블록 깨기 과제

목차

설계

해당 프로젝트를 시작하기 전 과제에 필요한 것들이 무엇인지 확인할 필요가 있었다. 해당 과제를 읽어보던 것 중 눈에 띄던 것은 각 클래스로 타이틀화면, 메인화면, 엔딩화면 등을 구성하는 것과, 다양한 사운드들을 넣으라는 것이 신경쓰였다. 해당 자바 구현에서 패널 교체를 요구하는 것은 한번도 해본적 없기 때문에 신경써야 할 것으로 보였다. 또한 JAR파일 구성도 중요했다. JAR파일을 구성하기 위해서는 사운드 이미지등의 모든 외부 파일들은 상대 위치로 입력해주어야 하기 때문에 신경을 써줄 필요가 있었다. 또한 다양한 오브젝트 들이었다. 이 과제에서 가장 핵심적인 내용으로 오브젝트 간의 상호작용을 신경써야 하는데, 예를들어 공 오브젝트와 일반 벽이 부딪혔을 때의 상호작용, 블록이 부딪혔을때의 상호작용, 플레이어 판이 부딪혔을 때의 상호작용등 여러가지를 생각해야하기 때문이다. 마지막으로는 Thread를 이용한 객체의 이동 구현이었는데, Thread를 하나만 사용하는 것이 아니고, 쓰레드를 사용한 패널이 다양하고, 다른 패널로 이동할 때 쓰레드가 정지해야하며, 또 다시 쓰레드가 시작할 때 돌아가야하는 것을 생각하였다.

클래스 전환

먼저 메인 프레임 클래스를 만들어준 후 패널은 크게 타이틀 패널, 엔딩패널, 메인 화면패널을 만들 수 있었다.

```
class titlePanel extends JPanel implements Runnable{
class mainPanel extends JPanel implements KeyListener, Runnable{
class endingPanel extends JPanel implements Runnable{
```

각 패널별로 소개하자면 타이틀 패널의 경우 메인 프레임 클래스에서 처음 불려와 시작하는 패널로 게임명과 타이틀 배경화면, 스페이스바를 누르시오 등의 간단한 메인화면을 출력해줄 것이다. 두번째 메인 패널의 경우 스페이스바를 누를 경우 화면이 전환되어 게임에 시작되는 간단한 객체들이 생성, 업데이트되기 시작하고, 실제적인 게임을 즐길 수 있는 패널이다. 마지막 엔딩패널의 경우 공이 모두 없어졌을 때 생성되며 현재 자신의 스코어와 스페이스바를 눌러 다음 패널을 생성하는 역할을 할 것이다.

타이틀 -> 메인패널

가장 먼저 클래스를 전환하기 위해서 할 것은 스페이스 바를 위한 keylistener를 추가해주는 것이었다. 키리스너를 추가해주어 화면을 전환해주기 위함이었는데, 해당 방식의 경우 메인 프레임에 만들어 줄 수 밖에 없었다. 일반 패널에 만들 경우 변수를 받아와야하는 불편함이 있는데 이것은 간단하게 메인프레임에 만드는 것으로 해결할 수 있었다. 여기서 title의 visible을 false로 전환해준 이후 메인 프레임을 new연산자를 통해 생성자를 불러오고, t.start를 이용해 쓰레드를 시작해준다. 여기서 중요한 것은 이미 시작된 쓰레드를 한번 더 불러올 경우 오류가 나기 때문에 t.isAlive()함수를 이용하여 쓰레드가 시작되고 있는지 확인해줄 필요가 있었다. 이후 프레임에 메인 프레임을 생성해주는 방식으로 add 연산자를 이용해 메인패널을 생성해주면 화면 전환이 완성된다.

여기서 잊지 말아야 할 것이 있는데 key리스너이다 setFocusable을 true로 만들어 준 이후 requestFocus를 이용해 강제적으로 포커스를 가지고 올 수 있다. 난 이것을 이용해 main에서도 똑같이 requestFocus를 생성자에 만들어 주었따. 그러나 포커스가 이동하지 않는다는 것을 깨닳았고, requestfocus가 생성될 때 무시될 가능성도 있다는 것을 알게 되었다. 해당 문제를 고치기 위해서 메인 패널의 쓰레드에서 hasfocus를 이용해 반복문으로 계속해서 focus를 얻어주는 방식을 이용해 주었다.

```
public void keyPressed(KeyEvent e) {
    if(e.getKeyCode()==KeyEvent.VK_SPACE) {
        if(titlep.isVisible()) {
            titlep.setVisible(false);
            mainp = new mainPanel();
            if(!mainp.t.isAlive()) {
                mainp.t.start();
            }
```

