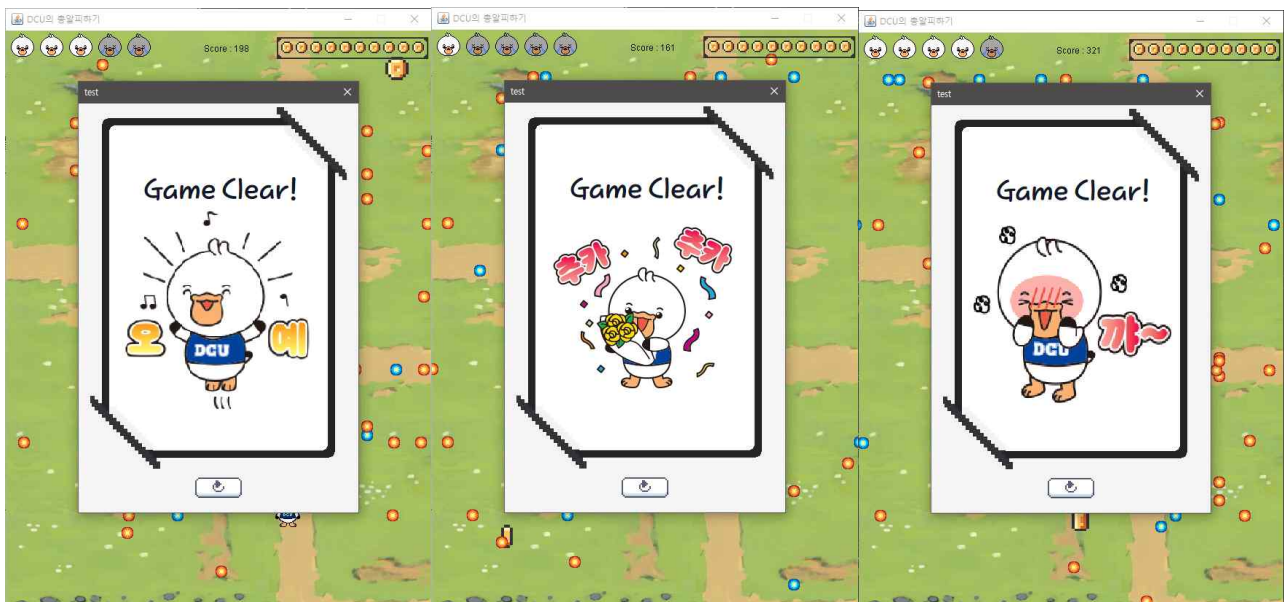
마지막으로 사라짐을 방지하는 이미지는 플레이어가 무적이 되면서 사라졌을 때 키를 떼버리면 계속 플레이어가 사라지게 되거나 다른 이미지가 불러와지는 등 각종 버그를 방지 하기위해 무적이 끝나면 현재 진행하고 있는 방향의 이미지를 덧씌워 주었다.

5. 프로그램의 동작과 결과



<처음 시작>

<게임중>



<실행 후 처음 클리어>

<클리어 이후 추가 클리어>

<스코어300이상 클리어>



<게임 오버>

<코인 8개이상먹고 게임오버>

6. 결과 분석 및 토의

플레이어와 총알의 충돌이 일어날 때 이미지들이 서로 충돌이 일어나지 않았는데 충돌이 일어나거나 닿아있는데 충돌이 안 일어나는 버그가 있었다. 그래서 이미지의 위치와 실제 포인트의 차이가 얼마나 나는지 알아보기 위해 `buffG.drawString` 함수로 "●"을 플레이어 위치에 그려보았더니 상당히 다른 위치에 있었다. 생각해보면 실제 포인트 위치에서 좌 상단부터 그림이 그려지는데 그 것을 잊고 있었다. 그래서 "●"을 이미지 중간에 들어갈 수 있도록 수정값을 더해주어서 해결 하였다.

다이얼로그를 이용하면서 `esc`키를 눌러 일시정지를 시킬 때 플레이어컨트롤러의 킷값들이 `true`에서 다이얼로그를 띄우고 다시 게임으로 돌아가면 스스로 움직이는 버그를 고쳐야만 했다. 다이얼로그에 같은 킷값을 `esc`를 누르면 현재의 킷값들을 복사 후 전달 해주도록 바꾸고 다이얼로그에서도 같이 킷값을 받아 반대로 전해줄 수 있도록 변경 하였다.

다이얼로그를 사용하여 설명 및 엔딩 창을 띄울 때 모달 다이얼로그를 이용하여 `isVisible()`을 이용하여 게임을 리셋 시켰는데 무슨 오류인지 될 때도 있고 안 될 때도 있었다. 그래서 모달리스 다이얼로그를 사용하고 전제적으로 그에 맞게 수정해서 버그를 고쳤다.

7. 결론

이 프로젝트에서 이번 2021학년 1학기 자바프로그래밍에서 배운 모든 장(4장~14장)의 기능들을 전부 활용하여 구현해 보았다. 사실 모든 것을 혼자 힘으로 만들고 싶었지만 혼자 힘으로 주어진 기간 안에서 모든 것을 해결해야 한다면 상당히 완성도가 낮아질 것이다. 그 이유로 이 프로젝트에서 게임을 주제로 만들었는데 게임은 단연 그래픽과 사운드가 90프로를 차지한다고 생각이 들기 때문에 그만큼 혼자 힘으로 해결하면 많이 힘들어질 것이다.

음악을 전공한 것도 아니고 그림을 잘 그리는 것도 아니기 때문에 엔딩카드에 쓴 이모티콘, 배경 음악, 효과음, 맵을 만드는 타일 이 4가지만 제작을 안 하고 가져와서 사용해 보았다.

디쿠(DCU마스코트) 및 UI를 그려내고 맵을 만들고 이모티콘을 수정하는 시간이 상당히 걸리긴 했다. 하지만 좀 더 나은 게임을 만들기 위해 열심히 제작해 보았다. Unity를 다룬 적이 있어서 게임을 만드는 데는 보다 쉬웠지만 상당히 어렵긴 했다. 모든 편의를 제공해주는 Unity와는 다르게 모든 것을 직접 코드를 짜야한다는 것은 생각만큼 쉽지가 않았다. 특히 플레이어를 움직일 때 이미지를 변경해야하는 코드를 작성하는 것을 꽤나 오래 걸린 것 같다. 생각한 것들을 코드로 옮긴다는 게 쉽지만은 않은 것 같다.

