설계패턴 Term Project 보고서

Team5

20216064 양시훈

20202203 박호근

20200278 유윤재

목차

1. 프로젝트 내용 요약

- 위 프로젝트는 lucasvigier가 만든 Java 팩맨 코드에 설계패턴 및 객체지향 설계 개념을 활용하여 게임의 기능을 확장하고 설계를 개선하는 것이 목표이다. 우리 팀은 설계 개선보다는 게임의 기능 확장을 중심으로 개발하였으며, 여러 설계패턴과 객체지향 개념을 고려하여 구조를 설계하였다. 추가한 기능에 대한 간단한 요약은 아래와 같다.
- 기능 확장
    1. 사운드 추가 : Singleton 패턴, SRP
    2. 라이프 추가 : Observer 패턴
    3. 사망 시 게임 재개 기능 : Observer 패턴
    4. GameOver, GameClear 기능 : State 패턴
    5. 아이템 추가, 팩맨에 특수 효과 적용 : Factory, Command 패턴, OCP
    6. 레벨 별 난이도 설정 (유령 수, 아이템 스폰 확률 기반) : State, Factory 패턴

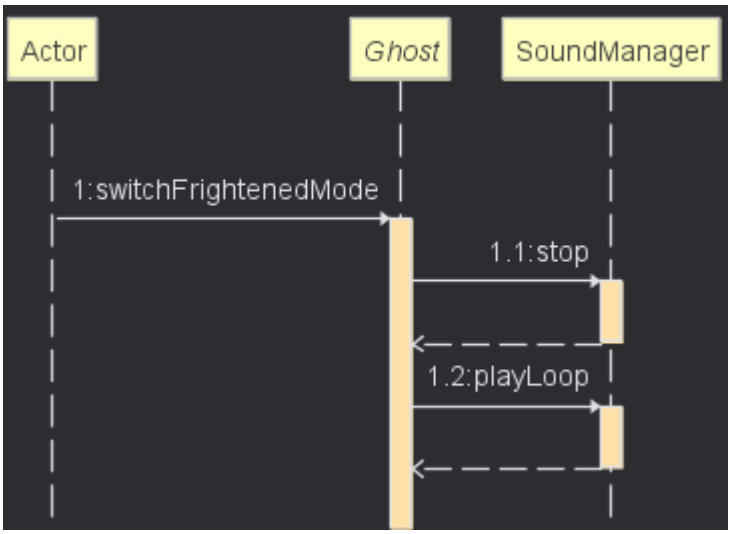
2. 확장한 기능, 설계 개선 내용

1. **사운드 추가**

a. 기능 요약 : resources 폴더에 담긴 wav파일을 load하여 사운드의 재생/중지 관리를 하는 매니저 클래스이다. 사용할 사운드의 종류를 enum으로 정의하고, enum-clip을 key-value로 가지는 Map을 멤버로 가진다. 그리고 재생 및 중지 함수는 매개변수로 실행할 사운드 enum을 받아서 Map에서 실제 사운드 Clip을 찾은 뒤에 조작하는 구조이다. 한 번 실행하는 play함수, 반복 실행하는 playLoop함수, 중지하는 stop함수 등의 API가 있다.

b. 설계 패턴/원칙 : Singleton 패턴, SRP

c. 설계 이유 : 기존 팩맨 게임에 모든 요소에 원작과 동일한 사운드를 적용하기 위해 SoundManager클래스를 Singleton패턴을 활용하여 구현하였다. Singleton패턴을 사용한 이유는 사운드 특성상 프로젝트의 매우 넓은 영역에 광범위하게 사용될 가능성이 높기 때문에, 사운드 적용이 필요한 모든 클래스가 Sound로직을 각자 가지고 있는 것보다 하나의 단일 매니저 객체가 Sound에 대한 단일 책임을 가지고 있고 이를 static하게 호출하는 편이 더욱 깔끔하고 직관적으로 동작할 것이라고 생각했기 때문이다.




```
public class SoundManager {
    public enum Sound {
        START,
        SUCCESS,
        FAIL,
        PAC_EAT,
        PAC_FRUIT,
        PAC_DOT,
        PAC_EXTRA,
        GHOST_NORMAL,
```

```
        GHOST_EATEN,
        GHOST_FRIGHTENED,
    }

    private final static SoundManager instance = new SoundManager
();

    private final Map<Sound, Clip> soundMap = new HashMap<Sou
nd, Clip>();

    private SoundManager() {
        loadAllSoundFiles();
    }

    public static SoundManager getInstance() { return instance; }

    //특정 사운드 재생
    public void play(Sound sound) {
        stop(sound);
        soundMap.get(sound).start();
    }

    //특정 사운드 반복 재생
    public void playLoop(Sound sound) {
        soundMap.get(sound).loop(20);
    }

    //특정 사운드 정지하고 시작지점으로 되돌림
    public void stop(Sound sound) {
        Clip clip = soundMap.get(sound);
        clip.stop();
        clip.setFramePosition(0);
    }
    ...
}
```

d. 추가 설명 : 팩맨 원작에서는 전체 유령의 상태에 따라 단일 사운드가 재생되는 구조이기 때문에, 이를 최대한 비슷하게 구현하기 위해서 Ghost클래스 내부에

frightened 상태인 유령의 수와 eaten 상태인 유령의 수를 저장하는 static int 변수를 두고, 이를 유령의 state switch 함수에서 증가/감소시키도록 하여 여러가지 상황에 맞는 사운드가 재생되도록 작성하였다.

- 처음에는 전체 유령의 상태 별 개수를 관리하는 클래스를 따로 만들어서 이를 Ghost클래스가 참조하는 방식으로 구현하려고 했지만, 그렇게 설계하게 되면 SRP는 만족할 수 있겠지만 Ghost에 대한 새로운 클래스 의존관계가 생기고, 단순히 Ghost 내부에 static counter를 두는 것으로 해결 가능한 문제를 과도하게 해결한다고 판단했다. 그리고 모든 Ghost 객체들에 대한 counter라는 특성상 Ghost 클래스의 static 변수로 두는 것이 더욱 직관적이라고 판단하였다. 즉, 불필요한 의존관계 생성의 방지와 단순함을 위해 static int로 해당 기능을 작성했다. 참고로 새로운 클래스를 추가하는 선택지 대신 기존의 Game 클래스에서 위의 기능을 관리하는 방법도 있겠으나, 이렇게 되면 필연적으로 ghost가 상위 모듈인 Game에 대해 참조를 가져야하는 dependency rot이 발생하거나 Observer패턴을 통한 과도한 문제 해결이 필요할 것이라 판단하여 해당 방법은 선택하지 않았다.

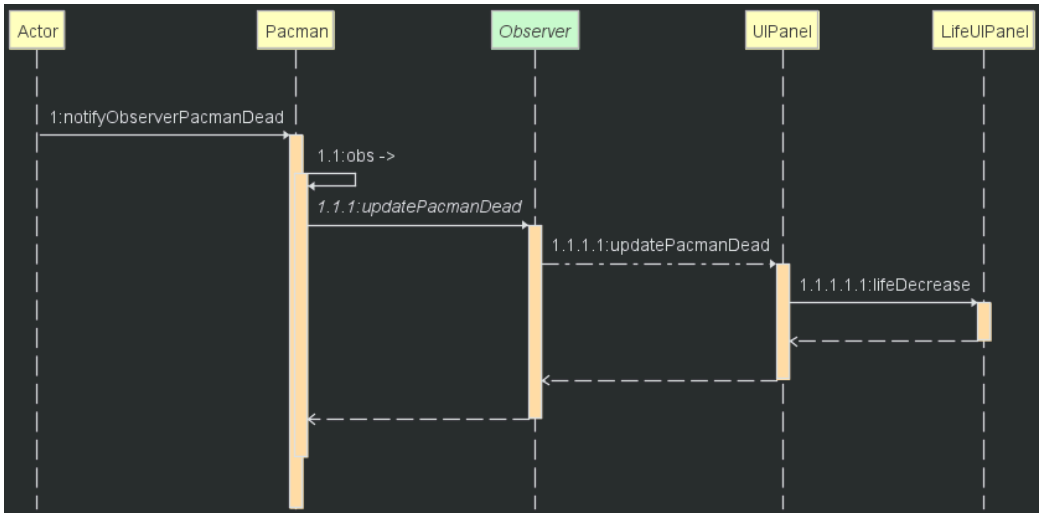
```
public abstract class Ghost extends MovingEntity {
        ...
        private static int frightenedCnt;//공포 상태의 유령 개수 (먹힌상태 유령포함)
    private static int eatenCnt; //먹힌 상태의 유령 개수
       ...

    public void switchFrightenedMode() {
        frightenedCnt++;
        frightenedTimer = 0;
        SoundManager.getInstance().stop(SoundManager.Sound.GHOST_NORMAL);
        SoundManager.getInstance().playLoop(SoundManager.Sound.GHOST_FRIGHTENED);
        state = frightenedMode;
    }

    public void switchEatenMode() {
        eatenCnt++;
        SoundManager.getInstance().stop(SoundManager.Sound.GHOST_FRIGHTENED);
        SoundManager.getInstance().playLoop(SoundManager.Sound.
```

```
GHOST_EATEN);
        state = eatenMode;
    }

    public void switchHouseMode() {
        state = houseMode;

        if(eatenCnt == 0) //맨 처음 시작할 때 stay→house로 인한 함수 호출인 경우, 뒷 내용 필요 없으므로 바로 리턴
            return;

        eatenCnt--;
        if (eatenCnt == 0) { //다른 먹힌 상태의 유령이 없는 경우에만 다른 사운드로 교체한다
            SoundManager.getInstance().stop(SoundManager.Sound.GHOST_EATEN);
            if (frightenedCnt - 1 == 0) {//공포 상태 유령이 없으면 (본인에 대한 frightenedCnt는 밑에 있는 함수에서 처리하므로 여기에서는 -1을 해줘야 함) 일반 사운드를 재생한다
                SoundManager.getInstance().playLoop(SoundManager.Sound.GHOST_NORMAL);
            } else { //공포 상태 유령이 존재하는 경우 공포 사운드를 재생한다
                SoundManager.getInstance().playLoop(SoundManager.Sound.GHOST_FRIGHTENED);
            }
        }
    }

    public void switchChaseModeOrScatterMode() {
        if (isChasing) {
            switchChaseMode();
        }else{
            switchScatterMode();
        }

        if(frightenedCnt == 0) //처음 시작할 때인 경우 아래의 코드를 실행할 필요X
            return;
```

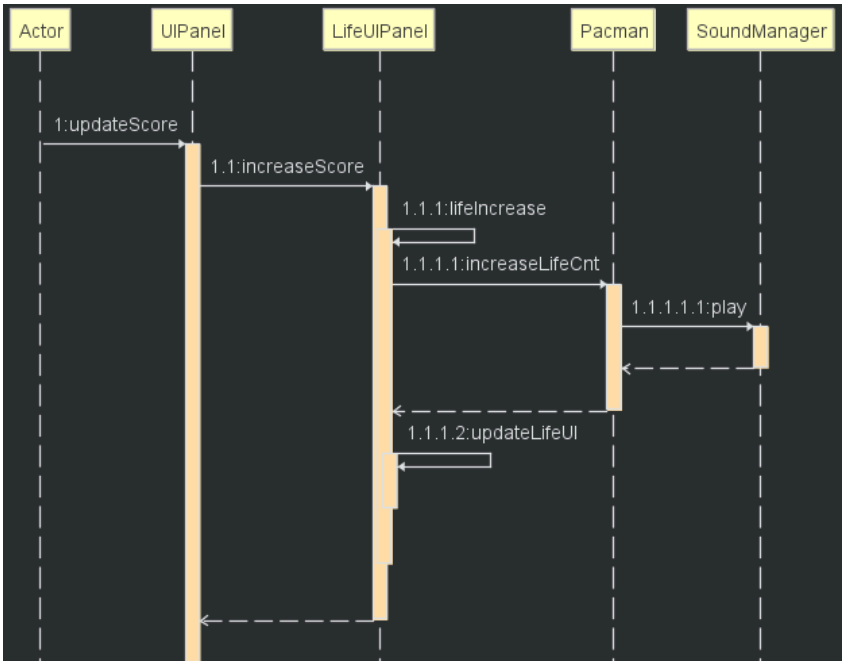
```
        frightenedCnt--;
        if (frightenedCnt == 0) { //공포 상태의 유령이 없으므로 일반 사운드
재생
            SoundManager.getInstance().stop(SoundManager.Sound.GH
OST_FRIGHTENED);
            SoundManager.getInstance().playLoop(SoundManager.Soun
d.GHOST_NORMAL);
        }
    }

    ...
}
```

2. **라이프 기능**
    a. 기능 요약 : 팩맨이 라이프를 가지도록 하여 유령에게 닿더라도 GameOver되지 않고, 보유한 라이프를 소모하여 부활하여 재개할 수 있도록 한다. 팩맨이 유령에게 닿으면 라이프가 하나 차감되며 팩맨이 일정 점수(5000점 단위) 이상 획득할 경우, 라이프가 하나 증가한다. 라이프가 0이 되는 경우 GameOver 창으로 넘어가며 게임을 완전히 새로 시작하게 된다. 또한 현재 보유하고 있는 라이프의 개수만큼 라이프UI가 게임 하단에 표시된다.
    b. 사용된 설계 패턴 : Observer 패턴
    c. 설계 이유
        - LifeUIPanel은 기존의 UIPanel 클래스 내부에서 생성하여 수명 주기를 공유하는 Composition을 사용하였다. 왜냐하면 기존 UIPanel과 Pacman 사이의 Observer-Subject 관계를 활용하기 위해서, 그리고 UIPanel의 점수 증가 로직에 LIfeUIPanel이 직접적인 관련이 있기 때문이다.
        - 우선 실질적인 라이프의 개수를 의미하는 lifeCnt 변수는 의미상 직관적인 Pacman 클래스 내부에 멤버변수로 존재하고, lifeCnt를 다루는 함수를 제공한다. 이때 Pacman의 lifeCnt의 변화가 라이프UI에 반영되어야 하므로 이 둘 사이에 의존관계가 필요한데, 인게임 모듈과 UI 모듈 간의 직접적인 의존관계를 완전히 제거하여 low coupling, high cohesion을 유지하고 싶었기 때문에 기존 UIPanel의 Observer 인터페이스의 updatePacmanDead함수를 통

해 Pacman이 죽었을 때 LifeUIPanel의 **라이프 감소** 함수를 실행하도록 하였다.



- 두번째로는 라이프 증가의 조건이 팩맨이 일정 단위 점수를 획득하는 것이므로 직접적인 포함관계(Compsition)이 적절하다고 생각했기 때문이다. 팩맨이 얻은 점수는 UIPanel 클래스에서 관리하는데, UIPanel의 updateScore 함수 내부에 lifeUIPanel의 increaseScore를 호출하여 lifeUIPanel 쪽에서 얻은 점수만큼 tempScore에 누적하고, tempScore가 단위점수를 넘겼을 경우에 **라이프를 증가**시키도록 하였다.