메인패널->엔딩패널

메인 패널에서 엔딩패널로 가는 조건은 공이 모두 떨어졌을 때 뿐이다. 이것을 엔딩조건으로 만들어 메인 프레임에서 화면을 전환해 주기 위해서는 메인 프레임에서 쓰레드를 돌리고 있는 방식을 이용하는 것이다. 계속해서 공이 모두 사라졌는지 확인해주는 쓰레드를 돌린 이후 공이 모두 사라졌을 때 화면을 전환해 주는 방식을 이용할 것이

다. 그렇기에 mainp.isOver변수를 만들어준 후 true 가 되었을 경우 엔딩 패널을 새로 만들어 주어 생성해준다. 이때 포커스도 다시 받아와주도록 한다.

```
if(mainp.isOver) {
    mainp.setVisible(false);
endp = new endingPanel(bestScore,Score);
add(endp);
if(hasFocus()) {
    requestFocus();
    if(hasFocus())
        mainp.isOver = false;
}
```

이렇게 화면을 전환해준다. 그런데 이것에서 중요한 것은 mainp는 아직 생성되기 직전이기 때문에 해당 if문이 실행될 수 없다는 점을 생각해주어야 한다. 그렇기 때문에 해당 쓰레드를 스페이스바가 눌렸을 때 생성되는 방식으로 전환해주기로 했다.

```
if(!t.isAlive())
    t.start();
add(mainp);
```

엔딩 패널->타이틀패널
엔딩에서 스페이스바를 누를경우 다시 타이틀 패널로 돌아와야 하기 때문에 엔딩패널이 보이는 중일 경우에는 스페이스바를 누르면 타이틀 패널을 생성해주도록 만들었다,그렇게 해서 완성된 코드는

```
    public void keyPressed(KeyEvent e) {
        if(e.getKeyCode()==KeyEvent.VK_SPACE)
            if(titlep.isVisible()) {
                titlep.setVisible(false);
                mainp = new mainPanel();
                if(!mainp.t.isAlive()) {
                    mainp.t.start();
                }
                if(!t.isAlive())
                    t.start();
                add(mainp);
            }D
            else if(endp.isVisible()){
                endp.setVisible(false);
                titlep = new titlePanel();
                add(titlep);
            }
        }
    }
});
```

이렇게 되었다.

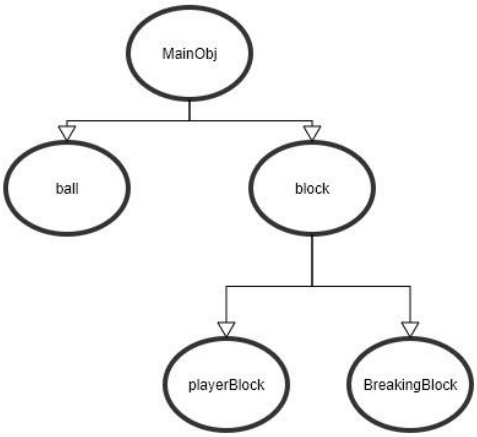
입출력

이제 키 포커스를 이용하여 키를 이동해주어야한다. 가장 먼저 신경 쓴 것은 메인 프레임에서 키 포커스를 가지고 있던 것을 교환해주는 것이었는데 앞에서 말했듯이 한 번에 말을 안 듣는 것 같았다 그렇게 해서 if문을 이용해 프레임을 못 가지고 왔으면 다시 불러보는 방식을 이용했다. 이후 메인 프레임에서의 입출력이었다. 메인에서 신경쓴 것은 좌우를 눌렀을 때 플레이어 블록이 이동하는 것이었는데, 키 리스너가 패널에 있기 때문에 오브젝트 클래스를 상속받은 플레이어 블록을 직접 건드리는 것은 힘들어 보였다. 그래서 플레이어 블록을 찾을 때 까지 반복한 후 플레이어 블록의 경우 속력을 직접 바꿔주는 방식으로 바꾸었다. 또한 해당 키를 떼었을 경우 속력이 다시 0으로 돌아가도록 설정해 주었다.

```
public void keyReleased(KeyEvent e) {
    if(pressKey) {
        pressKey = false;
        for(int i =0; i < obj.size(); i++) {
            if(obj.get(i) instanceof pWall) {
                pWall a = (pWall) obj.get(i);
                a.vx = 0;
                break;
            }
        }
    }
}
public void keyPressed(KeyEvent e) {
    switch(e.getKeyCode()) {
    case KeyEvent.VK_LEFT:
        pressKey = true;
        for(int i = 0; i < obj.size(); i++) {
            if(obj.get(i) instanceof pWall) {
                pWall a = (pWall) obj.get(i);
                a.vx = -200;
                break;
            }
        }
        break;
    case KeyEvent.VK_RIGHT:
        pressKey = true;
        for(int i = 0; i < obj.size(); i++) {
            if(obj.get(i) instanceof pWall) {
                pWall a = (pWall) obj.get(i);
                a.vx = 200;
                break;
            }
        }
        break;
    }
```

해당 방식으로 입력을 설정해 준 이후 만약 공이 다 떨어지면 다시 포커스를 잃도록 하기 위해서 end를 불러주는 프레임에서 쓰레드를 돌리던 중 실패하면 키보드 리스너를 다시 가져오는 방식으로 만들어 줄 수 있었다.