클래스와 코드를 작성하면 할수록 보기도 힘들어지고 수정하기도 까다로워지는 이번 텀 프로젝트를 하면서 제대로 느꼈다. 앞으로 좀 더 많은 코드들을 접하게 될 텐데 더 많은 코드들을 보고 알고리즘들을 공부해야겠다고 느꼈다.

8. 참고문헌

배경음악(BGM) : <https://youtu.be/1c6cGpwaHwI>
디쿠 카드 이모티콘 : http://ocean.cu.ac.kr/photo_view.html?idx=1406
맵 타일 에셋 : <https://assetstore.unity.com/packages/2d/environments/2d-fantasy-forest-tileset-19553>
효과음 에셋 : <https://assetstore.unity.com/packages/audio/sound-fx/free-casual-game-sfx-pack-54116>

9. 별첨 (소스 코드와 주석)

< Audio >

package script;

```
import java.io.File;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.Clip;

public class Audio implements Runnable {
    AudioInputStream stream//스트림
    File file//생성되는파일
    Clip clip//클립
    String fileName//현재파일이름
    boolean isPlaying = false//실행중

    public Audio(String fileName) { //매개변수의이름에따라file이다르게생성
        this.fileName = fileName
        switch(fileName) {
            case "BGMSound":
                this.file = new File("audio/KartRiderMusic_Mansion Dance Battle.wav");
                break
            case "clickSound":
                this.file = new File("audio/DM-CGS-32.wav");
                break
            case "PlayerClashEnemySound":
                this.file = new File("audio/DM-CGS-41.wav");
                break
            case "PlayerClashItemSound":
                this.file = new File("audio/DM-CGS-28.wav");
                break
            case "GameClearSound":
                this.file = new File("audio/DM-CGS-18.wav");
                break
            case "GameOverSound":
                this.file = new File("audio/DM-CGS-43.wav");
                break
        }
        try {
            stream = AudioSystem.getAudioInputStream(file);
            this.clip = AudioSystem.getClip();
            this.clip.open(stream);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    //스레드호출로플레이
    @Override
    public void run() {
        try {
            this.clip.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

< Object >

```
package script;

import java.awt.*;
//Item, Enemy의 부모 오브젝트 5장 상속을 위해 구현
public abstract class Object {
    int respawnPosition//생성되는 위치
    int xpos//x좌표
    int ypos//y좌표
    float speed//스피드(이미지 or 움직임)
    float collider//콜라이더
    Toolkit imageTool = Toolkit.getDefaultToolkit();//이미지 킷
    Image image//이미지
}
```

< Item >

```
package script;

import java.awt.Image;

public class Item extends Object implements Runnable{
    Player player//플레이어

    boolean isCrash = false//플레이어와 충돌
    boolean isStop = false//움직임 통제

    int itemScore = 0;//현재 아이템 스코어

    //코인 이미지 관련 설정
    Image itemimage[] = new Image[4];
    final int imageSpeed = 5;
    int itemimageCount = 0;
    int curitemImage = 0;

    //충돌 거리를 위한 변수
    double dis1, dis2;
    double deltaX, deltaY
    //플레이어 값을 받기 위한 매개 변수
    public Item(Player player) {
        this.player = player
        this.xpos = 300;//초기 위치 값 X
        this.ypos = 150;//초기 위치 값 Y
        this.collider = 16f;//코인의 콜라이더(반지름 길이)
        this.image = imageTool.getImage("Image/Object/Coin_0.png");//코인 이미지 초기화
        for(int i = 0; i < 4; i++) {
            this.itemimage[i] = imageTool.getImage("Image/Object/Coin_"+i+".png");
        }
    }
    //아이템 위치 변경
    public void Respawn() {
        int spawnPointX = (int)(Math.random() * (550 - 50)) + 50;
        int spawnPointY = (int)(Math.random() * (750 - 80)) + 80;

        this.xpos = spawnPointX
        this.ypos = spawnPointY
    }
    //아이템 이미지(움직임)
    public void ItemImage() { //스레드로 카운트가 돌아가면서 이미지 스피드와 값이 같아지면 1씩 증가해서 이미지 교체
        if(itemimageCount == imageSpeed) {
            if(curitemImage == 4)
                curitemImage = 0;
            this.image = itemimage[curitemImage++];
            itemimageCount = 0;
        }
        itemimageCount++;
    }
}
//충돌 함수
```

```

public void Clash(Player player) {
    this.deltaX = (player.xpos - this.xpos);
    this.deltaY = (player.ypos - this.ypos);
    this.dis1 = Math.sqrt( deltaX * deltaX +  deltaY * deltaY );
    this.dis2 = player.collider + this.collider
    //충돌
    if(dis1 < dis2) {
        player.GetItemNum++;
        itemScore += 10;
        Respawn();
        ClashSound();
    }
}
//충돌사운드
public void ClashSound() {
    Audio crashSound = new Audio("PlayerClashItemSound");
    Thread ItemTh = new Thread(crashSound);
    ItemTh.start();
}
//스레드
@Override
public void run() {
    try {
        while(true){
            ItemImage();
            Clash(player);
            Thread.sleep(10);
        }
    } catch (Exception e){
        e.printStackTrace();
    }
}
}
}

```

< Enemy >

package script;

```
public class Enemy extends Object {
    int enemyName//생성되는총알이름
    int randomPoint//랜덤포인트를받기위한변수
    int score = 0;//이총알의점수

    public Enemy(int respawnPosition, int enemyName) { //리스폰될위치, 총알이름

        this.respawnPosition = respawnPosition
        this.enemyName = enemyName
        this.collider = 8f;

        switch(enemyName) {
            //빨간색총알
            case 0:
                this.image = imageTool.getImage("Image/Object/BulletA.png");
                this.speed = 1f;
                break
            //파란색총알
            case 1:
                this.image = imageTool.getImage("Image/Object/BulletB.png");
                this.speed = 5f;
                break
        }
        Respawn(respawnPosition);//지정된위치에총알생성
    }
    //총알의위치리셋을위한함수
    public void setRespawnPosition(int respawnPosition) {
        this.respawnPosition = respawnPosition
    }
    //총알이생성되는위치
    public void Respawn(int respawnPosition) { //매개변수의값에따라다른곳에총알생성
        switch(respawnPosition) {
            //top
            case 0:
                this.xpos = 100;
                this.ypos = -10;
                break
            case 1:
                this.xpos = 200;
                this.ypos = -10;
                break
            case 2:
                this.xpos = 300;
                this.ypos = -10;
                break
            case 3:
                this.xpos = 400;
                this.ypos = -10;
                break
            case 4:
                this.xpos = 500;
                this.ypos = -10;
                break