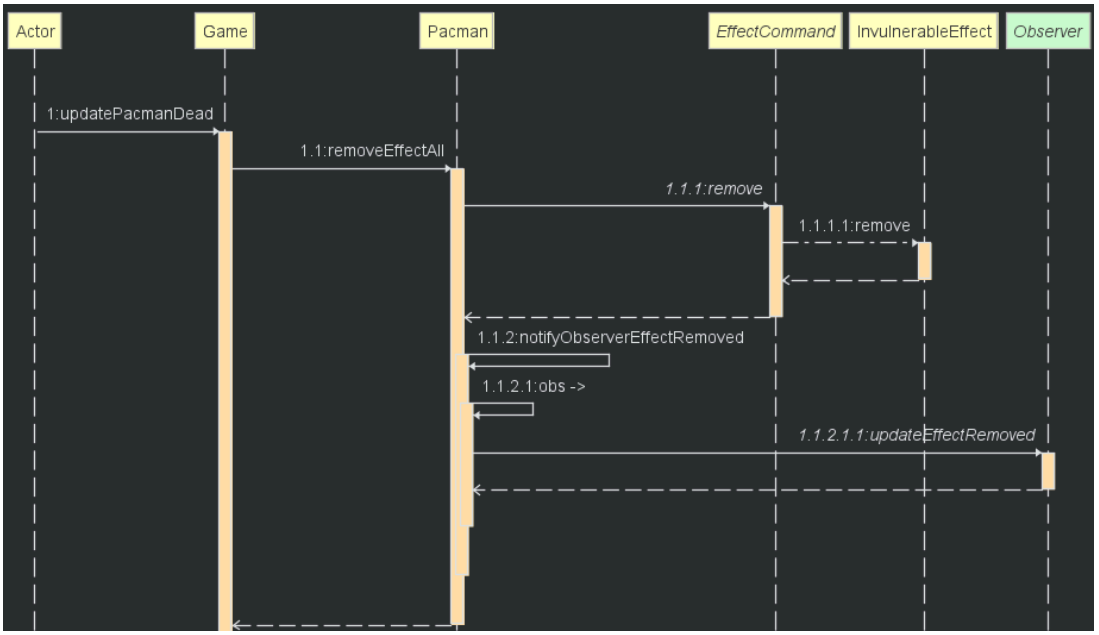

```
public class UIPanel extends JPanel implements Observer {
    private LifeUIPanel lifePanel;//Composition
```

```
        ...
        public void updateScore(int incrScore) {
        this.score += incrScore;
        this.lifePanel.increaseScore(incrScore);
        this.scoreLabel.setText("Score: " + score);
    }
    ...

    @Override
    public void updatePacmanDead() {
        lifePanel.lifeDecrease();
    }
    ...
}
```

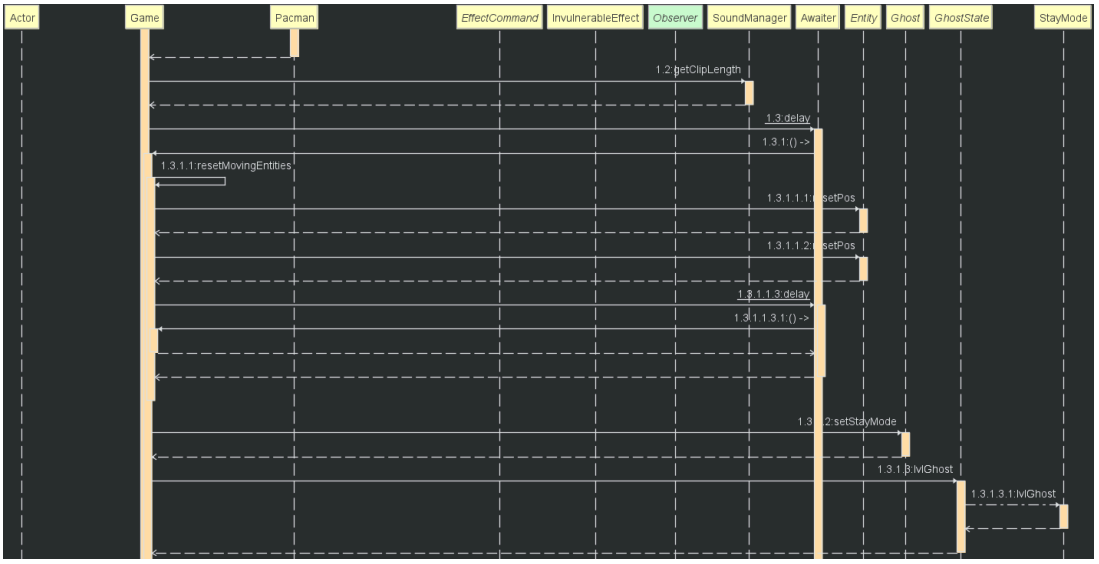
```
public class LifeUIPanel extends JPanel {
        private int tempScore; //획득한 점수 버퍼
    private final int unitScore; //라이프가 오르는 기준 점수값
    private Pacman pacman; //lifeCnt증감을 delegate하기 위한 참조
    ...

    //tempScore에 점수 누적하고 life로 변환 가능하면 변환
    public void increaseScore(int addScore) {
        tempScore += addScore;
        lifeIncrease();
    }

    //쌓인 점수로 라이프 증가 가능한만큼 증가
    private void lifeIncrease() {
        while(tempScore >= unitScore) {
            tempScore -= unitScore;
            pacman.increaseLifeCnt();
            updateLifeUI();
        }
    }

    //라이프 감소
    public void lifeDecrease() {
```

```
        pacman.decreaseLifeCnt();
        updateLifeUI();
    }
    ...
}
```

2. **사망 시 게임 재개 기능**
    a. 기능 요약 : 원래는 팩맨이 유령과 충돌했을 경우 기존 게임의 경우 바로 프로그램이 종료되는 형식이었지만, life 기능의 연장선으로 팩맨과 유령이 충돌했을 때 라이프를 차감하고 잠시 게임을 정지시키고 게임의 점수와 아이템 배치 상태만 보존하고 나머지 요소는 전부 초기화시켜 다시 게임을 이어서 플레이하도록 수정하였다. 라이프가 전부 차감되었을 경우 GameOver 상태로 전환하여 게임을 완전히 초기화하여 0레벨부터 다시 시작할 수 있도록 하였다.
    b. 사용된 설계 패턴 : Observer 패턴
    c. 설계 이유 : 팩맨이 죽으면 두 가지 일을 해야한다. 첫 번째는 Game의 일부를 다시 리셋하는 것이고, 두 번째는 라이프를 차감하고 라이프UI를 갱신하는 것이다. 만약에 Game 리셋만 필요했다면 이미 Game이 Pacman을 알고있으므로 굳이 Observer패턴을 안쓰고 직접 참조하는 방식으로 간단하게 로직을 작성했을 것이다. 그러나 라이프 UI를 갱신하는 과정은 현재 프로젝트 구조가 인게임 모듈과 UI 모듈의 철저히 분리하고 있기 때문에, 이러한 low coupling + high cohesion 구조의 일관성을 유지하기 위해 기존의 Observer-Subject 구조를 활용한 것이다.
    d. 구조 추가 설명 : 두 번째 기능인 라이프 차감 및 라이프UI 갱신은 이미 1번 : 라이프 기능에서 설명했으므로 생략하고, 첫 번째 기능인 Game의 일부를 리셋하는 기능에 대해서만 상세히 설명하겠다.
        - 게임 일시정지 : isPause를 true로 설정하여 Game의 update를 막는다.
        - 팩맨에게 적용 중이던 모든 아이템 효과 제거 : pacman의 removeEffectAll 함수를 통해 pacman에게 적용 중이던 모든 아이템 효과를 제거한다.



- 일정 시간 대기 후 Pacman과 Ghost 리셋: Pacman이 죽으면 Sound.FAIL 효과음이 SoundManager를 통해 실행되고, 직접 만든 Awaiter 비동기 유틸 클래스를 통해 해당 효과음이 끝날 때까지 대기 후 콜백으로 resetMovingEntities함수를 호출하고 Ghost 상태를 초기화한다.
    - resetMovingEntityes함수는 Entity 클래스의 resetPos함수를 호출하여 Pacman과 모든 Ghost들의 위치를 초기화 위치로 리셋한다.
    - 모든 Ghost들의 상태 또한 게임을 시작 단계로 다시 초기화시켜야하므로, 모든 ghost의 setStayMode함수를 호출해서 StayMode로 초기화하고 lvlGhost함수를 호출하여 현재 게임 level에 따른 ghost 활동 여부를 결정한다.
- 게임 재개 : resetMovingEntities함수에서 Awaiter함수로 2초 대기 후 isPause를 false로 설정하는 콜백을 실행하고, 다시 Game의 update함수가 정상 동작하면서 게임이 재개된다.




```
public class Game implements Observer {
       public void update() {
      if(isPause) //정지 상태이면 모든 update를 하지 못하도록 바로 리턴시킴
          return;
      ...
       }
       ...
       @Override
   public void updateGhostCollision(Ghost gh) {
       if (gh.getState() instanceof FrightenedMode) {
           SoundManager.getInstance().play(SoundManager.Sound.PAC_EAT);
           gh.getState().eaten(); //유령이 먹혔을 때 특별한 전환이 존재하면, 그 상태가 그에 맞게 변경된다
       }
       else if (!(gh.getState() instanceof EatenMode) && !pacman.getInvulnerable()) {
           //팩맨 사망
           pacman.die();//die함수 안에서 notifyObserverPacmanDead실행
       }
   }
```

```
        @Override
    public void updatePacmanDead() {
        isPause = true;
        pacman.removeEffectAll();

        float length = SoundManager.getInstance().getClipLength(SoundM
anager.Sound.FAIL);
        Awaiter.delay(length, ()-> {
            resetMovingEntities();
            for(Ghost g : ghosts) {
                g.setStayMode();
                g.getState().lvlGhost(level);
            }
        });

        if(pacman.isLifeZero()) {
            //완전히 게임오버
            gameState.die();
        }
    }
    ...

    private void resetMovingEntities() {
        pacman.resetPos();
        for(Ghost g : ghosts) {
            g.resetPos();
        }
        Awaiter.delay(2, ()->isPause = false);
    }
    ...
}
```

2. **GameOver, GameClear 기능**
    a. 기능 요약 : game을 생성할 때 pacgum을 생성하는데 이 때 pacgumcnt를 1씩 증가시켜 pacgum의 총 개수를 확인하고 이 후 pacman이 pacgum을 먹을 때 마다 pacgumcnt를 1씩 감소시켜 0이 되는 순간 GameClear 상태로 넘어간다. 이 때 미리 준비한 gameclear 화면을 렌더링해 프린트하고 입력을 감지한다. y를 입력하면 다음 레벨로, n을 입력하면 프로그램을 종료한다.

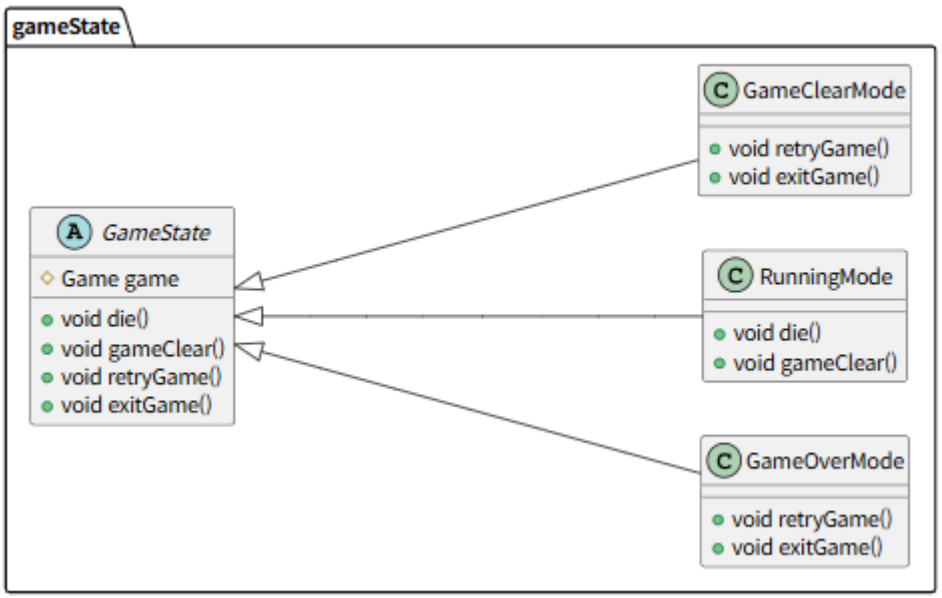
다음 레벨을 진행할 때 GamePlayPanel.java의 init method와 Game.java의 cleanup method를 사용해 기존 오브젝트를 초기화하고 다시 할당한다.

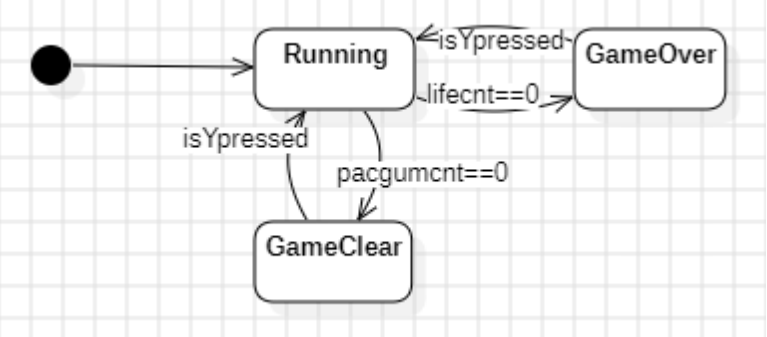
game 진행 중 pacman이 ghost와 충돌하여 life가 0이 되는 것을 감지해 gamestate를 gameover로 전환한다. 이 때 gameover 화면을 출력하고 gameclear 때와 비슷하지만 y를 입력할 경우 스코어와 레벨을 0으로 초기화해 처음부터 다시 도전하고, n을 입력하면 프로그램을 종료한다.

b. 적용된 설계 패턴 : State 패턴

c. 설계 이유

- 게임 진행 중인 Running과 GameClear, GameOver를 각각의 상태로 정의하고 GamePlayPanel에서의 Run 메소드에서 현재의 상태를 받아왔을 때 각각의 상태에 따라 다른 화면을 출력하도록 하였다.
    - state 패턴을 사용하여 세 가지 상태를 정의하였고 상태를 검사하는 형식으로 각각의 상태를 확인하여 해당하는 화면을 출력하는 완벽한 State 패턴의 형식을 갖추지는 못했지만 기존의 코드를 살리는 형식으로 코드를 수정하였다.