오브젝트 컨트롤



오브젝트를 생성하고 삭제하고 하는 과정을 위해서 모든 객체들을 하나의 obj클래스를 상속받도록 제작해주었다. Obj클래스의 경우 필요한 것은 해당 위치의 시작점, 색이었다. 또한 해당 오브젝트 클래스의 생성자를 제작하여 상속 받도록 만들어주었고, 추상함수로 그리는 함수, 부딪히는 collision함수를 만들어 부딪히게 해주었다. 또한 업데이트를 해야하는 플레이어 블록, 공등을 위해서 업데이트 함수 또한 만들어 주었다. 옆의 그림의 경우 해당 오브젝트가 상속받는 모식도이다. 메인 오브젝트의 경우 공과 블록으로 상속되게 해주었다. 공의 경우 움직이는 업데이트 함수와 반지름을 가진 r, 속력, 과거 자신의 모습등을 변수로 가지고 있다. 또한 블록의 경우 자신의 너비, 높이를 가진 변수를 가지게 해주고 그림을 그리는 함수를 구현해주었다. 해당 블록의 경우 플레이어블록과 부수는 블록으로 또 나눠지도록 설정해주었는데 플레이어블록의 경우 업데이트 함수를 추가해주어 블록이 키보드 컨트롤에 맞추어 움직일 수 있도록 구현해주었고 브레이킹 블록의 경우 해당 블록이 부숴지도록 따로 클래스를 만들어 둔 것 뿐이다. 해당 오브젝트들을 이용하여 직접 생성을 해야했다.

튕기기

먼저 공을 움직이고 해당 공이 튕기는 콜리전 함수를 구현해줄 필요가 있었다. 해당 함수를 구현하기 위해서 쓰레드를 구현해주었다. 쓰레드를 만들어 준 이후 해당 쓰레드에서 먼저 오브젝트가 움직이도록 업데이트 함수를 불러오도록 했다.concurrent오류가 나지 않도록 설정해주기 위해서 foreach문 을 사용하지 않고 조금 아쉽더라도for 반복문을 이용해주었다. 오브젝트를 업데이트 해주는 함수의 경우ball의 경우 강의에서 봤던대로 속도를 정해준후 각도를 랜덤 설정해주어 속도를 업데이트 해 준 이후 해당 속도를 위치에 더하는 방식으로 설정해주었고 block의 경우 위에서 말한 방식으로 설정해주었다. 이후 콜리전 함수인데 먼저 튕겨지는 것의 경우 강의를 따라해

```
for (int i = 0; i<obj.size(); i++) {
    if(!(obj.get(i) instanceof gBall))continue; //이게 공이면 충돌
    gBall gball = (gBall)obj.get(i);
    for(int j = 0; j<obj.size(); j++) {
        if(!(obj.get(j) instanceof gWall))continue; {
        gWall gwall = (gWall)obj.get(j);

            if(gball.isCollide(gwall)){
                gball.collision(gwall);
                if(gwall instanceof bWall) {
                    obj.remove(j);
                    score += 10;
                    if(gwall.c.equals(Color.decode("#B5E61D"))) {
                        gBall clone = addClone(gball);
                        obj.add(clone);
                        balls++;
                        obj.add(addClone(clone));
                        balls++;
                        itemBlock.setFramePosition(0);
                        itemBlock.start();
                    }
                    breakBlock.setFramePosition(0);
                    breakBlock.start();
                    blocks--;
                }
                if(gwall instanceof pWall) {
                    balltouch.setFramePosition(0);
                    balltouch.start();
                }
            }
        }
    }
```

instance연산자를 이용해 벽에 튕기도록 구현해주었고, 일반 벽은 block함수를 이용해주었으나, 부수는 벽과 플레이어의 벽 모두 block에서 상속받아왔기 때문에 자동으로 구현되는 것을 알 수 있다. 이후 instanceof연산자를 이용하여 부수는 벽의 경우 실제 오브젝트가 사라지도록 설정하여 주었다. 또한 해당 벽이 생성될 때 랜덤으로 색을 설정해주도록 만들었는데, 해당 색이 gray(기본 부수는 블록)이 아닌 다른 색의 경우 clone함수를 만들어 주었다.