            //bottom
            case 5:
                this.xpos = 100;
                this.ypos = 850;
                break
            case 6:
                this.xpos = 200;
                this.ypos = 850;
                break
            case 7:
                this.xpos = 300;
                this.ypos = 850;
                break
            case 8:
                this.xpos = 400;
                this.ypos = 850;
                break
            case 9:
                this.xpos = 500;
                this.ypos = 850;
                break
            //left
```



```

case 10:
    this.xpos = -10;
    this.ypos = 100;
    break
case 11:
    this.xpos = -10;
    this.ypos = 200;
    break
case 12:
    this.xpos = -10;
    this.ypos = 300;
    break
case 13:
    this.xpos = -10;
    this.ypos = 400;
    break
case 14:
    this.xpos = -10;
    this.ypos = 500;
    break
case 15:
    this.xpos = -10;
    this.ypos = 600;
    break
case 16:
    this.xpos = -10;
    this.ypos = 700;
    break
//right
case 17:
    this.xpos = 610;
    this.ypos = 100;
    break
case 18:
    this.xpos = 610;
    this.ypos = 200;
    break
case 19:
    this.xpos = 610;
    this.ypos = 300;
    break
case 20:
    this.xpos = 610;
    this.ypos = 400;
    break
case 21:
    this.xpos = 610;
    this.ypos = 500;
    break
case 22:
    this.xpos = 610;
    this.ypos = 600;
    break
case 23:
    this.xpos = 610;
    this.ypos = 700;
    break
//diagonal TL
case 24:
    this.xpos = 45;
    this.ypos = -10;
    break
case 25:
    this.xpos = -10;
    this.ypos = 45;
    break
//diagonal TR
case 26:
    this.xpos = 555;
    this.ypos = -10;
    break
case 27:
    this.xpos = 610;
    this.ypos = 45;
    break
//diagonal BL
case 28:
    this.xpos = -10;
    this.ypos = 755;
    break
case 29:
    this.xpos = 45;

```

```

        this.ypos = 850;
        break
    //diagonal BT
    case 30:
        this.xpos = 555;
        this.ypos = 850;
        break
    case 31:
        this.xpos = 610;
        this.ypos = 755;
        break
    }
}
//총알움직임구현
public void Move() { //총알의시작위치에따라움직임이다르게설정//설정된위치를벗어나면재배치
    //top
    if(this.respawnPosition <= 4) {
        this.ypos += this.speed
    }
    //bottom
    else if(this.respawnPosition <= 9) {
        this.ypos -= this.speed
    }
    //left
    else if(this.respawnPosition <= 16) {
        this.xpos += this.speed
    }
    //right
    else if(this.respawnPosition <= 23) {
        this.xpos -= this.speed
    }
    //diagonal
    else if(this.respawnPosition <= 25) {
        this.xpos += this.speed
        this.ypos += this.speed
    }
    else if(this.respawnPosition <= 27) {
        this.xpos -= this.speed
        this.ypos += this.speed
    }
    else if(this.respawnPosition <= 29) {
        this.xpos += this.speed
        this.ypos -= this.speed
    }
    else if(this.respawnPosition <= 31) {
        this.xpos -= this.speed
        this.ypos -= this.speed
    }
    //Reset respawn
    if(this.xpos > 900) {
        Relocation();
    }
    //top respawn
    else if(this.ypos > 850) {
        Relocation();
        this.score++;
    }
    //bottom respawn
    else if(this.ypos < -50) {
        Relocation();
        this.score++;
    }
    //left respawn
    else if(this.xpos > 650) {
        Relocation();
        this.score++;
    }
    //right respawn
    else if(this.xpos < -50) {
        Relocation();
        this.score++;
    }
}
//재배치
public void Relocation() {
    randomPoint = (int)(Math.random() * 31) + 0;
    setRespawnPosition(randomPoint);
    Respawn(respawnPosition);
}
}

```

<EnemySpawn>

```
package script;

import java.util.Vector;

public class EnemySpawn implements Runnable{
    Player player//충돌을위해필요
    Vector<Enemy> enemys//Vector를이용해총알생성
    int spawnPoint//총알의위치
    int randomPoint//총알의위치를랜덤하게배치시키기위한변수
    int spawnName//총알이름
    int addEnemyCount = 0;//총알이추가되기위한변수
    int enemysScore = 0;//총알들의총점수
    int sumScore = 0;//총점수에넣어주기위한변수
    boolean isCrash = false//플레이어와충돌
    boolean isStop = true//움직임통제

    //충돌거리를위한변수
    double dis1, dis2
    double deltaX, deltaY
    //플레이어위치값을받기위한매개변수
    public EnemySpawn(Player player) {
        this.player = player
        this.enemys = new Vector<>();