Game.java

```
...
public void update() {
      if(isPause) //정지 상태이면 모든 update를 하지 못하도록 바로 리턴시킴
         return;

      for (Entity o: objects) {
         if (!o.isDestroyed()) o.update();
      }
      if (this.pacGumCount <= 0)
      {
         // pacMan이 pacGum을 모두 먹었다면,
         isPause = true;
         SoundManager.getInstance().stopAllSound();
         SoundManager.getInstance().play(SoundManager.Sound.SUCCES
S);
         System.out.println("Level Clear!!");
         gameState.gameClear();
      }
   }
...
@Override
   public void updatePacmanDead() {
      isPause = true;
      pacman.removeEffectAll();

      float length = SoundManager.getInstance().getClipLength(SoundMana
ger.Sound.FAIL);
      Awaiter.delay(length, ()-> {
```

```
            resetMovingEntities();
            for(Ghost g : ghosts) {
                g.setStayMode();
                g.getState().lvlGhost(level);
            }
        });

        if(pacman.isLifeZero()) {
            //완전히 게임오버
            gameState.die();
        }
    }
...
public void cleanup() {
        objects.clear();
        ghosts.clear();
        walls.clear();

        pacman = null;
        blinky = null;

        pacGumCount = 0;
        firstInput = false;
        //level = 0;

        if (running != null) {
            gameState = running;
        }
    }
    public void delegatelvlGhost(int level){
        for(Ghost gh : ghosts){
            gh.getState().lvlGhost(level);
        }
    }
public void switchGameOver() {
        gameState=gameover;
    }
    public void switchGameClear(){
```

```
        gameState=gameclear;
    }
    public void switchRunning(){
        gameState=running;

public GameState getGameState() {return gameState;}
```

GamePlayPanel.java

```
...
public void cleanup(){
        if (g != null) {
            g.dispose();
        }
        img = null;
        g = null;
        key = null;
        if (game != null) {
            game.cleanup();
        }
        game = null;
    }
@Override
    public void run() {
        ...
        while (running) {
            ...
            if(game.getGameState() instanceof RunningMode){
                render();
            }
            else if(game.getGameState() instanceof GameOverMode){
                gameoverScreen();
                if (key.k_y.isPressed) {
                    level=0;
                    GameLauncher.getUIPanel().scoreReset();
                    GameLauncher.getUIPanel().updateScore(0);
                    cleanup();
```

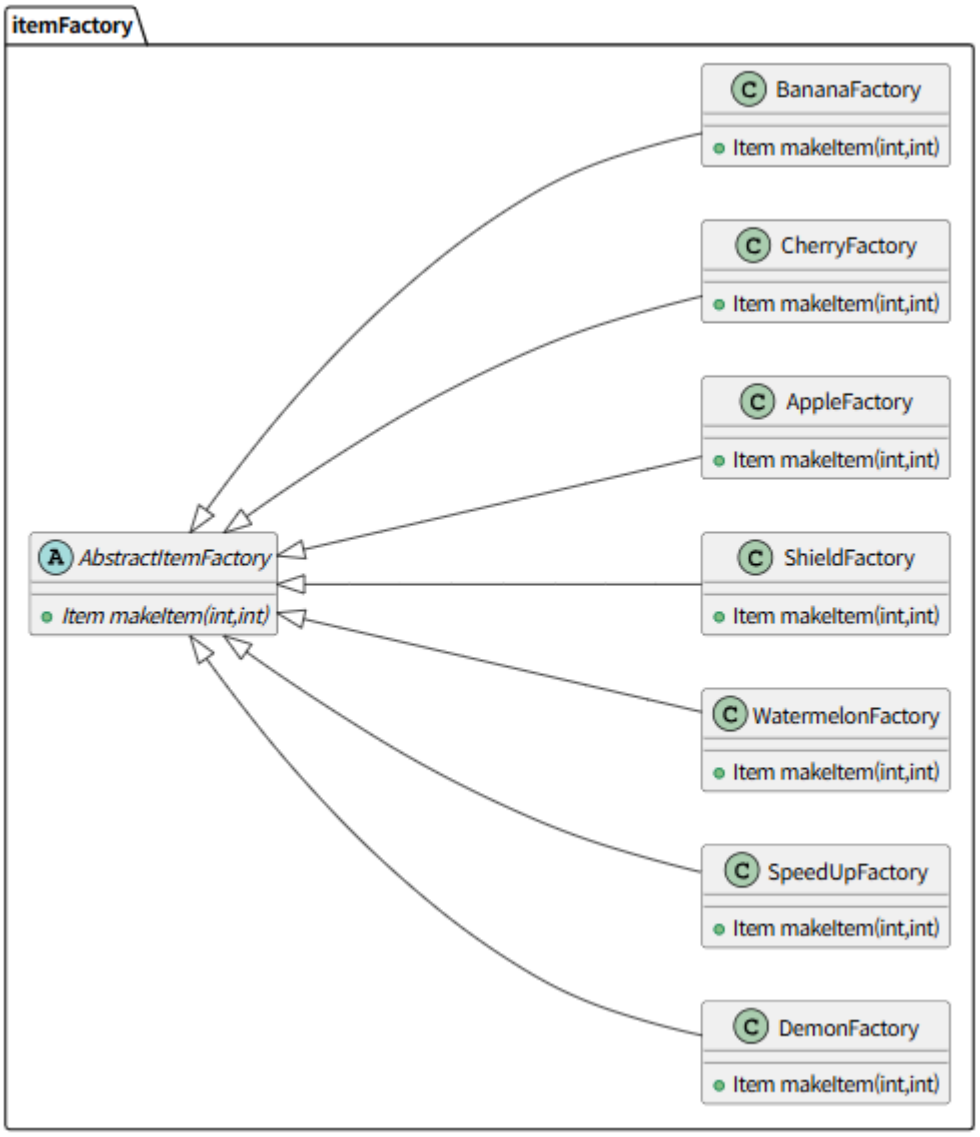
```
                init();
                game.getGameState().retryGame();
            }
            // 'N' 입력: 게임 종료 (Exit)
            else if (key.k_n.isPressed) {
                game.getGameState().exitGame();
            }
        }
        else{
            gameclearScreen();
            if (key.k_y.isPressed) {
                if(level<4){level++;}
                cleanup();
                init();
                game.getGameState().retryGame();
            }
            // 'N' 입력: 게임 종료 (Exit)
            else if (key.k_n.isPressed) {
                game.getGameState().exitGame();
            }
        }
        ...
    }
    }
...
```

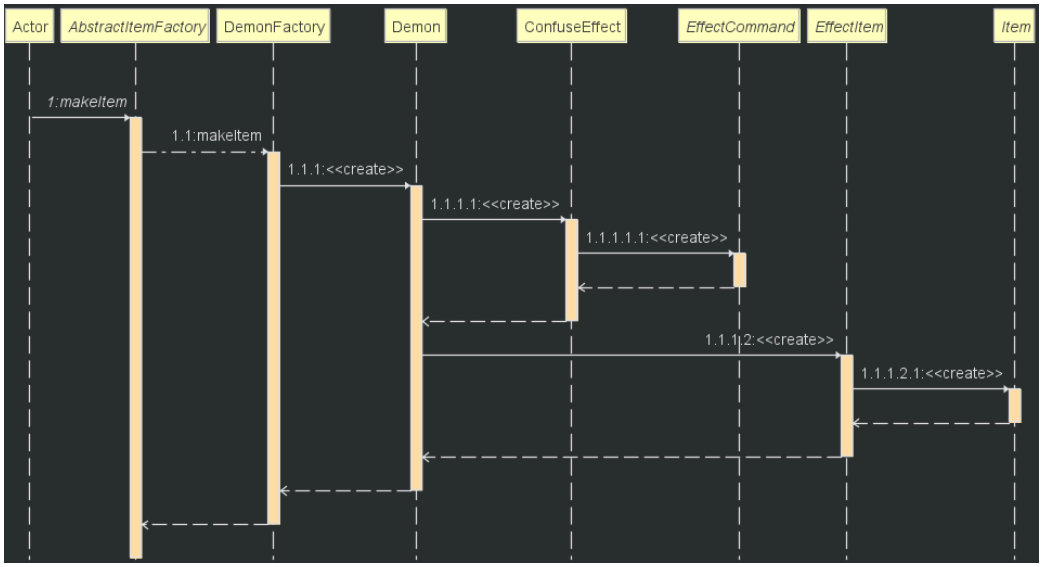
GameState.java

```
package game.gameState;

import game.Game;

public abstract class GameState {
    protected Game game;
    public GameState(Game game) {this.game = game;}

    public void die(){}
    public void gameClear(){}
```

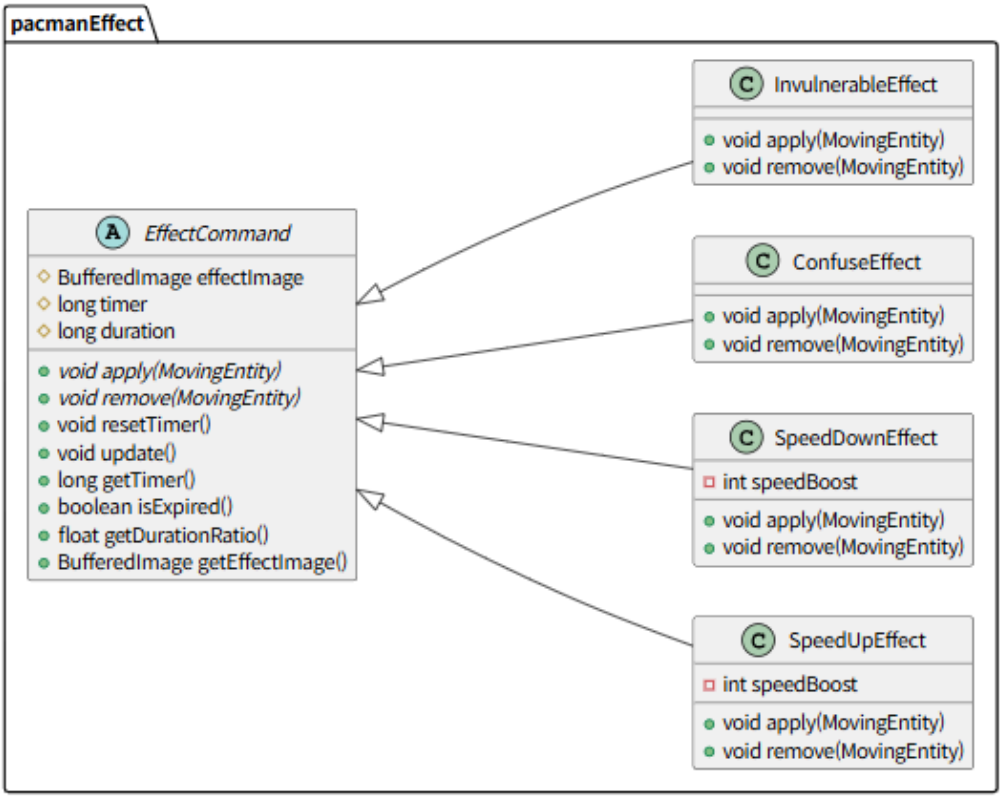
```
    public void retryGame(){}
    public void exitGame(){
}
```

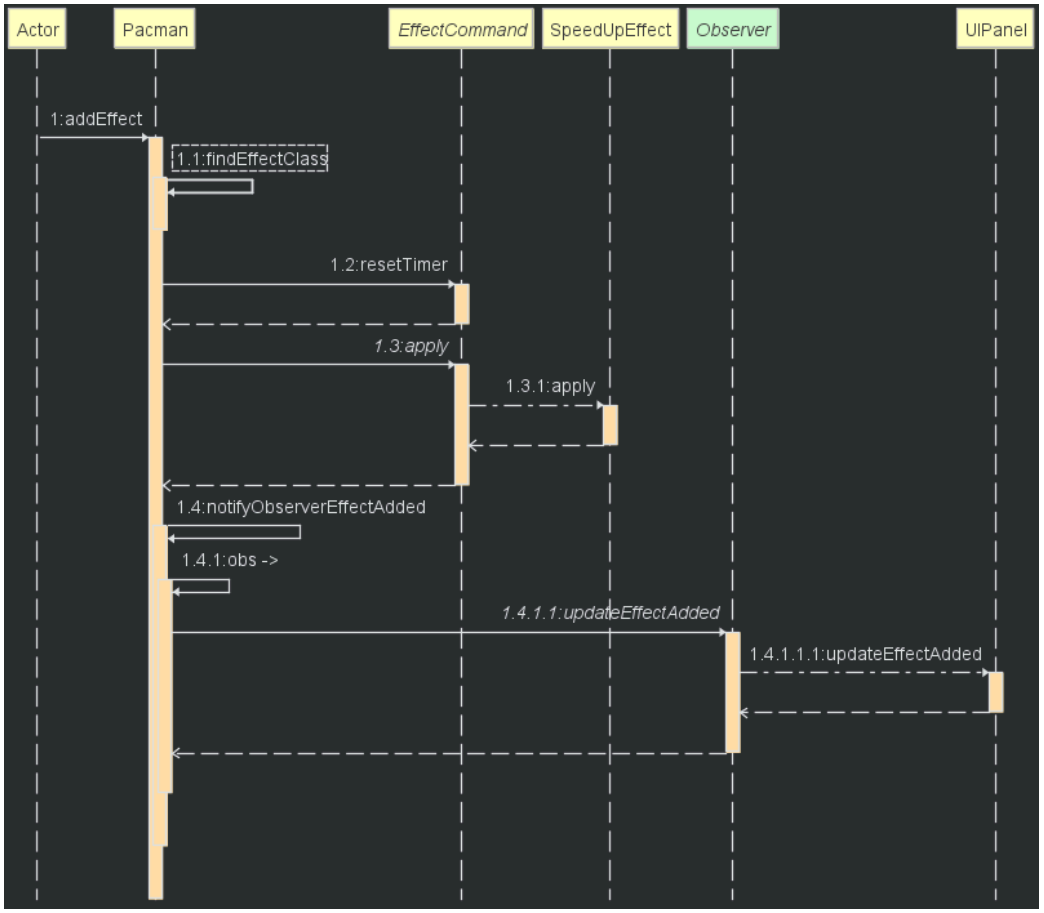
5. 아이템 추가, 팩맨에 특수 효과 적용
    a. 기능 요약 : pacgum이 생성되어야할 자리에 레벨에 비례한 확률로 추가점수인 과일 아이템과 버프, 디버프 아이템을 생성하고, 이것을 팩맨이 먹을 경우 팩맨에 버프, 디버프 효과가 적용된다.
    b. 적용된 설계 패턴/원칙 : Factory, Command 패턴, OCP
    c. 설계 이유
        - 아이템 오브젝트를 생성하기 위해서는 생성 패턴이 필요했고, 아이템의 종류가 여러가지였기 때문에 Factory 패턴으로 아이템 생성을 factory 클래스에 각각 delegate하였다.

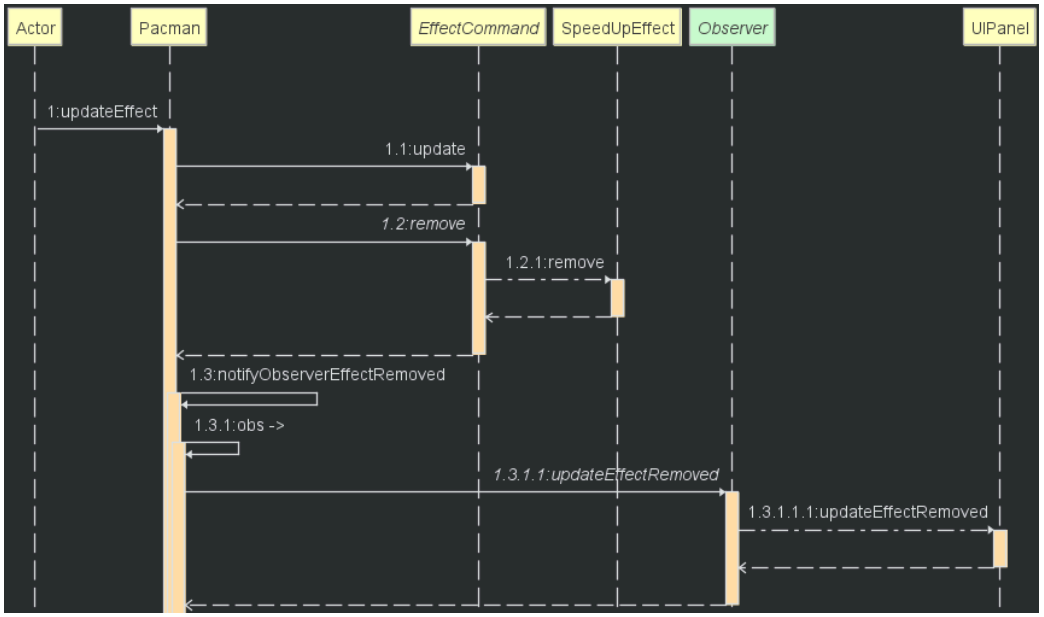

itemFactory
BananaFactory
Item makeItem(int,int)
CherryFactory
Item makeItem(int,int)
AppleFactory
Item makeItem(int,int)
AbstractItemFactory
Item makeItem(int,int)
ShieldFactory
Item makeItem(int,int)
WatermelonFactory
Item makeItem(int,int)
SpeedUpFactory
Item makeItem(int,int)
DemonFactory
Item makeItem(int,int)


Actor
AbstractItemFactory
DemonFactory
Demon
ConfuseEffect
EffectCommand
EffectItem
Item
1:makeItem
1.1:makeItem
1.1.1:<<create>>
1.1.1.1:<<create>>
1.1.1.1.1:<<create>>
1.1.1.2:<<create>>
1.1.1.2.1:<<create>>

- 게임 진행 과정에서 아이템을 적용 및 제거할 때 일일이 item이 어떤 하위 클래스인지 down casting해서 확인하는 구조로 작성하면 OCP를 위반하여 item이 추가될 때마다 새로운 조건문을 추가해야해서 번거로울 것이라고 판단했다. 따라서 Command 패턴을 활용하여 팩맨이 특수 아이템을 획득하면 추상화된 EffectCommand의 인터페이스를 통해서 해당 아이템의 효과를 Apply/Undo 하였다.







AbstractItemFactory.java

```
package game.itemFactory;

import game.entities.items.Item;

public abstract class AbstractItemFactory {
    public abstract Item makeItem(int xPos, int yPos);
}
```

Item.java

```
package game.entities.items;

import game.entities.StaticEntity;

import javax.imageio.ImageIO;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.IOException;

// static Entity를 상속한 item.. 근데 사진을 넣어야함.
public abstract class Item extends StaticEntity {
    protected BufferedImage sprite;
    protected int point = 0;

    public Item(int xPos, int yPos, String spriteName) {
        super(32, xPos, yPos);
        try {
            this.sprite = ImageIO.read(getClass().getClassLoader().getResource
("img/items/" + spriteName));
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }

    public Item(int xPos, int yPos, String spriteName, int point) {
        this(xPos, yPos, spriteName);
        this.point = point;
```

```
    }

    @Override
    public void render(Graphics2D g) {
        if (this.sprite != null) {
            // 문법: g.drawImage(이미지, x좌표, y좌표, 가로크기, 세로크기, 옵저버);
            g.drawImage(this.sprite, (int)this.xPos, (int)this.yPos, 32, 32, null);
        }
    }

    public void setPoint(int value) {
        this.point = value;
    }

    public int getPoint() {
        return this.point;
    }

    public BufferedImage getImage() { // just for test
        return this.sprite;
    }
}
```

EffectItem.java

```
package game.entities.items;

import game.pacmanEffect.EffectCommand;

public abstract class EffectItem extends Item {
    protected final EffectCommand effectCommand;

    public EffectItem(int xPos, int yPos, String spriteName, EffectCommand e
ffectCommand) {
        super(xPos, yPos, spriteName);
        if (effectCommand == null)
            throw new NullPointerException("수정 필요! EffectItem 생성 시 comm
and가 누락되었습니다!");
```

```
        this.effectCommand = effectCommand;
    }

    public EffectCommand getEffectCommand() {
        return effectCommand;
    }
}
```

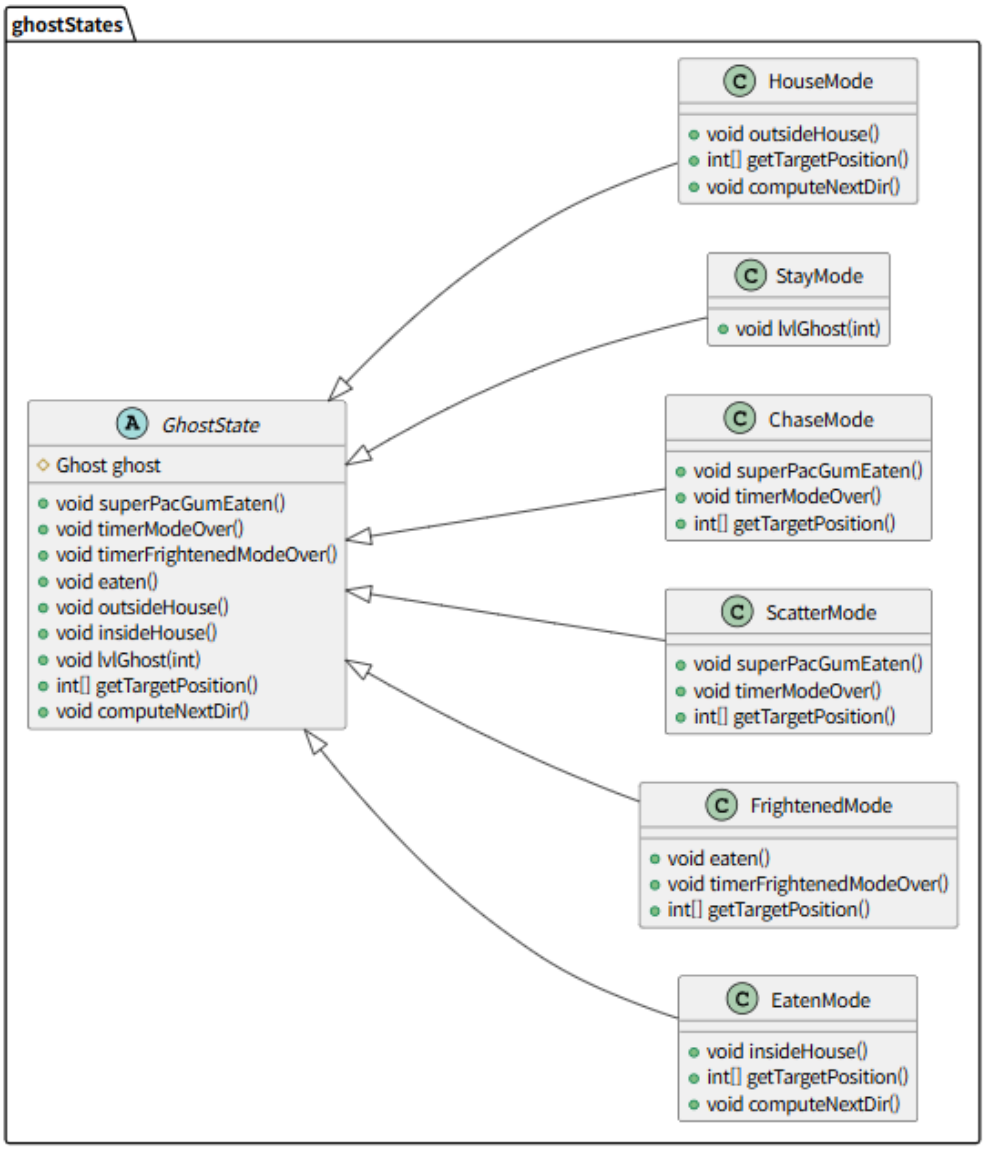
6. 레벨 별 난이도 설정
    a. 기능 요약 : GamePlayPanel.java의 level 변수에 따라 게임 시작 시 Stay mode에서 house mode로 전환되는 ghost의 개수를 점차 늘린다.

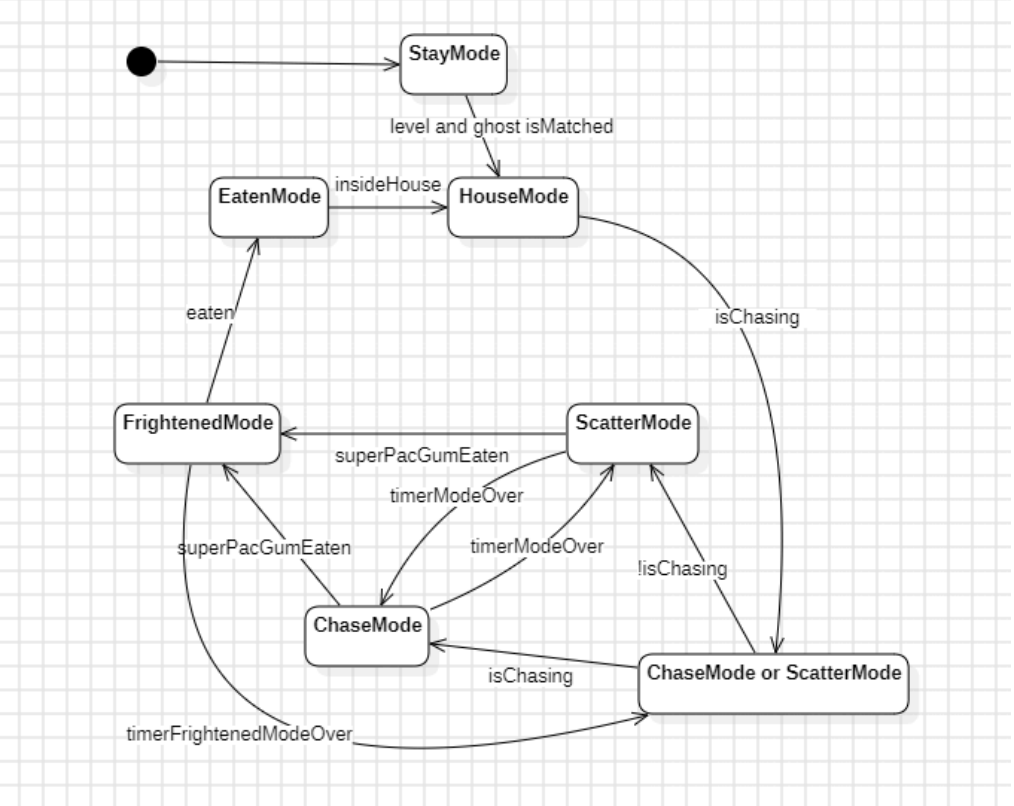
       pacgum 생성 시 pacgum이 아이템으로 변환되는 확률을 레벨에 비례하게 설정하였다.

    b. 적용된 설계 패턴 : State, Factory 패턴

    c. 설계 이유

        - 게임에서 활동하는 유령의 수를 제한하기 위해서 유령이 생성되었을 때 StayMode로 설정되도록 하였다.
            - 이 때 각각의 유령들의 활동성을 레벨 별로 확인하기는 번거롭기 때문에 ghoststate에서 staymode를 추가하여 최초로 유령이 생성될 때 해당 상태로 생성되도록 하였고 레벨 변수와 유령의 종류를 확인하여 housemode로 넘어가도록 변경하였다.
            - 다만 처음에는 game.java에 레벨 변수를 넣어 Game.java에서 유령을 생성할 때 바로 staymode에서 housemode로 전환되도록 만들었지만 이후 GameClear와 GameOver기능을 추가하는 과정에서 생성된 오브젝트를 초기화하는 과정이 필요했고 이때 레벨 변수도 같이 초기화 되는 문제가 발생하여 레벨 변수를 GamePlayPanel.java로 이동하게 되었다.
        - 추가적으로 난이도 조절을 위해 추가한 레벨 변수를 사용해 기존에 고정 확률로 생성되는 아이템을 레벨 별로 높은 확률로 생성되도록 조정하였다.


ghostStates
HouseMode
void outsideHouse()
int[] getTargetPosition()
void computeNextDir()
StayMode
void lvlGhost(int)
GhostState
Ghost ghost
void superPacGumEaten()
void timerModeOver()
void timerFrightenedModeOver()
void eaten()
void outsideHouse()
void insideHouse()
void lvlGhost(int)
int[] getTargetPosition()
void computeNextDir()
ChaseMode
void superPacGumEaten()
void timerModeOver()
int[] getTargetPosition()
ScatterMode
void superPacGumEaten()
void timerModeOver()
int[] getTargetPosition()
FrightenedMode
void eaten()
void timerFrightenedModeOver()
int[] getTargetPosition()
EatenMode
void insideHouse()
int[] getTargetPosition()
void computeNextDir()



GamePlayPanel.java

```
...
private int level = 0;
...
   public void init() {
      running = true;
      img = new BufferedImage(width, height, BufferedImage.TYPE_INT_AR
GB);
      g = (Graphics2D) img.getGraphics();

      key = new KeyHandler(this);

      game = new Game(level);
      game.delegatelvlGhost(level);

   }
...
```

Game.java

```
...
else if (dataChar.equals(".")) { //팩검(팩맨 먹이) 생성
                // random하게 Item소환.
                if (random.nextDouble() < 0.02 + 0.01 * min(0, level)) {
                //if (random.nextDouble() < 0.03) {
                    int randomIndex = random.nextInt(itemfactories.size());
                    Item item = itemfactories.get(randomIndex).makeItem(xx * c
ellSize, yy * cellSize);
                    objects.add(item);
                }
...
    public void delegatelvlGhost(int level){
        for(Ghost gh : ghosts){
            gh.getState().lvlGhost(level);
        }
    }
...
```

StayMode.java

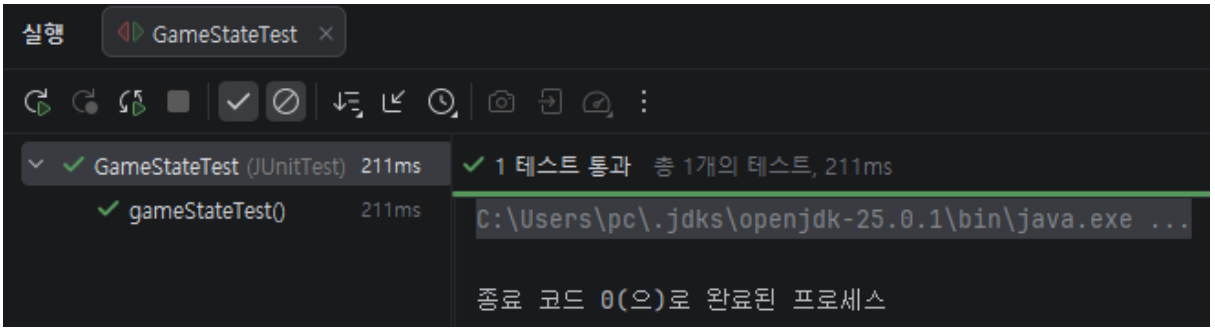
```
package game.ghostStates;

import game.entities.ghosts.*;

public class StayMode extends GhostState{
    public StayMode(Ghost ghost) {super(ghost);}
    @Override
    public void lvlGhost(int level){
        if(ghost.getClass()== Blinky.class){ghost.switchHouseMode();}
        if(ghost.getClass()== Clyde.class&&level>0){ghost.switchHouseMode
();}
        if(ghost.getClass()== Inky.class&&level>1){ghost.switchHouseMode
();}
        if(ghost.getClass()== Pinky.class&&level>2){ghost.switchHouseMode
();}
```

```
    }
}
```

3. 테스트 케이스

1. GameStateTest

게임에서 state 변경이 일어나는 상황과 같은 상황에서 제대로 게임의 state가 정상적으로 변경되는지 확인한다.