        //적생성
        for(int i = 0; i < 10; i ++){
            spawnPoint = (int)(Math.random() * 31) + 0;
            if(i < 8)
                enemys.add(new Enemy(spawnPoint , 0));
            else
                enemys.add(new Enemy(spawnPoint , 1));
        }
    }
    //적생성을위한테스트함수모든리스폰장소에서나옴
    public void TestEnemy() {
        for(int i = 0; i < 5; i ++){enemys.add(new Enemy(i , 0));}
        for(int i = 5; i < 10; i ++){enemys.add(new Enemy(i , 0));}
        for(int i = 10; i < 17; i ++){enemys.add(new Enemy(i , 0));}
        for(int i = 17; i < 24; i ++){enemys.add(new Enemy(i , 0));}
        for(int i = 24; i < 32; i ++){enemys.add(new Enemy(i , 0));}
    }
    //적증가
    public void AddEnemys(int num) { //랜덤한값으로위치와총알선택
        for(int i = 0; i < num i ++){
            spawnPoint = (int)(Math.random() * 31) + 0;
            spawnName = (int)(Math.random() * 9) + 0;
            if(spawnName < 6)//6할은빨간색4할은파란색생성
                enemys.add(new Enemy(spawnPoint , 0));
            else
                enemys.add(new Enemy(spawnPoint , 1));
        }
    }
    //아이템획득할때마다적증가함수호출
    public void CheckPlayerGetItemNum() {
        if(player.GetItemNum != addEnemyCount) {
            AddEnemys(5);
            addEnemyCount++;
        }
    }
    //움직임
    public void EnemyMove() {
        for(int i = 0; i < this.enemys.size(); i++){
            if(!isStop) {
                this.enemys.get(i).Move();
            }
            sumScore = this.enemys.get(i).score//움직이는함수에서스코어도같이처리
            enemysScore += sumScore //
            this.enemys.get(i).score = 0;
            sumScore = 0;
        }
    }
    //모든총알위치초기화
    public void AllEnemyReset() {
        for(int i = 0; i < this.enemys.size(); i++){
            Enemy enemy = this.enemys.get(i);
            enemy.xpos = 1000;
            enemy.ypos = 1000;
        }
    }
}
```

```

    }
}
//충돌함수
public void Clash(Player player) {
    for(int i = 0; i < this.enemys.size(); i++) {
        deltaX = (player.xpos - enemys.get(i).xpos);
        deltaY = (player.ypos - enemys.get(i).ypos);
        dis1 = Math.sqrt( deltaX * deltaX + deltaY * deltaY );
        dis2 = player.collider + enemys.get(i).collider
        if(player.isInvincible) {
            player.nvincibleCount++;
            if(player.nvincibleCount == player.nvincibleTime) {
                player.nvincibleCount = 0;
                player.isInvincible = false
            }
        }
        //충돌
        if(dis1 < dis2) {
            if(!player.isInvincible)) { //무적이아날때
                player.Health--;
                ClashSound();
            }
            player.isInvincible = true
            randomPoint = (int)(Math.random() * 31) + 0; //충돌한충알은다시리스폰
            enemys.get(i).setRespawnPosition(randomPoint);
            enemys.get(i).Respawn(randomPoint);
        }
    }
}
//충돌사운드
public void ClashSound() {
    Audio crashSound = new Audio("PlayerClashEnemySound");
    Thread enemySpawnTh = new Thread(crashSound);
    enemySpawnTh.start();
}
//스레드
@Override
public void run() {
    try {
        while(true){
            EnemyMove();
            Clash(player);
            CheckPlayerGetItemNum();
            Thread.sleep(10);
        }
    } catch (Exception e){
        e.printStackTrace();
    }
}
}
}

```

<Player>

```
package script;
```

```
import java.awt.*;
```

```
import java.io.File;
```

```
import java.io.IOException;
```

```
import javax.imageio.ImageIO;
```

```
public class Player {
```

```
    //이미지 설정
```

```
    Toolkit imageTool = Toolkit.getDefaultToolkit();
```

```
    Image playerImage = imageTool.getImage("Image/Player/DcuFront0.png");
```

```
    //이미지 움직임을 위한 배열
```

```
    Image playerImgFront[] = new Image[4];
```

```
    Image playerImgBack[] = new Image[4];
```

```
    Image playerImgLeft[] = new Image[4];
```

```
    Image playerImgRight[] = new Image[4];
```

```
    Image playerImgInvincible = imageTool.getImage("Image/Player/Invincible.png");
```

```
    //플레이어 설정값
```

```
    int Health = 5;                //체력
```

```
    int GetItemNum = 0;            //모은 아이템 갯수
```

```
    float Speed = 3f;              //현재스피드
```

```
    float normalSpeed = 2.5f;      //기본스피드
```

```
    float dashSpeed = 5f;          //대쉬스피드
```

```
    float collider = 26f;          //콜라이더 크기
```

```
    int xpos = 300;                //초기위치X
```

```
    int ypos = 400;                //초기위치Y
```

```
    boolean isInvincible = false;  //무적?
```

```
    int nvincibleCount = 0;        //무적카운트를 위한 변수
```

```
    int nvincibleTime = 1500;      //무적시간
```

```
    //플레이어 이미지 얻기
```

```
    public Player() {
```

```
        try {
```

```
            GetPlayerImage();
```

```
        } catch (IOException e) {
```

```
            e.printStackTrace();
```

```
        }
```

```

}

//플레이어 이미지 세팅
public void GetPlayerImage() throws IOException{
    File suorceFrontImage;
    File suorceBackImage ;
    File suorceLeftImage;
    File suorceRightImage;

    for(int i = 0; i < 4; i++) {
        suorceFrontImage = new File("Image/Player/DcuFront"+i+".png");
        suorceBackImage = new File("Image/Player/DcuBack"+i+".png");
        suorceLeftImage = new File("Image/Player/DcuLeft"+i+".png");
        suorceRightImage = new File("Image/Player/DcuRight"+i+".png");

        playerImgFront[i] = ImageIO.read(suorceFrontImage);
        playerImgBack[i] = ImageIO.read(suorceBackImage);
        playerImgLeft[i] = ImageIO.read(suorceLeftImage);
        playerImgRight[i] = ImageIO.read(suorceRightImage);
    }
}

//플레이어 리셋
public void playerReset() {
    this.Health = 5;
    this.GetItemNum = 0;
    this.xpos = 300;
    this.ypos = 400;
    this.playerImage = playerImgFront[0];
}
}

```

<PlayerController>

```
package script;

import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;

public class PlayerController extends KeyAdapter implements Runnable{
    //움직일 플레이어
    Player player;

    //이미지 움직임을 위한 설정
    int playerImgSpeed = 10;
    int playerImageCount = 0;
    int platerInvincibleCount = 0;
    int curPlayerImage = 1;
    int lastKey = 1;    //플레이어가 바라보고있는 곳, 마지막으로 받은 키, 메뉴얼을 위한 값

    //이미지 설정값
    boolean isFirstKeyUp = true;
    boolean isFirstKeyDown = true;
    boolean isFirstKeyLeft = true;
    boolean isFirstKeyRight = true;

    //키처리
    boolean isStop = true;
    boolean keyUp = false;
    boolean keyDown = false;
    boolean keyLeft = false;
    boolean keyRight = false;
    boolean keyShift = false;
    boolean keyEsc = false;

    //플레이어 초기화 생성자
    public PlayerController(Player player) {
        this.player = player;
    }

    //키 true
    @Override
    public void keyPressed(KeyEvent e) {
```

```

switch (e.getKeyCode()){
    case KeyEvent.VK_UP :
        this.keyUp = true;
        this.keyDown = false; //하나의 키를 제외한 모든 키 false, 즉 위 아래 왼쪽 오른쪽만 방향으로만 이동 가능
        this.keyLeft = false;
        this.keyRight = false;
        break;
    case KeyEvent.VK_DOWN :
        this.keyDown = true;
        this.keyUp = false;
        this.keyLeft = false;
        this.keyRight = false;
        break;
    case KeyEvent.VK_LEFT :
        this.keyLeft = true;
        this.keyUp = false;
        this.keyDown = false;
        this.keyRight = false;
        break;
    case KeyEvent.VK_RIGHT :
        this.keyRight = true;
        this.keyUp = false;
        this.keyDown = false;
        this.keyLeft = false;
        break;
    case KeyEvent.VK_SHIFT :
        this.keyShift = true;
        break;
    case KeyEvent.VK_ESCAPE :
        this.keyEsc = true;
        break;
}
}
//7] false
@Override
public void keyReleased(KeyEvent e) {
    switch (e.getKeyCode()){
        case KeyEvent.VK_UP :
            this.keyUp = false;
            this.isFirstKeyUp = true;
            this.lastKey = 0;

```



```

        break;
    case KeyEvent.VK_DOWN :
        this.keyDown = false;
        this.isFirstKeyDown = true;
        this.lastKey = 1;
        break;
    case KeyEvent.VK_LEFT :
        this.keyLeft = false;
        this.isFirstKeyLeft = true;
        this.lastKey = 2;
        break;
    case KeyEvent.VK_RIGHT :
        this.keyRight = false;
        this.isFirstKeyRight = true;
        this.lastKey = 3;
        break;
    case KeyEvent.VK_SHIFT :
        this.keyShift = false;
        break;
    }
}
//이동
public void keyProcess() {
    if(this.keyUp) {
        if(player.ypos >= 80)
            this.player.ypos -= player.Speed;
    }
    if(this.keyDown){
        if(player.ypos < 780)
            this.player.ypos += player.Speed;
    }
    if(this.keyLeft){
        if(player.xpos > 30)
            this.player.xpos -= player.Speed;
    }
    if(this.keyRight){
        if(player.xpos < 560)
            this.player.xpos += player.Speed;
    }
    //삼항연산자
    this.player.Speed = (this.keyShift) ? player.dashSpeed : player.normalSpeed;
}

```

```

        this.playerImgSpeed = (this.keyShift) ? 6 : 10;
    }
    //든 방향키 통제
    public void AllKeyFalse() {
        this.keyUp = false;
        this.keyDown = false;
        this.keyLeft = false;
        this.keyRight = false;
        this.keyShift = false;
    }
    // Player 이미지(움직임) 처리
    public void PlayerImage() {
        //무적 중일때 깜빡임
        if(player.isInvincible) {
            if(keyUp) {
                if(playerImageCount >= (playerImgSpeed / 2)) {           //플레이어 이미지 카운터가 증가하면서 이미지스피드
의 1/2일때 투명, 1일 때 걷는 이미지
                    player.playerImage = player.playerImgInvincible; //이미지 사이사이에 투명 이미지 추가
                }
                if(playerImageCount >= playerImgSpeed) {
                    if(curPlayerImage == 4)
                        curPlayerImage = 0;
                    player.playerImage = player.playerImgBack[curPlayerImage++];
                    playerImageCount = 0;
                }
                playerImageCount++;
            }
            else if(keyDown) {
                if(playerImageCount >= (playerImgSpeed / 2)) {
                    player.playerImage = player.playerImgInvincible;
                }
                if(playerImageCount >= playerImgSpeed) {
                    if(curPlayerImage == 4)
                        curPlayerImage = 0;
                    player.playerImage = player.playerImgFront[curPlayerImage++];
                    playerImageCount = 0;
                }
                playerImageCount++;
            }
            else if(keyLeft) {
                if(playerImageCount >= (playerImgSpeed / 2)) {

```

```

        player.playerImage = player.playerImgInvincible;
    }
    if(playerImageCount >= playerImgSpeed) {
        if(curPlayerImage == 4)
            curPlayerImage = 0;
        player.playerImage = player.playerImgLeft[curPlayerImage++];
        playerImageCount = 0;
    }
    playerImageCount++;
}

else if(keyRight) {
    if(playerImageCount >= (playerImgSpeed / 2)) {
        player.playerImage = player.playerImgInvincible;
    }
    if(playerImageCount >= playerImgSpeed) {
        if(curPlayerImage == 4)
            curPlayerImage = 0;
        player.playerImage = player.playerImgRight[curPlayerImage++];
        playerImageCount = 0;
    }
    playerImageCount++;
}

else { //가만히 있을 때 무적 이미지
    if(playerImageCount >= (playerImgSpeed / 2)) {
        player.playerImage = player.playerImgInvincible;
    }
    if(playerImageCount >= playerImgSpeed) { //기본 걷기 이미지
        switch(lastKey) {
            case 0:
                player.playerImage = player.playerImgBack[0];
                break;
            case 1:
                player.playerImage = player.playerImgFront[0];
                break;
            case 2:
                player.playerImage = player.playerImgLeft[0];
                break;
            case 3:
                player.playerImage = player.playerImgRight[0];
                break;
        }
    }
}

```