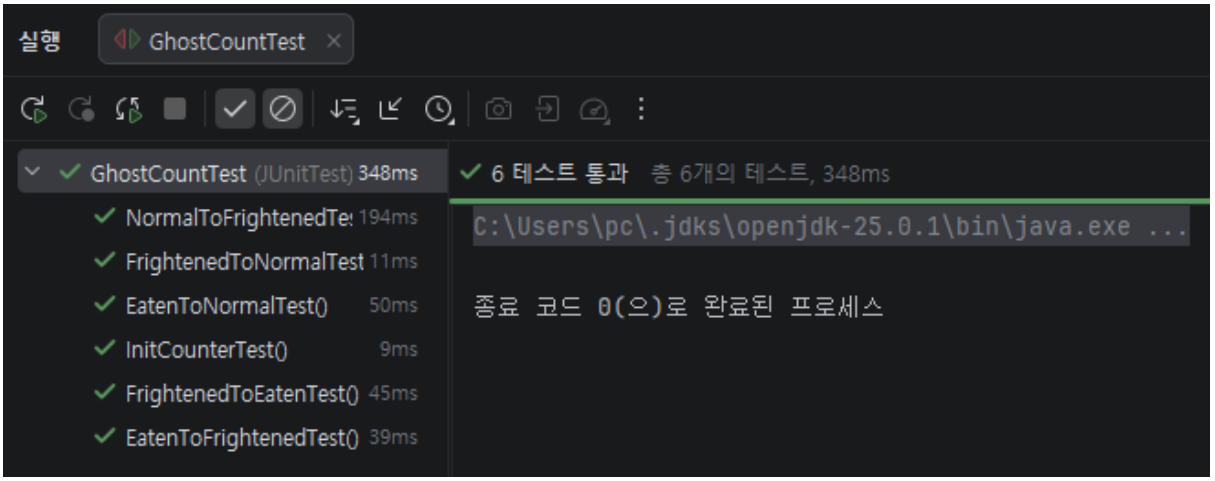



```
package JUnitTest;

import game.Game;
import game.GameLauncher;
import game.GameplayPanel;
import game.UIPanel;
import game.gameState.GameClearMode;
import game.gameState.GameOverMode;
import game.gameState.GameState;
import game.gameState.RunningMode;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import javax.swing.*;
import java.awt.*;
import java.io.IOException;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertInstanceOf;
```

```
public class GameStateTest {
    Game game=new Game(0);
    UIPanel ui;
    JPanel gameWindow = new JPanel(new BorderLayout());

    @BeforeEach
    public void setUp() {
        try {
            gameWindow.add(new GameplayPanel(448,496), BorderLayout.CE
NTER); //448 496
        } catch (
                IOException e) {
            e.printStackTrace();
        }
        ui=new UIPanel(0,0);
        gameWindow.add(ui, BorderLayout.NORTH);
        ui.updateScore(100);
    }
    @Test
    public void gameStateTest() {
        assertInstanceOf(RunningMode.class,game.getGameState(), "Running
Mode");
        game.getGameState().gameClear();
        assertInstanceOf(GameClearMode.class,game.getGameState(), "Gam
eClearMode");
        game.getGameState().retryGame();
        game.getGameState().die();
        assertInstanceOf(GameOverMode.class,game.getGameState(), "Game
ClearMode");
    }

}
```

2. GhostCountTest

각각의 ghost들의 상태 전환에 따라 static 카운터 변수가 정확하게 변화하였는지 테스트한다. 실제로 발생할 수 있는 다양한 상황을 가정하여 frightenedCnt와 eatenCnt가 정확하게 변화하는지 확인한다.




```
package JUnitTest;

import game.entities.ghosts.*;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import java.util.ArrayList;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class GhostCountTest {
    ArrayList<Ghost> ghosts;

    @BeforeEach
    void init() {
        ghosts = new ArrayList<>();
        ghosts.add(new Blinky(0,0));
        ghosts.add(new Clyde(0,0));
        ghosts.add(new Inky(0,0));
        ghosts.add(new Pinky(0,0));
        ghosts.forEach(g -> g.getState().lvlGhost(3));
    }

    @Test
    void InitCounterTest() {
        assertEquals(0, Ghost.getFrightenedCnt(), "FrightenedCnt의 초기값은 0
이어야 함.");
```

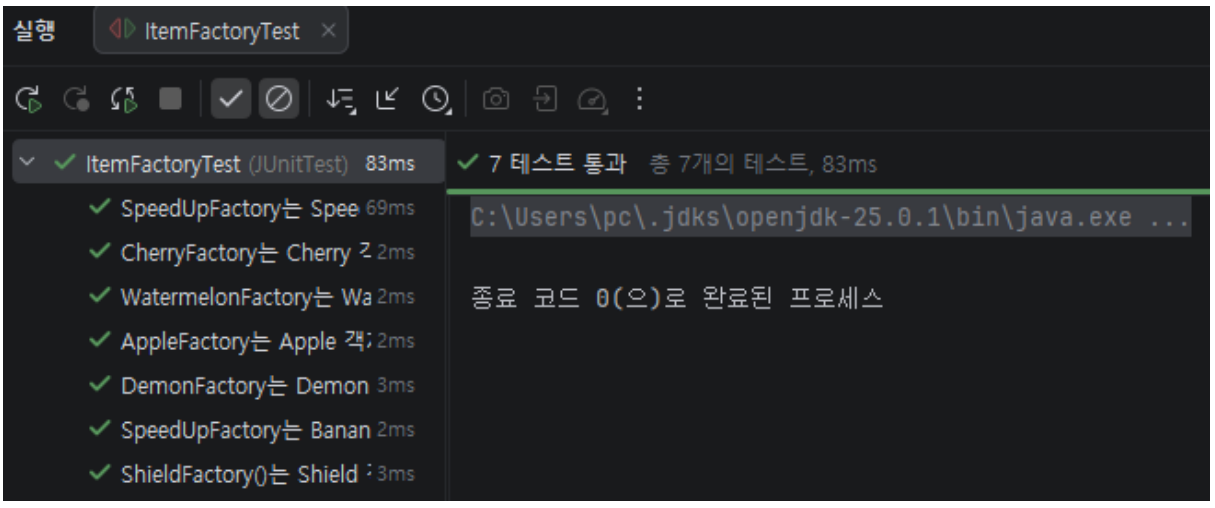
```
        assertEquals(0, Ghost.getEatenCnt(), "EatenCnt의 초기값은 0이어야
함.");
    }

    @Test
    void NormalToFrightenedTest() {
        for(int i = 0; i < 4; i++) {
            init();
            for(int j = 0; j <= i; j++) {
                ghosts.get(j).switchFrightenedMode();
            }
            assertEquals(i + 1, Ghost.getFrightenedCnt(), "공포 모드로 변경 시, 변
경된 유령 수만큼 FrightenedCnt가 증가해야 함.");
        }
    }

    @Test
    void FrightenedToNormalTest() {
        for(int i = 0; i < 4; i++) {
            for(int j = 0; j <= i; j++) {
                ghosts.get(j).switchFrightenedMode();
            }

            for(Ghost g : ghosts) {
                g.switchChaseModeOrScatterMode();
            }
            assertEquals(0, Ghost.getFrightenedCnt(), "모두 일반 모드로 돌아왔다
면, FrightenedCnt는 0이어야 함.");
        }
    }

    @Test
    void FrightenedToEatenTest() {
        for(int i = 0; i < 4; i++) {
            init();
            for(Ghost g : ghosts) {
                g.switchFrightenedMode();
            }
```

```
            for(int j = 0; j <= i; j++) {
                ghosts.get(j).switchEatenMode();
            }
            assertEquals(i + 1, Ghost.getEatenCnt(), "먹힌 유령 개수만큼 eatenCnt
가 증가해야 함.");
        }
    }

    @Test
    void EatenToFrightenedTest() { //먹힌 유령이 house로 돌아갔지만 아직 Fright
enedTimer시간이 남은 상황
        for(int i = 0; i < 4; i++) {
            init();
            for(Ghost g : ghosts) {
                g.switchFrightenedMode();
            }

            for(int j = 0; j <= i; j++) {
                ghosts.get(j).switchEatenMode();
                ghosts.get(j).switchHouseMode();
                ghosts.get(j).switchChaseModeOrScatterMode();
            }
            assertEquals(0, Ghost.getEatenCnt(), "먹힌 유령이 다 house로 복귀했
다면, eatenCnt는 0 이어야 함.");
            assertEquals(3 - i, Ghost.getFrightenedCnt(), "먹힌 유령이 아닌 유령들
의 개수만큼 FrightenedCnt가 존재해야 함.");
        }
    }

    @Test
    void EatenToNormalTest() { //먹힌 유령이 house로 돌아갔는데 이미 Frighten
edTimer가 끝난 상황
        for(int i = 0; i < 4; i++) {
            init();
            for(Ghost g : ghosts) {
                g.switchFrightenedMode();
            }
```

```
            for(int j = 0; j <= i; j++) {
                ghosts.get(j).switchEatenMode();
                ghosts.get(j).switchHouseMode();
                ghosts.get(j).switchChaseModeOrScatterMode();
            }

            for(int j = i + 1; j < 4; j++) {
                ghosts.get(j).switchChaseModeOrScatterMode();
            }

            assertEquals(0, Ghost.getEatenCnt(), "먹힌 유령이 다 house로 복귀했다면, eatenCnt는 0이어야 함.");
            assertEquals(0, Ghost.getFrightenedCnt(), "공포 타이머가 이미 끝났으므로 FrightenedCnt가 0이어야 함.");
        }
    }
}
```

3. ItemFactoryTest

각각의 아이템을 생성하고 생성된 아이템의 점수와 위치를 검사한다.




```
package JUnitTest;

import game.entities.items.*;
import game.itemFactory.*;
```

```
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

public class ItemFactoryTest {

    @Test
    @DisplayName("CherryFactory는 Cherry 객체를 생성하고, 점수는 200점이어야 한다")
    void testCherryFactory() {
        AbstractItemFactory factory = new CherryFactory();
        int x = 100;
        int y = 200;

        Item item = factory.makeItem(x, y);
        assertInstanceOf(Cherry.class, item, "생성된 객체는 Cherry 타입이어야 합니다.");
        assertEquals(200, item.getPoint(), "Cherry의 점수는 200점이어야 합니다.");
        assertEquals(x, item.getxPos(),  "X 좌표가 일치해야 합니다.");
        assertEquals(y, item.getyPos(), "Y 좌표가 일치해야 합니다.");
        assertNotNull(item.getImage(), "이미지 로드 실패");
        item.destroy();
        assertEquals(-32, item.getxPos(), "X 좌표가 일치해야 합니다.");
        assertEquals(-32, item.getyPos(), "Y 좌표가 일치해야 합니다.");

    }

    @Test
    @DisplayName("AppleFactory는 Apple 객체를 생성하고, 점수는 500점이어야 한다")
    void testAppleFactory() {
        AbstractItemFactory factory = new AppleFactory();
        int x = 100;
        int y = 200;

        Item item = factory.makeItem(x, y);
```

```
        assertInstanceOf(Apple.class, item, "생성된 객체는 Apple 타입이어야 합니다.");
        assertEquals(500, item.getPoint(), "Apple의 점수는 500점이어야 합니다.");
        assertNotNull(item.getImage(), "이미지 로드 실패");
        assertEquals(x, item.getxPos(), "X 좌표가 일치해야 합니다.");
        assertEquals(y, item.getyPos(), "Y 좌표가 일치해야 합니다.");
        assertNotNull(item.getImage(), "이미지 로드 실패");
        item.destroy();
        assertEquals(-32, item.getxPos(), "X 좌표가 일치해야 합니다.");
        assertEquals(-32, item.getyPos(), "Y 좌표가 일치해야 합니다.");
    }

    @Test
    @DisplayName("WatermelonFactory는 Watermelon 객체를 생성하고, 점수는 1000점이어야 한다")
    void testWatermelonFactory() {
        AbstractItemFactory factory = new WatermelonFactory();
        int x = 100;
        int y = 200;

        Item item = factory.makeItem(x, y);
        assertInstanceOf(Watermelon.class, item, "생성된 객체는 Watermelon 타입이어야 합니다.");
        assertEquals(1000, item.getPoint(), "Watermelon의 점수는 1000점이어야 합니다.");
        assertEquals(x, item.getxPos(), "X 좌표가 일치해야 합니다.");
        assertEquals(y, item.getyPos(), "Y 좌표가 일치해야 합니다.");
        assertNotNull(item.getImage(), "이미지 로드 실패");
        item.destroy();
        assertEquals(-32, item.getxPos(), "X 좌표가 일치해야 합니다.");
        assertEquals(-32, item.getyPos(), "Y 좌표가 일치해야 합니다.");
    }

    @Test
    @DisplayName("ShieldFactory()는 Shield 객체를 생성하고, 점수는 0점이어야 한다")
    void testShieldFactory() {
```

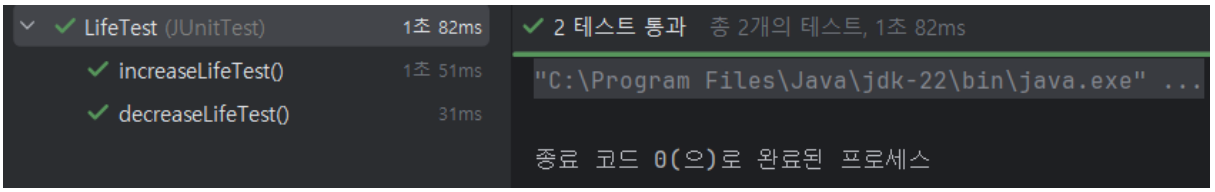
```
        AbstractItemFactory factory = new ShieldFactory();
        int x = 100;
        int y = 200;

        Item item = factory.makeItem(x, y);
        assertInstanceOf(Shield.class, item, "생성된 객체는 Shield 타입이어야 합
니다.");
        assertEquals(0, item.getPoint(), "Shield의 점수는 0점이어야 합니다.");
        assertEquals(x, item.getxPos(), "X 좌표가 일치해야 합니다.");
        assertEquals(y, item.getyPos(), "Y 좌표가 일치해야 합니다.");
        assertNotNull(item.getImage(), "이미지 로드 실패");
        assertNotNull(((EffectItem)item).getEffectCommand(), "effect commna
nd가 없음");
        item.destroy();
        assertEquals(-32, item.getxPos(), "X 좌표가 일치해야 합니다.");
        assertEquals(-32, item.getyPos(), "Y 좌표가 일치해야 합니다.");
    }

    @Test
    @DisplayName("SpeedUpFactory는 SpeedUp 객체를 생성하고, 점수는 0점
이어야 한다")
    void testSpeedUpFactory() {
        AbstractItemFactory factory = new SpeedUpFactory();
        int x = 100;
        int y = 200;

        Item item = factory.makeItem(x, y);
        assertInstanceOf(SpeedUp.class, item, "생성된 객체는 SpeedUp 타입이
어야 합니다.");
        assertEquals(0, item.getPoint(), "SpeedUp의 점수는 0점이어야 합니다.");
        assertEquals(x, item.getxPos(), "X 좌표가 일치해야 합니다.");
        assertEquals(y, item.getyPos(), "Y 좌표가 일치해야 합니다.");
        assertNotNull(((EffectItem)item).getEffectCommand(), "effect commna
nd가 없음");
        assertNotNull(item.getImage(), "이미지 로드 실패");
        item.destroy();
        assertEquals(-32, item.getxPos(), "X 좌표가 일치해야 합니다.");
        assertEquals(-32, item.getyPos(), "Y 좌표가 일치해야 합니다.");
```

```
    }

    // speed down
    @Test
    @DisplayName("SpeedUpFactory는 Banana 객체를 생성하고, 점수는 0점이어야 한다")
    void testBananaFactory() {
        AbstractItemFactory factory = new BananaFactory();
        int x = 100;
        int y = 200;

        Item item = factory.makeItem(x, y);
        assertInstanceOf(Banana.class, item, "생성된 객체는 Banana 타입이어야 합니다.");
        assertEquals(0, item.getPoint(), "Banana의 점수는 0점이어야 합니다.");
        assertEquals(x, item.getxPos(), "X 좌표가 일치해야 합니다.");
        assertEquals(y, item.getyPos(), "Y 좌표가 일치해야 합니다.");
        assertNotNull(item.getImage(), "이미지 로드 실패");
        assertNotNull(((EffectItem)item).getEffectCommand(), "effect commnand가 없음");
        item.destroy();
        assertEquals(-32, item.getxPos(), "X 좌표가 일치해야 합니다.");
        assertEquals(-32, item.getyPos(), "Y 좌표가 일치해야 합니다.");
    }

    // confuse
    @Test
    @DisplayName("DemonFactory는 Demon 객체를 생성하고, 점수는 0점이어야 한다")
    void testDemonFactory() {
        AbstractItemFactory factory = new DemonFactory();
        int x = 100;
        int y = 200;

        Item item = factory.makeItem(x, y);
        assertInstanceOf(Demon.class, item, "생성된 객체는 Demon 타입이어야 합니다.");
        assertEquals(0, item.getPoint(), "Demon의 점수는 0점이어야 합니다.");
```

```
        assertEquals(x, item.getxPos(), "X 좌표가 일치해야 합니다.");
        assertEquals(y, item.getyPos(), "Y 좌표가 일치해야 합니다.");
        assertNotNull(item.getImage(), "이미지 로드 실패");
        assertNotNull(((EffectItem)item).getEffectCommand(), "effect commna
nd가 없음");
        item.destroy();
        assertEquals(-32, item.getxPos(), "X 좌표가 일치해야 합니다.");
        assertEquals(-32, item.getyPos(), "Y 좌표가 일치해야 합니다.");
    }
}
```

4. LifeTest

라이프의 증가와 감소가 인게임 흐름에서 잘 적용되는지 확인한다. 증가는 사과 아이템을 10개 먹는 상황을 가정하여 테스트하고, 감소는 팩맨이 사망한 상황을 가정하여 테스트한다.