```

        playerImageCount = 0;
    }
    playerImageCount++;
}
}
//기본 이미지(움직임)
else if(keyUp) {
    if(isFirstKeyUp) {
        player.playerImage = player.playerImgBack[0];
        isFirstKeyUp = false;
    }
    if(playerImageCount >= playerImgSpeed) {
        if(curPlayerImage == 4)
            curPlayerImage = 0;
        player.playerImage = player.playerImgBack[curPlayerImage++];
        playerImageCount = 0;
    }
    playerImageCount++;
}
else if(keyDown) {
    if(isFirstKeyDown) {
        player.playerImage = player.playerImgFront[0];
        isFirstKeyDown = false;
    }
    if(playerImageCount >= playerImgSpeed) {
        if(curPlayerImage == 4)
            curPlayerImage = 0;
        player.playerImage = player.playerImgFront[curPlayerImage++];
        playerImageCount = 0;
    }
    playerImageCount++;
}
else if(keyLeft) {
    if(isFirstKeyLeft) {
        player.playerImage = player.playerImgLeft[0];
        isFirstKeyLeft = false;
    }
    if(playerImageCount >= playerImgSpeed) {
        if(curPlayerImage == 4)
            curPlayerImage = 0;
        player.playerImage = player.playerImgLeft[curPlayerImage++];

```

```

        playerImageCount = 0;
    }
    playerImageCount++;
}
else if(keyRight) {
    if(isFirstKeyRight) {
        player.playerImage = player.playerImgRight[0];
        isFirstKeyRight = false;
    }
    if(playerImageCount >= playerImgSpeed) {
        if(curPlayerImage == 4)
            curPlayerImage = 0;
        player.playerImage = player.playerImgRight[curPlayerImage++];
        playerImageCount = 0;
    }
    playerImageCount++;
}
//무적 이미지 풀어주기(플레이어 사라짐 방지)
else if(!player.isInvincible) {
    if(keyUp) {
        player.playerImage = player.playerImgBack[0];
    }
    else if(keyDown) {
        player.playerImage = player.playerImgFront[0];
    }
    else if(keyLeft) {
        player.playerImage = player.playerImgLeft[0];
    }
    else if(keyRight) {
        player.playerImage = player.playerImgRight[0];
    }
    else {
        switch(lastKey) {
            case 0:
                player.playerImage = player.playerImgBack[0];
                break;
            case 1:
                player.playerImage = player.playerImgFront[0];
                break;
            case 2:
                player.playerImage = player.playerImgLeft[0];

```

```

        break;
    case 3:
        player.playerImage = player.playerImgRight[0];
        break;
    }
}
}

@Override
public void run() {
    try {
        while(true){
            if(!isStop) {
                keyProcess();
                PlayerImage();
            }
            Thread.sleep(10);
        }
    } catch (Exception e){
        e.printStackTrace();
    }
}
}

```

<Manual>

```
package script;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Manual extends JDialog implements WindowListener{
    //메뉴얼 이미지 초기화
    ImageIcon ManuallImage[] = new ImageIcon[6];

    //버튼 이미지
    ImageIcon StartBtn[] = new ImageIcon[3];
    ImageIcon ReturnBtn[] = new ImageIcon[3];

    //메뉴얼 구성목록
    JLabel imageLabel;
    JButton okButton;

    int getItemNum = 0;    //엔딩 변경을 위한 현재 코인값
    boolean isfirst = true;    //처음 클리어 이후 엔딩 변경을 위한 값

    //스탑 보정 컨트롤러 값
    boolean keyUp = false;
    boolean keyDown = false;
    boolean keyLeft = false;
    boolean keyRight = false;
    boolean keyShift = false;
    boolean iscloseManual = false;

    int x, y;    //게임 창 중앙에 표시하기 위한 값
    int score;    //스코어

    public Manual(JFrame frame, String title, int x, int y) {
        super(frame,title, false);    //모달리스 다이얼로그

        for(int i = 0; i < 6; i++) {    //엔딩카드
            ManuallImage[i] = new ImageIcon("Image/Card/Manual_Card_"+i+".png");
        }

        this.imageLabel = new JLabel(ManuallImage[0]);    //메뉴얼 이미지 초기화
```

```

//스타트 버튼 이미지
StartBtn[0] = new ImageIcon("Image/Ui/UI Start Btn On.png");
StartBtn[1] = new ImageIcon("Image/Ui/UI Start Btn It.png");
StartBtn[2] = new ImageIcon("Image/Ui/UI Start Btn Down.png");

//리턴 버튼 이미지
ReturnBtn[0] = new ImageIcon("Image/Ui/UI Btn On.png");
ReturnBtn[1] = new ImageIcon("Image/Ui/UI Btn It.png");
ReturnBtn[2] = new ImageIcon("Image/Ui/UI Btn Down.png");

//설정값
this.x = x;
this.y = y;
setResizable(false);
this.setLocation(x + 100, y + 100); //게임창의 위치 x, y 를 받아 중앙에 표시
setLayout(new FlowLayout());
setSize(400, 600);
this.addWindowListener(this); //창을 x로 꺾을 때를 위한 보정

//버튼 이미지 설정값
this.okButton = new JButton(StartBtn[0]);
this.okButton.setRolloverIcon(StartBtn[1]);
this.okButton.setPressedIcon(StartBtn[2]);
//버튼 이미지만 표시
this.okButton.setBorderPainted(false);
this.okButton.setFocusPainted(false);
this.okButton.setContentAreaFilled(false);

//창에 추가
add(this.imageLabel);
add(this.okButton);

//버튼 액션 리스너
this.okButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        iscloseManual = true; //다이얼 로그를 꺼도 게임구동이 안되는 버그를 막기위한 보정 값
        ClickSound();
        setVisible(false);
    }
});

```


//키를 누르고 스탑을 하고 게임창으로 돌아갈때 키값을 다시 주고 받기 위한 함수

```
this.okButton.addKeyListener(new KeyAdapter() {  
    public void keyPressed(KeyEvent e) {  
        if(KeyEvent.VK_ESCAPE == e.getKeyCode()) {  
            iscloseManual = true;  
            ClickSound();  
            setVisible(false);  
        }  
        if(KeyEvent.VK_UP == e.getKeyCode()) {  
            keyUp = true;  
        }  
        if(KeyEvent.VK_DOWN == e.getKeyCode()) {  
            keyDown = true;  
        }  
        if(KeyEvent.VK_LEFT == e.getKeyCode()) {  
            keyLeft = true;  
        }  
        if(KeyEvent.VK_RIGHT == e.getKeyCode()) {  
            keyRight = true;  
        }  
        if(KeyEvent.VK_SHIFT == e.getKeyCode()) {  
            keyShift = true;  
        }  
    }  
    public void keyReleased(KeyEvent e) {  
        if(KeyEvent.VK_UP == e.getKeyCode()) {  
            keyUp = false;  
        }  
        if(KeyEvent.VK_DOWN == e.getKeyCode()) {  
            keyDown = false;  
        }  
        if(KeyEvent.VK_LEFT == e.getKeyCode()) {  
            keyLeft = false;  
        }  
        if(KeyEvent.VK_RIGHT == e.getKeyCode()) {  
            keyRight = false;  
        }  
        if(KeyEvent.VK_SHIFT == e.getKeyCode()) {  
            keyShift = false;  
        }  
    }  
}
```

```

        });
    }
//게임클리어 UI변경
public void GameClearUi() {
    if(score >= 300)
        this.imageLabel.setIcon(ManuallImage[5]);
    else if(isfirst) {
        this.imageLabel.setIcon(ManuallImage[1]);
        this.isfirst = false;
    }
    else
        this.imageLabel.setIcon(ManuallImage[3]);
    this.okButton.setIcon(ReturnBtn[0]);
    this.okButton.setRolloverIcon(ReturnBtn[1]);
    this.okButton.setPressedIcon(ReturnBtn[2]);
}
//게임오버 UI변경
public void GameOverUi() {
    if(getItemNum < 8)
        this.imageLabel.setIcon(ManuallImage[2]);
    else
        this.imageLabel.setIcon(ManuallImage[4]);

    this.okButton.setIcon(ReturnBtn[0]);
    this.okButton.setRolloverIcon(ReturnBtn[1]);
    this.okButton.setPressedIcon(ReturnBtn[2]);
}
//게임스탑 UI변경
public void GameStopUi() {
    this.imageLabel.setIcon(ManuallImage[0]);

    this.okButton.setIcon(StartBtn[0]);
    this.okButton.setRolloverIcon(StartBtn[1]);
    this.okButton.setPressedIcon(StartBtn[2]);
}

public void ClickSound() {
    Audio clickSound = new Audio("clickSound");
    Thread enemySpawnTh = new Thread(clickSound);
    enemySpawnTh.start();
}

```

```
//X로 창을 때 이벤트
@Override
public void windowOpened(WindowEvent e) {}
@Override
public void windowClosing(WindowEvent e) {
    iscloseManual = true;
}
@Override
public void windowClosed(WindowEvent e) {}
@Override
public void windowIconified(WindowEvent e) {}
@Override
public void windowDeiconified(WindowEvent e) {}
@Override
public void windowActivated(WindowEvent e) {}
@Override
public void windowDeactivated(WindowEvent e) {}
}
```

<GameManager>

```
package script;
```

```
import javax.sound.sampled.AudioSystem;
```

```
import javax.swing.*;
```

```
import java.awt.*;
```

```
public class GameManager extends JFrame {
```

```
    Player player; //플레이어
```

```
    PlayerController playerController; //플레이어 컨트롤러
```

```
    Item item; //아이템(동전)
```

```
    EnemySpawn enemySpawn; //적(총알)
```

```
    Manual manual; //메뉴얼(다이얼로그)
```

```
    Audio backGroundMusic; //오디오
```

```
    int Score = 0; //현재스코어
```

```
    int isDie = 1; //UI교체 후 게임오버가 되게 만들어주는 보정값
```

```
    int isClear = 1; //UI교체 후 게임클리어가 되게 만들어주는 보정값
```

```
    Toolkit imageTool = Toolkit.getDefaultToolkit(); //이미지킷
```

```
    Image lifeImage[] = new Image[2]; //현재 체력UI
```

```
    Image coinImage; //현재 모
```

```
은 동전UI
```

```
    Image coinBackGroundImage; //코인 배경 이미지
```

```
(틀)
```

```
    int GetItemnum = 0; //현재 모
```

```
은 동전 개수
```

```
    //이미지 버퍼를 위한 세팅
```

```
    Image buffImg;
```

```
    Graphics buffG;
```

```
    ImageIcon BackGroundImage; //배경이미지(맵)
```

```
    public GameManager(Player player, PlayerController playerController, Item item, EnemySpawn enemySpawn) {
```

```
        //설정값
```

```
        setTitle("DCU의 총알피하기");
```

```
        setSize(600,830);
```

```

setResizable(false);
setLocationRelativeTo(null);
setDefaultCloseOperation(EXIT_ON_CLOSE);

//초기화
this.BackgroundImage = new ImageIcon("Image/Map/BackGround.png");
this.player = player;
this.playerController = playerController;
this.item = item;
this.enemySpawn = enemySpawn;

//UI세팅
lifeImage[0] = imageTool.getImage("Image/Ui/DcuIcon.png");
lifeImage[1] = imageTool.getImage("Image/Ui/DcuIcon_Die.png");
coinImage = imageTool.getImage("Image/Ui/CoinIcon.png");
coinBackGroundImage = imageTool.getImage("Image/Ui/CoinIconBackGround.png");

backgroundMusic = new Audio("BGMSound"); //BGM
manual = new Manual(this, "test", this.getLocation().x, this.getLocation().y); //메뉴얼(다이얼로그)

setVisible(true);

//게임 배경음악
Thread backMusicTh = new Thread(backgroundMusic);
backMusicTh.start();

//게임 시작 전 설명
manual.setVisible(true);
SetControllerToManul();

//플레이어 컨트롤러 쓰레드
addKeyListener(this.playerController);
Thread Move = new Thread(this.playerController);
Move.setDaemon(true);
Move.start();

//아이템(동전) 컨트롤러 쓰레드
Thread ItemTh = new Thread(this.item);
ItemTh.setDaemon(true);
ItemTh.start();

```

```

//적(총알) 컨트롤러 쓰레드
Thread EnemyMove = new Thread(this.enemySpawn);
EnemyMove.setDaemon(true);
EnemyMove.start();
}
//BGM연속재생
public void continuousPlay() {
    if(!backGroundMusic.clip.isRunning()) {
        try {
            backGroundMusic.stream = AudioSystem.getAudioInputStream(backGroundMusic.file);
            backGroundMusic.clip = AudioSystem.getClip();
            backGroundMusic.clip.open(backGroundMusic.stream);
            backGroundMusic.clip.start();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
//모든 움직임 통제
public void AllStop() {
    this.playerController.isStop = true;
    this.enemySpawn.isStop = true;
    this.item.isStop = true;
}
//모든 움직임 통제 해제
public void AllClear() {
    this.playerController.isStop = false;
    this.enemySpawn.isStop = false;
    this.item.isStop = false;
}
//게임클리어
public void GameClear() {
    ClearSound();
    AllStop();
    manual.score = this.Score;
    manual.GameClearUi();
    manual.setVisible(true);
    manual.iscloseManual = true;
    GetItemnum = 0;
}

```