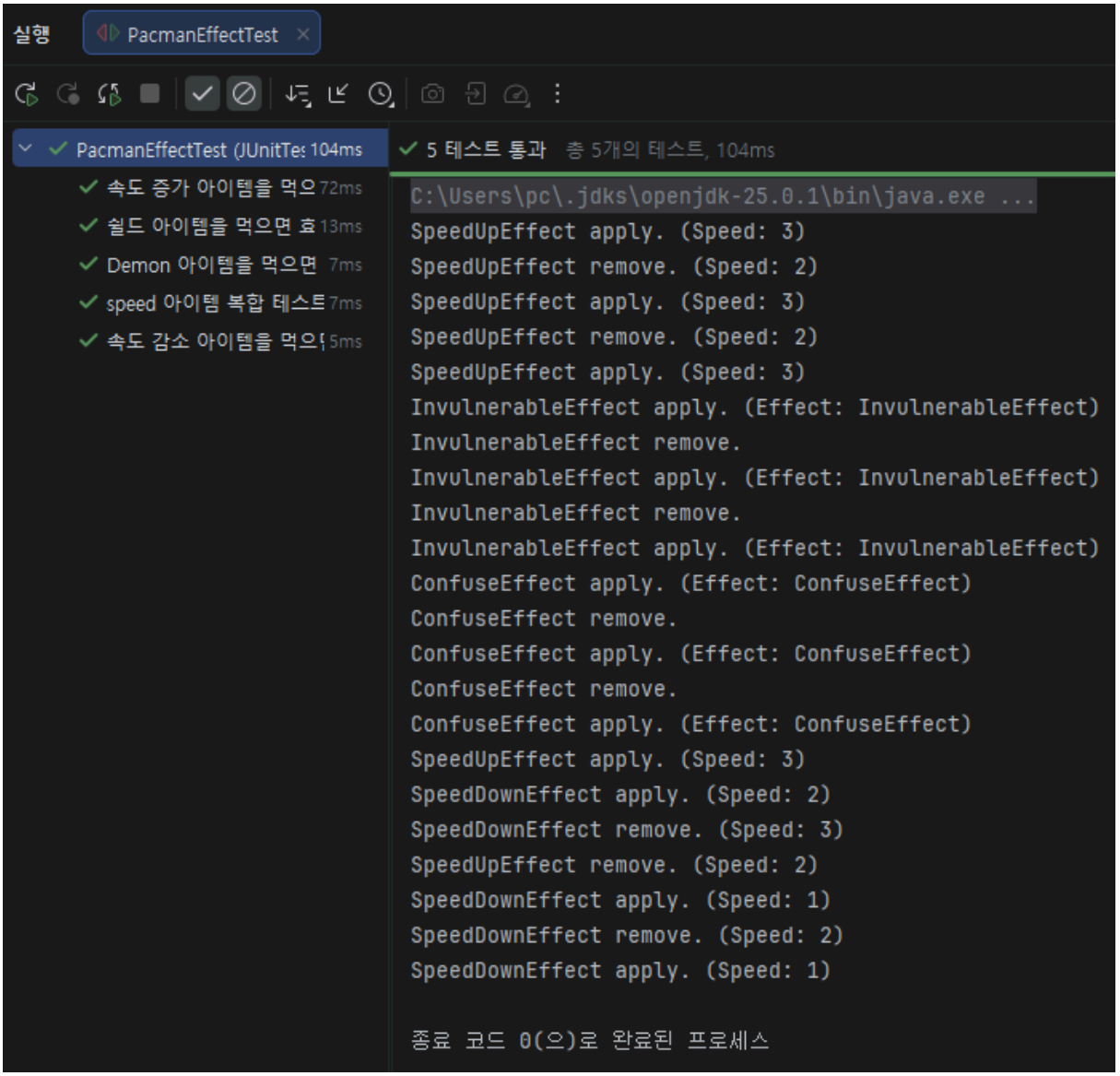



```
public class LifeTest {
    Pacman pacman;
    UIPanel ui;

    @BeforeEach
    public void init() {
        ui = new UIPanel(0, 0);
        pacman = new Pacman(0, 0);
        ui.setPacman(pacman);
        pacman.registerObserver(ui);
    }

    @Test
    public void increaseLifeTest() {
        AbstractItemFactory factory = new AppleFactory();
        Item item = factory.makeItem(0, 0);
```

```
        int before = pacman.getLifeCnt();
        for(int i = 0; i < 10; i++) { //500 * 10 = 5000점 획득
            pacman.notifyObserverItemEaten(item);
        }
        int after = pacman.getLifeCnt();
        assertEquals(before + 1, after, "라이프가 1 증가해야 함.");
    }

    @Test
    public void decreaseLifeTest() {
        int before = pacman.getLifeCnt();
        pacman.notifyObserverPacmanDead();
        int after = pacman.getLifeCnt();
        assertEquals(before - 1, after, "라이프가 1 감소해야 함.");
    }
}
```

5. PacmanEffectTest

효과가 있는 아이템을 먹은 경우 팩맨에 적용된 효과가 아이템의 효과와 일치하는지 확인한다. 또한 지속시간이 끝났을 경우 정상적으로 효과가 사라지는지 확인한다.




```
package JUnitTest;

import game.entities.Pacman;
import game.entities.ghosts.Ghost;
import game.entities.items.*;
import game.ghostFactory.AbstractGhostFactory;
import game.ghostFactory.BlinkyFactory;
import game.itemFactory.*;
import game.pacmanEffect.EffectCommand;
import game.pacmanEffect.SpeedUpEffect;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
```

```
import static org.junit.jupiter.api.Assertions.*;

class PacmanEffectTest {

    private Pacman pacman;

    @BeforeEach
    void setUp() {
        // 테스트할 때마다 새로운 팩맨 생성 (초기 속도 2 가정)
        pacman = new Pacman(100, 100);
    }

    @Test
    @DisplayName("속도 증가 아이템을 먹으면 효과가 적용되고, 지속시간이 지나면 사라져야 한다")
    void testSpeedUpEffectLifecycle() {
        // 5초, 속도 +1 증가 아이템
        int duration = 5;
        int boostAmount = 1;
        int initialSpeed = pacman.getSpd(); // 기본 속도 저장
        AbstractItemFactory itemFactory = new SpeedUpFactory();
        Item item = itemFactory.makeItem(40, 40);
        EffectCommand speedUp = ((EffectItem)item).getEffectCommand();

        // When: 효과 적용 (아이템 획득 시뮬레이션)
        pacman.addEffect(speedUp);

        // Then  즉시 효과가 적용되었는지 확인
        assertEquals(initialSpeed + boostAmount, pacman.getSpd(), "속도가 즉시 증가해야 합니다.");
        assertEquals(pacman.findEffectClass(speedUp).getClass(), speedUp.getClass(), "활성화된 효과 리스트에 추가되어야 합니다.");

        // 여전히 효과 유지
        for (int i = 0; i < duration * 60 - 1; i++) {
            pacman.updateEffect();
        }
```

```
        assertEquals(initialSpeed + boostAmount, pacman.getSpd(), "지속시간
내에는 속도가 유지되어야 합니다.");

        // 효과 제거
        pacman.updateEffect();
        assertEquals(initialSpeed, pacman.getSpd(), "지속시간이 지나면 원래 속
도로 돌아와야 합니다.");
        assertNull(pacman.findEffectClass(speedUp), "활성화된 효과 리스트에 제
거되어야 합니다.");

        speedUp = ((EffectItem)item).getEffectCommand();
        pacman.addEffect(speedUp);
        for (int i = 0; i < duration * 60 - 10; i++) {
            pacman.updateEffect();
        }
        item = itemFactory.makeItem(40, 40);
        speedUp = ((EffectItem)item).getEffectCommand();
        pacman.addEffect(speedUp);
        assertEquals(0, pacman.findEffectClass(speedUp).getTimer(), "지속시
간이 초기화 되어야함.");
    }


    @Test
    @DisplayName("속도 감소 아이템을 먹으면 효과가 적용되고, 지속시간이 지나면
사라져야 한다")
    void testSpeedDownEffectTimerReset() {
        // 2초, 속도 -1 증가 아이템
        int duration = 2;
        int boostAmount = -1;
        int initialSpeed = pacman.getSpd(); // 기본 속도 저장
        AbstractItemFactory itemFactory = new BananaFactory();
        Item item = itemFactory.makeItem(40, 40);
        EffectCommand speedDown = ((EffectItem)item).getEffectCommand
();

        // When: 효과 적용 (아이템 획득 시뮬레이션)
        pacman.addEffect(speedDown);
```

```
        // Then  즉시 효과가 적용되었는지 확인
        assertEquals(initialSpeed + boostAmount, pacman.getSpd(), "속도가 즉시 증가해야 합니다.");
        assertEquals(pacman.findEffectClass(speedDown).getClass(), speedDown.getClass(), "활성화된 효과 리스트에 추가되어야 합니다.");

        // 여전히 효과 유지
        for (int i = 0; i < duration * 60 - 1; i++) {
            pacman.updateEffect();
        }
        assertEquals(initialSpeed + boostAmount, pacman.getSpd(), "지속시간 내에는 속도가 유지되어야 합니다.");

        // 효과 제거
        pacman.updateEffect();
        assertEquals(initialSpeed, pacman.getSpd(), "지속시간이 지나면 원래 속도로 돌아와야 합니다.");
        assertNull(pacman.findEffectClass(speedDown), "활성화된 효과 리스트에 제거되어야 합니다.");



        item = itemFactory.makeItem(40, 40);
        speedDown = ((EffectItem)item).getEffectCommand();
        pacman.addEffect(speedDown);
        assertEquals(0, pacman.findEffectClass(speedDown).getTimer(), "지속시간이 초기화 되어야함.");
    }


    @Test
    @DisplayName("speed 아이템 복합 테스트")
    void testSpeedEffectTimerReset() {
        // speed up, speed down 둘다 적용
        int speedDownDuration = 2;
        int speedUpDuration = 5;
        int boostAmount = 1;
        int initialSpeed = pacman.getSpd(); // 기본 속도 저장
```

```
        AbstractItemFactory speedUpfactory = new SpeedUpFactory();
        AbstractItemFactory speedDownfactory = new BananaFactory();

        Item speedDownitem = speedDownfactory.makeItem(40, 40);
        EffectCommand speedDown = ((EffectItem)speedDownitem).getEffect
Command();

        Item speedUpitem = speedUpfactory.makeItem(32, 32);
        EffectCommand speedUp = ((EffectItem)speedUpitem).getEffectCom
mand();

        // When: 효과 적용 (아이템 획득 시뮬레이션)
        pacman.addEffect(speedUp);
        assertEquals(initialSpeed + boostAmount, pacman.getSpd(), "속도가 즉
시 증가해야 합니다.");
        assertEquals(pacman.findEffectClass(speedUp).getClass(), speedUp.
getClass(), "활성화된 효과 리스트에 추가되어야 합니다.");

        pacman.addEffect(speedDown);
        assertEquals(initialSpeed, pacman.getSpd(), "슬로우 아이템을 먹으므로써
속도가 원상 복귀 되어야합니다.");
        assertEquals(pacman.findEffectClass(speedDown).getClass(), speedD
own.getClass(), "활성화된 효과 리스트에 추가되어야 합니다.");



        // 여전히 효과 유지
        for (int i = 0; i < speedDownDuration * 60 - 1; i++) {
            pacman.updateEffect();
        }
        assertEquals(initialSpeed, pacman.getSpd(), "지속시간 내에는 속도가 유
지되어야 합니다.");

        // 속도 감소 효과 제거
        pacman.updateEffect();
        assertEquals(initialSpeed + boostAmount, pacman.getSpd(), "speedD
own이 종료되어 다시 속도가 증가해야 합니다.");
        assertNull(pacman.findEffectClass(speedDown), "활성화된 효과 리스트
에 제거되어야 합니다.");
```

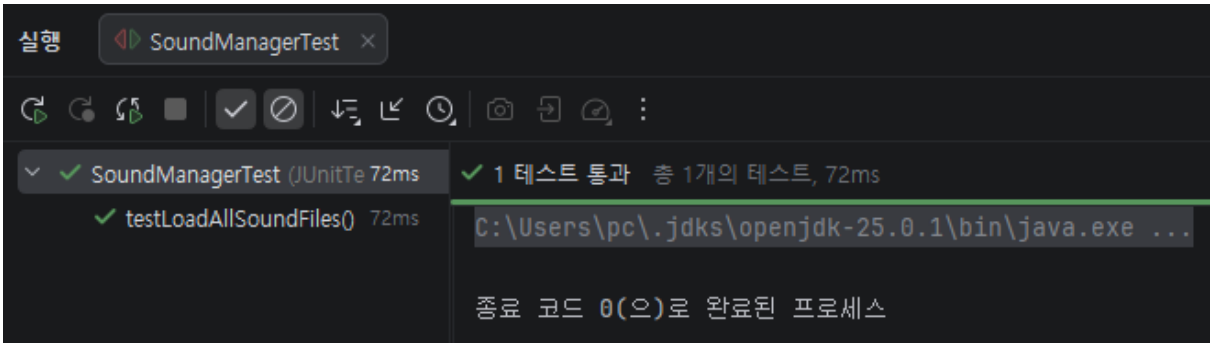
```
        pacman.removeEffectAll();
        assertEquals(initialSpeed, pacman.getSpd(), "효과 제거 -> 속도 다시 원상복귀해야 합니다.");
        assertNull(pacman.findEffectClass(speedDown), "활성화된 효과 리스트에 제거되어야 합니다.");
    }



    @Test
    @DisplayName("쉴드 아이템을 먹으면 효과가 적용되고, 지속시간이 지나면 사라져야 한다")
    void testShieldEffectLifecycle() {
        // 5초, 무적 아이템
        int duration = 5;
        AbstractItemFactory itemFactory = new ShieldFactory();
        Item item = itemFactory.makeItem(40, 40);
        EffectCommand shieldEffect = ((EffectItem)item).getEffectCommand();

        AbstractGhostFactory ghostFactory = new BlinkyFactory();
        Ghost ghost = ghostFactory.makeGhost(40, 40);

        // When: 효과 적용 (아이템 획득 시뮬레이션)
        pacman.addEffect(shieldEffect);

        // Then  즉시 효과가 적용되었는지 확인
        assertTrue(pacman.getInvulnerable(), "무적이어야합니다.");
        assertEquals(pacman.findEffectClass(shieldEffect).getClass(), shieldEffect.getClass(), "활성화된 효과 리스트에 추가되어야 합니다.");

        // 여전히 효과 유지
        for (int i = 0; i < duration * 60 - 1; i++) {
            pacman.updateEffect();
        }
        assertTrue(pacman.getInvulnerable(), "무적이어야합니다.");
```

```
        // 효과 제거
        pacman.updateEffect();
        assertFalse(pacman.getInvulnerable(), "무적이 아니어야 합니다.");
        assertNull(pacman.findEffectClass(shieldEffect), "활성화된 효과 리스트에
제거되어야 합니다.");

        shieldEffect = ((EffectItem)item).getEffectCommand();
        pacman.addEffect(shieldEffect);
        for (int i = 0; i < duration * 60 - 10; i++) {
            pacman.updateEffect();
        }
        item = itemFactory.makeItem(40, 40);
        shieldEffect = ((EffectItem)item).getEffectCommand();
        pacman.addEffect(shieldEffect);
        assertEquals(0, pacman.findEffectClass(shieldEffect).getTimer(), "지속
시간이 초기화 되어야함.");
    }



    @Test
    @DisplayName("Demon 아이템을 먹으면 효과가 적용되고, 지속시간이 지나면
사라져야 한다")
    void testConfuseEffectLifecycle() {
        // 5초, 방향 반대
        int duration = 5;
        AbstractItemFactory itemFactory = new DemonFactory();
        Item item = itemFactory.makeItem(40, 40);
        EffectCommand confuseEffect = ((EffectItem)item).getEffectCommand
();

        // When: 효과 적용 (아이템 획득 시뮬레이션)
        pacman.addEffect(confuseEffect);

        // Then  즉시 효과가 적용되었는지 확인
        assertTrue(pacman.getConfuse(), "confuse 상태이어야합니다.");
        assertEquals(pacman.findEffectClass(confuseEffect).getClass(), confu
seEffect.getClass(), "활성화된 효과 리스트에 추가되어야 합니다.");
```

```
        // 여전히 효과 유지
        for (int i = 0; i < duration * 60 - 1; i++) {
            pacman.updateEffect();
        }
        assertTrue(pacman.getConfuse(), "confuse 상태이어야합니다.");

        // 효과 제거
        pacman.updateEffect();
        assertFalse(pacman.getConfuse(), "confuse가 끝났어야 합니다.");
        assertNull(pacman.findEffectClass(confuseEffect), "활성화된 효과 리스
트에 제거되어야 합니다.");

        confuseEffect = ((EffectItem)item).getEffectCommand();
        pacman.addEffect(confuseEffect);
        for (int i = 0; i < duration * 60 - 10; i++) {
            pacman.updateEffect();
        }
        item = itemFactory.makeItem(40, 40);
        confuseEffect = ((EffectItem)item).getEffectCommand();
        pacman.addEffect(confuseEffect);
        assertEquals(0, pacman.findEffectClass(confuseEffect).getTimer(), "지
속시간이 초기화 되어야함.");
    }
}
```

6. SoundManagerTest

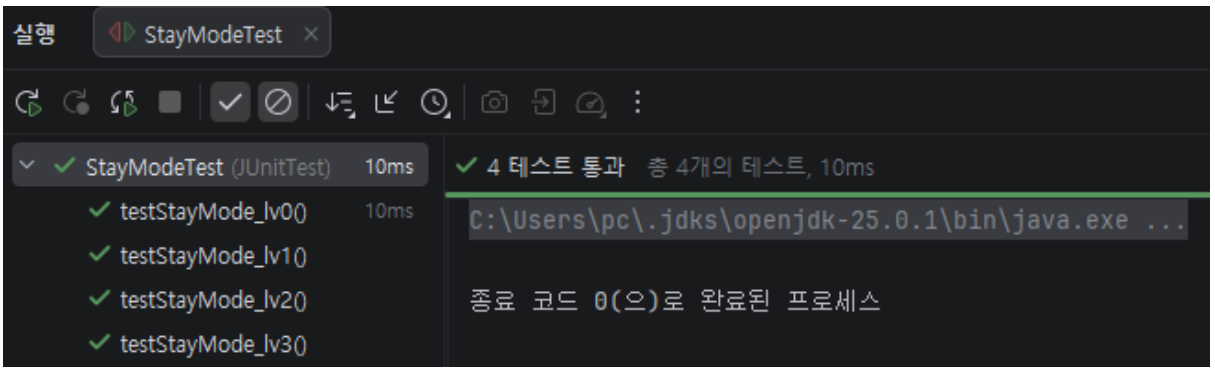
SoundManager가 정상적으로 동작하는지 확인한다. SoundManager의 Map<Sound, Clip>에 모든 사운드 파일이 정상적으로 맵핑되어 있는지, 그리고 유효한 길이를 가진 파일인지 테스트한다.



```
package JUnitTest;

import game.SoundManager;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

import javax.sound.sampled.Clip;

class SoundManagerTest {

    @Test
    public void testLoadAllSoundFiles() {
        SoundManager sm = SoundManager.getInstance();

        for (var s : SoundManager.Sound.values()) {
            Clip clip = sm.getClip(s);
            Assertions.assertNotNull(clip, "Clip should not be null for " + s);
            Assertions.assertTrue(clip.getFrameLength() > 0, "Clip should have
valid length for " + s);
        }
    }
}
```

7. StayModeTest

0~3 레벨 까지 각각의 유령이 house mode로 이동해 정상적인 게임이 동작하는지 확인한다.



```
package JUnitTest;

import game.entities.ghosts.*;
import game.ghostStates.GhostState;
import game.ghostStates.HouseMode;
import game.ghostStates.StayMode;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;


public class StayModeTest {
    Ghost blinky=new Blinky(0,0);
    Ghost clyde=new Clyde(0,0);
    Ghost inky=new Inky(0,0);
    Ghost pinky=new Pinky(0,0);
    GhostState stayMode = new StayMode(blinky);
    GhostState houseMode = new HouseMode(blinky);

    @Test
    public void testStayMode_lv0() {
        GhostState state = blinky.getState();
        state.lvlGhost(0);
        assertEquals(blinky.getState().getClass(),houseMode.getClass(),"blink
y - house");
        state = clyde.getState();
        state.lvlGhost(0);
        assertEquals(clyde.getState().getClass(),stayMode.getClass(),"clyde -
stay");
        state = inky.getState();
        state.lvlGhost(0);
        assertEquals(inky.getState().getClass(),stayMode.getClass(),"inky - st
ay");
        state = pinky.getState();
        state.lvlGhost(0);
        assertEquals(pinky.getState().getClass(),stayMode.getClass(),"pinky -
stay");
    }
```

```
    @Test
    public void testStayMode_lv1() {
        GhostState state = blinky.getState();
        state.lvlGhost(1);
        assertEquals(blinky.getState().getClass(),houseMode.getClass(),"blink
y - house");
        state = clyde.getState();
        state.lvlGhost(1);
        assertEquals(clyde.getState().getClass(),houseMode.getClass(),"clyde
- house");
        state = inky.getState();
        state.lvlGhost(1);
        assertEquals(inky.getState().getClass(),stayMode.getClass(),"inky - st
ay");
        state = pinky.getState();
        state.lvlGhost(1);
        assertEquals(pinky.getState().getClass(),stayMode.getClass(),"pinky -
stay");
    }
    @Test
    public void testStayMode_lv2() {
        GhostState state = blinky.getState();
        state.lvlGhost(2);
        assertEquals(blinky.getState().getClass(),houseMode.getClass(),"blink
y - house");
        state = clyde.getState();
        state.lvlGhost(2);
        assertEquals(clyde.getState().getClass(),houseMode.getClass(),"clyde
- house");
        state = inky.getState();
        state.lvlGhost(2);
        assertEquals(inky.getState().getClass(),houseMode.getClass(),"inky -
house");
        state = pinky.getState();
        state.lvlGhost(2);
        assertEquals(pinky.getState().getClass(),stayMode.getClass(),"pinky -
stay");
    }
```

```
    @Test
    public void testStayMode_lv3() {
        GhostState state = blinky.getState();
        state.lvlGhost(3);
        assertEquals(blinky.getState().getClass(),houseMode.getClass(),"blink
y - house");
        state = clyde.getState();
        state.lvlGhost(3);
        assertEquals(clyde.getState().getClass(),houseMode.getClass(),"clyde
- house");
        state = inky.getState();
        state.lvlGhost(3);
        assertEquals(inky.getState().getClass(),houseMode.getClass(),"inky -
house");
        state = pinky.getState();
        state.lvlGhost(3);
        assertEquals(pinky.getState().getClass(),houseMode.getClass(),"pinky
- house");
    }

}
```

4. Github 프로젝트 활용 요약

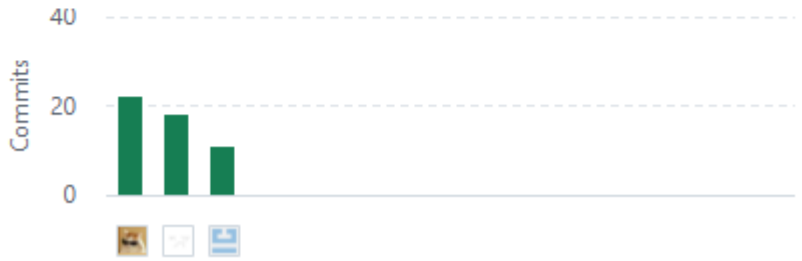
1. github 주소 : https://github.com/2025DesignPatternTeam5/PacManDP
2. 프로젝트 progress history 스크린샷

Summary

Excluding merges, **3 authors** have pushed **47 commits** to main and **51 commits** to all branches.

On main, **124 files** have changed and there have been **5,217 additions** and **0 deletions**

Top Committers

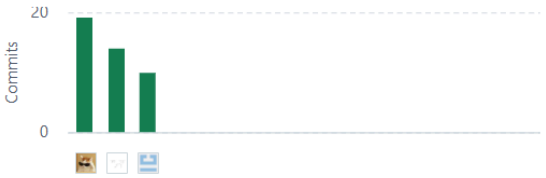


Summary

Excluding merges, **3 authors** have pushed **43 commits** to main and **43 commits** to all branches.

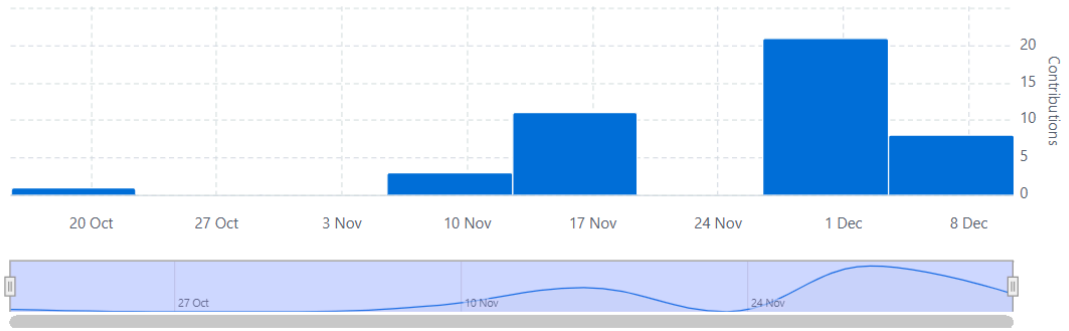
On main, **120 files** have changed and there have been **3,847 additions** and **0 deletions**

Top Committers



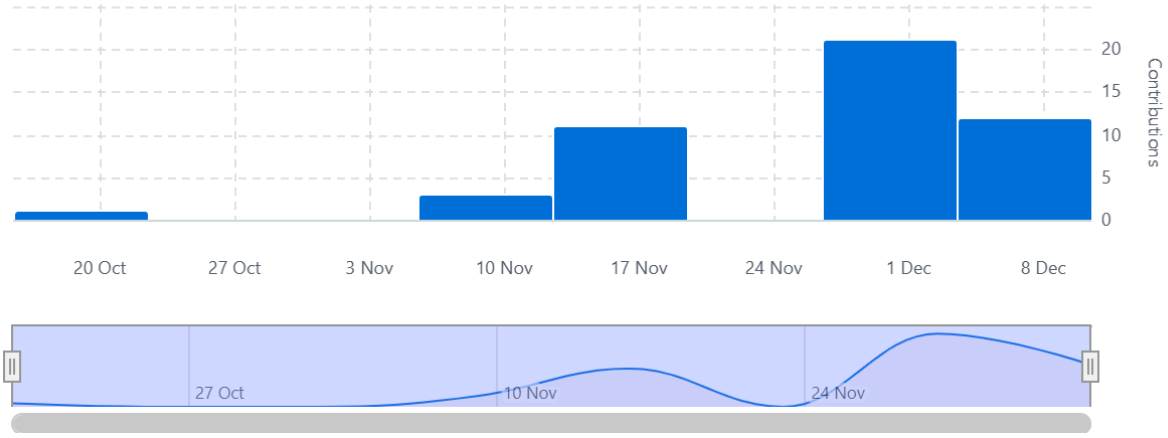
Commits over time

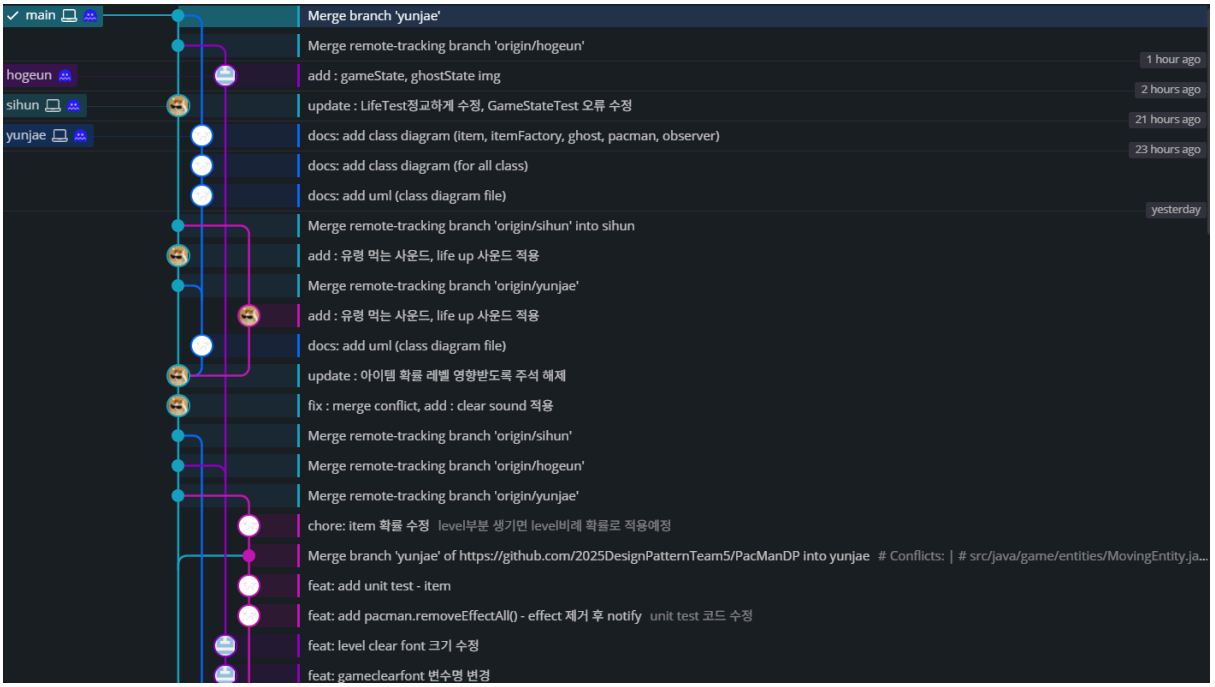
Weekly from 2025년 10월 19일 to 2025년 12월 7일