```

//게임클리어 사운드
public void ClearSound() {
    Audio clearSound = new Audio("GameClearSound");
    Thread enemySpawnTh = new Thread(clearSound);
    enemySpawnTh.start();
}

//게임오버
public void GameOver() {
    GameOverSound();
    AllStop();
    manual.getItemNum = 0;
    manual.getItemNum = GetItemnum;
    manual.GameOverUi();
    manual.setVisible(true);
    manual.iscloseManual = true;
}

//게임오버 사운드
public void GameOverSound() {
    Audio clearSound = new Audio("GameOverSound");
    Thread enemySpawnTh = new Thread(clearSound);
    enemySpawnTh.start();
}

//게임스탑(게임중 ESC를 눌렀을 때)
public void GameStop() {
    AllStop();
    playerController.keyEsc = false;
    SetManualToController();
    manual.GameStopUi();
    manual.setVisible(true);
    manual.iscloseManual = true;
}

//게임스탑 중 다이얼로그 호출 할 때 키값 변경을 위한 함수
public void SetManualToController() {
    manual.keyUp = playerController.keyUp;
    manual.keyDown = playerController.keyDown;
    manual.keyLeft = playerController.keyLeft;
    manual.keyRight = playerController.keyRight;
}

//게임스탑 중 다이얼로그 호출 할 때 키값 변경을 위한 함수
public void SetControllerToManul() {
    playerController.keyUp = manual.keyUp;

```

```

        playerController.keyDown = manual.keyDown;
        playerController.keyLeft = manual.keyLeft;
        playerController.keyRight = manual.keyRight;
    }
    //게임 오브젝트 리셋
    public void ReGame() {
        player.playerReset();
        playerController.AllKeyFalse();
        enemySpawn.AllEnemyReset();
        enemySpawn.enemys.setSize(10);
        enemySpawn.addEnemyCount = 0;
        enemySpawn.enemysScore = 0;
        item.itemScore = 0;
        item.Respawn();
        isDie = 1;
        isClear = 1;
        GetItemnum = 0;
    }
    //게임 중 설정
    public void GameSettting() {
        this.Score = enemySpawn.enemysScore + item.itemScore;
        if(playerController.keyEsc) {
            GameStop();
        }
        if(player.Health == 0) {
            if(isDie == 0) { //
                GameOver();
                isDie = 2;
            }
            else if(isDie == 1)
                isDie = 0 ;
        }
        if(player.GetItemNum == 10) {
            if(isClear == 0) { //
                GameClear();
                isClear = 2;
            }
            else if(isClear == 1)
                isClear = 0;
        }
        if(manual.iscloseManual && !manual.isVisible()) {

```



```

        manual.iscloseManual = false;
        SetControllerToManul();
        AllClear();
        if(player.Health == 0 || player.GetItemNum == 10) {
            ReGame();
        }
    }
    continuousPlay();
}

//그래픽 더블 버퍼링
@Override
public void paint(Graphics g) {
    buffImg = createImage(getWidth(),getHeight());
    buffG = buffImg.getGraphics();
    update(g);
}

@Override
public void update(Graphics g) {
    buffG.clearRect(0, 0, 600, 800);
    buffG.drawImage(BackGroundImage.getImage(), 0, 30, null);
    buffG.drawImage(this.player.playerImage,(this.player.xpos - 27),(this.player.ypos - 32), this);
    //buffG.drawString("●", (this.player.xpos), (this.player.ypos));    //실제플레이어포인트
    //buffG.drawImage(this.testImage.Image,this.testImage.xpos,this.testImage.ypos, this);
    GameSetttting();           //업데이트에서 설정이 돌아가게 놓았음
    DrawItem(g);               //동전 이미지
    DrawEnemy(g);              //총알 이미지
    DrawUi(g);                 //UI 이미지
    g.drawImage(buffImg,0,0,this);
    repaint();
}

//Item 그래픽
public void DrawItem(Graphics g) {
    buffG.drawImage(item.image, (item.xpos - 4), (item.ypos - 12), this);
}

//Enemy 그래픽
public void DrawEnemy(Graphics g) {
    for(int i = 0; i < this.enemySpawn.enemys.size(); i++) {
        Enemy enemy = this.enemySpawn.enemys.get(i);
        buffG.drawImage(enemy.image, (enemy.xpos - 4), (enemy.ypos - 12), this);
        //buffG.drawString("●", (enemy.xpos), (enemy.ypos));    //실제총알포인트
    }
}

```

```

    }
}
//UI 그래픽
public void DrawUi(Graphics g) {
    for(int i = 0; i < 5; i++) { //플레이어 체력 마다 UI변경
        if(player.Health <= i)
            buffG.drawImage(this.lifeImage[1], 15 + (i*40), 35, this);
        else
            buffG.drawImage(this.lifeImage[0], 15 + (i*40), 35, this);
    }
    buffG.drawString("Score : " + this.Score, 280, 60); //스코어 표시
    buffG.drawImage(this.coinBackGroundImage, 380 , 40, this); //코인 틀 표시
    this.GetItemnum = player.GetItemNum; //현재 모은 코인

    for(int i = 0; i < this.GetItemnum; i++) { //모은 코인에 따라 코인 표시
        buffG.drawImage(this.coinImage, 386+ (i * 20), 46, this);
    }
}
}

```

<Main>

```

package script;

public class Main {
    public static void main(String[] args){ //게임매니저에게매개변수로 넘겨주기위해생성
        Player player = new Player();
        PlayerController playerController = new PlayerController(player);
        EnemySpawn enemySpawn = new EnemySpawn(player);
        Item item = new Item(player);
        GameManager gameManager = new GameManager(player, playerController, item, enemySpawn);
    }
}

```