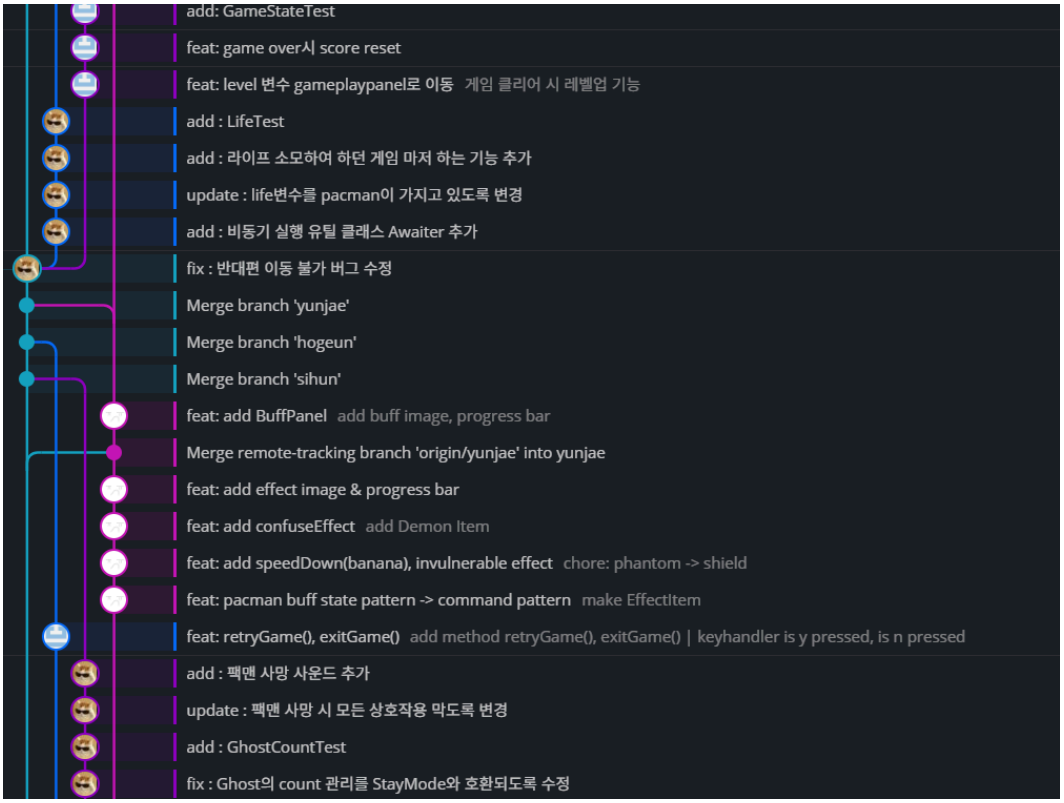


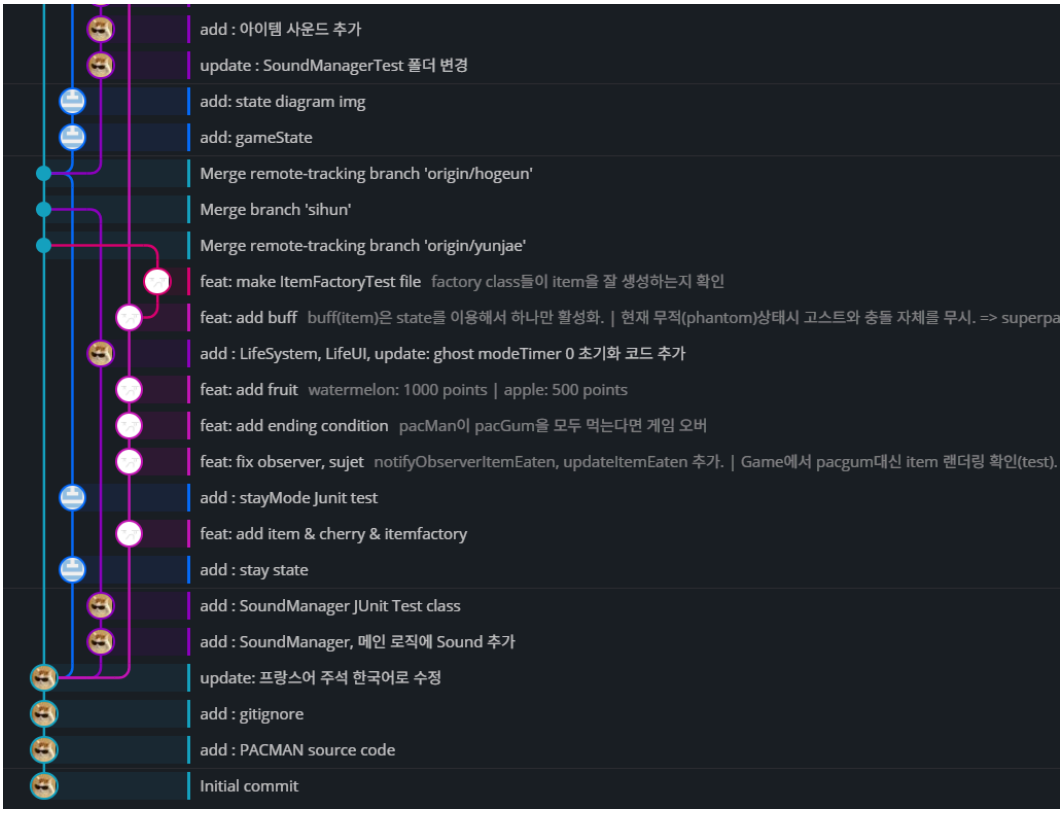
Commits over time

Weekly from 2025년 10월 19일 to 2025년 12월 7일





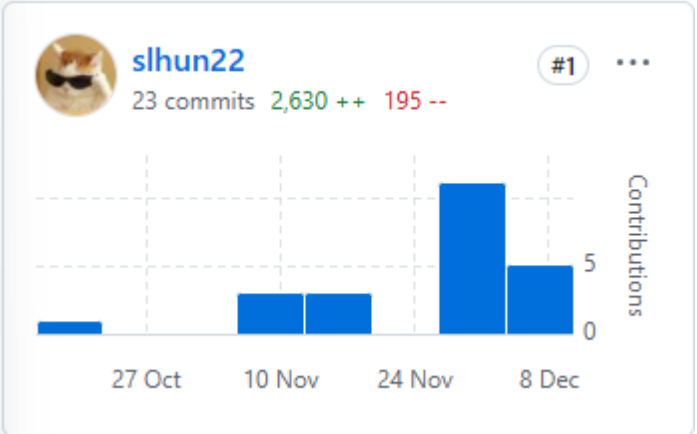

add: GameStateTest
feat: game over시 score reset
feat: level 변수 gameplaypanel로 이동 게임 클리어 시 레벨업 기능
add : LifeTest
add : 라이프 소모하여 하던 게임 마저 하는 기능 추가
update : life변수를 pacman이 가지고 있도록 변경
add : 비동기 실행 유틸 클래스 Awaiter 추가
fix : 반대편 이동 불가 버그 수정
Merge branch 'yunjae'
Merge branch 'hogeun'
Merge branch 'sihun'
feat: add BuffPanel add buff image, progress bar
Merge remote-tracking branch 'origin/yunjae' into yunjae
feat: add effect image & progress bar
feat: add confuseEffect add Demon Item
feat: add speedDown(banana), invulnerable effect chore: phantom -> shield
feat: pacman buff state pattern -> command pattern make EffectItem
feat: retryGame(), exitGame() add method retryGame(), exitGame() | keyhandler is y pressed, is n pressed
add : 팩맨 사망 사운드 추가
update : 팩맨 사망 시 모든 상호작용 막도록 변경
add : GhostCountTest
fix : Ghost의 count 관리를 StayMode와 호환되도록 수정


add : 아이템 사운드 추가
update : SoundManagerTest 폴더 변경
add: state diagram img
add: gameState
Merge remote-tracking branch 'origin/hogeun'
Merge branch 'sihun'
Merge remote-tracking branch 'origin/yunjae'
feat: make ItemFactoryTest file factory class들이 item을 잘 생성하는지 확인
feat: add buff buff(item)은 state를 이용해서 하나만 활성화. | 현재 무적(phantom)상태시 고스트와 충돌 자체를 무시. => superpa
add : LifeSystem, LifeUI, update: ghost modeTimer 0 초기화 코드 추가
feat: add fruit watermelon: 1000 points | apple: 500 points
feat: add ending condition pacMan이 pacGum을 모두 먹는다면 게임 오버
feat: fix observer, sujet notifyObserverItemEaten, updateItemEaten 추가. | Game에서 pacgum대신 item 랜더링 확인(test).
add : stayMode Junit test
feat: add item & cherry & itemfactory
add : stay state
add : SoundManager JUnit Test class
add : SoundManager, 메인 로직에 Sound 추가
update: 프랑스어 주석 한국어로 수정
add : gitignore
add : PACMAN source code
Initial commit

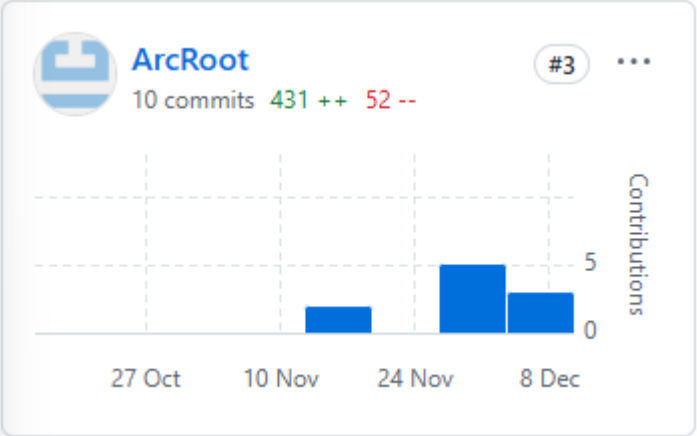
팀원 별 기여

- 양시훈

1. SoundManager
2. GhostCount 시스템
3. Life 시스템, LifeUIPanel
4. 게임 재시작 시스템

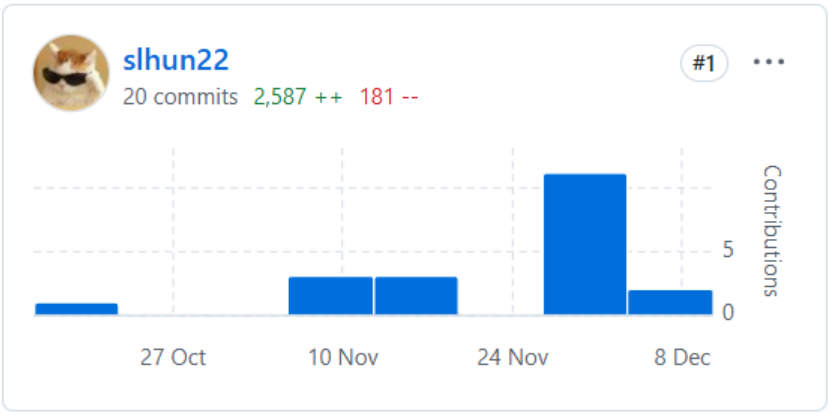


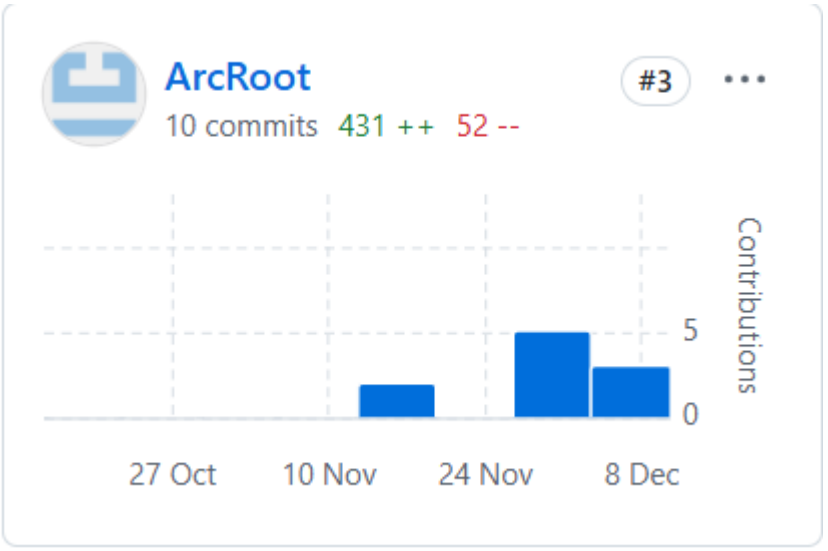
- 박호근
  1. GameOver, GameClear 화면 전환
  2. 게임 종료 시 선택지, 점수 초기화
  3. 레벨 별 ghost 상태 설정, 난이도 조정
  4. 게임 상태 추가 및 변경 조건 설정

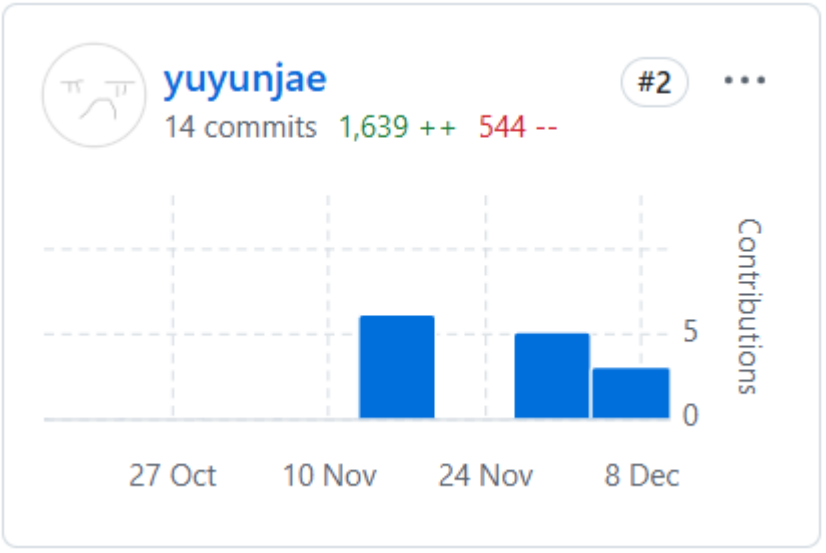


- 유윤재
  1. Item, ItemFactory
  2. MovingEntity(pacman)가 item충돌 시 Effect효과 추가

3. 적용중인 Effect를 보여주는 BuffPanel 생성
4. 난이도 별 Item 수 조절






yuyunjae
#2
14 commits 1,639 ++ 544 --
Contributions
5
0
27 Oct
10 Nov
24 Nov
8 Dec