벽돌소환 및 깨기

벽돌을 소환하기 위해서 따로 벽돌 소환 함수를 구현해주었다.

```
void makeBlock() {
    int wblock = (stage*2) + 3;
    int hblock = (stage*2) + 3;
    int blocksH = 200 + (stage * 50);
    for(int i= 0; i < hblock; i++) {
        for(int j = 0; j <wblock; j++) {
            int y = (int) (Math.random() * 2);
            if(y == 0) {
                obj.add(new bWall((getWidth() - 20)/wblock * j + 12, ((blocksH-10)/hblock) * i + 11,
                        (getWidth()-20)/wblock - 1, (blocksH-10)/hblock - 1, Color.gray));
            }
            else if(y == 1) {
                obj.add(new bWall((getWidth() - 20)/wblock * j + 12, ((blocksH-10)/hblock) * i + 11,
                        (getWidth()-20)/wblock - 1, (blocksH-10)/hblock - 1, Color.decode("#B5E61D")));
            }
            blocks++;
        }
    }
}
```

위의 방식으로 벽돌 소환을 구현할 수 있었는데 살펴보면, 스테이지 수에 따라 벽돌 개수를 설정해준 후 2/1확률으로 벽돌이 아이템일지 일반블록일지 설정해주었다. 일반벽돌의 경우 색의 차이를 주었고, 색이 다를 경우 위에서 나누는 함수를 불러오도록 설정해주었다. 블록 부수기의 경우 위의 충돌 함수를 보면 알 수 있는데, 충돌하여 해당 블록이 bWall즉 블록을 부수기위해 만든 블록일 경우 부숴지도록 설정해 두었다.

아이템 블록의 경우 아래의 방식으로 블록을 속력을 설정해주었다.

```
gBall addClone(gBall g) {
    gBall r= new gBall(g.x,g.y,g.r,g.c);
    r.vx = g.vx*1.3;
    if(r.vx >= 450) {
        r.vx -= 450;
    }
    if(r.vx < 0) {
        r.vx = 450 + r.vx;
    }
    r.vy = Math.sqrt(Math.pow(g.speed, 2) - Math.pow(r.vx, 2));
    r.pre_X = r.x;
    r.pre_Y = r.y;
    //System.out.println("clone info x:" + r.x + ", y :" + r.y + ", vx :"+ r.vx + ", vy:"+ r.vy);
    return r;
}
```

X의 속력을 올려주고 해당 속력을 이용하여 vy속력을 이끌어내는 방식을 이용해주었다.
블록을 모두 부쉈을 경우 다음스테이지로 이동하는데 공은 다시 초기화되어야하기 때문에 다시 모두 삭제후 만들어주는 과정을 거쳐주었다.

```
if(blocks == 0) {
    if(stage != 0) {
        balls = 0;
        obj.clear();
        obj.add(new pWall(200, 900, 100, 30, Color.blue));
        obj.add(new gWall(0,0,10,1000, Color.orange));
        obj.add(new gWall(575,0,10,1000, Color.orange));
        obj.add(new gWall(10,0,575,10,Color.orange));
        obj.add(new gBall(250, 850, 6, Color.red));
        balls++;
        stageClear.setFramePosition(0);
        stageClear.start();
    }
    makeBlock();
    stage++;
}
```

스코어

스코어 계산을 위해 메인 패널에서 블록을 깰때마다 계산해주고 해당 수를 메인 프레임에서 받아올 수 있도록 설정해주었다. 그러나 해당 스코어 계산 방식의 경우 있는 문제 단 하나는 엔딩 패널에 옮길 수 없었다는 점이었는데 해당 방식을 해결해주기 위해서 엔딩패널의 생성자에 스코어를 담아 보내는 방식을 이용해주었다.

사운드 입히기

내가 비쥬얼적으로 가장 신경쓴 점은 다양한 사운드에 있다. 먼저 메인 게임에서 플레이 할 때 배경음악을 loop를 이용해서 깔아주었고, 블록이 부숴지는 소리 플레이어가 튕겨내는 소리 게임이 시작하는 소리 끝나는 소리 엔딩 아이템 블록을 사용했을 때 나는 소리 다음 스테이지로 이동할 때 나는 소리등 다양한 소리와 효과음 들을 구현하였다. 그런데 해당 소리들의 경우 한번 나면 다시 나지 않는 문제가 발생하였는데 해당 문제를 수정하기 위해서 프레임을 0으로 고쳐주고 다시 만들어주었다. 이미지 또한 만들어주었는데 배경이미지만 덧씌워주었다.

```
try {
    background = AudioSystem.getClip();
    breakBlock = AudioSystem.getClip();
    itemBlock = AudioSystem.getClip();
    balltouch = AudioSystem.getClip();
    next = AudioSystem.getClip();
    fail = AudioSystem.getClip();
    stageClear = AudioSystem.getClip();
    URL url1 = getClass().getClassLoader().getResource("videoplayback.wav");
    URL url2 = getClass().getClassLoader().getResource("blockBreak.wav");
    URL url3 = getClass().getClassLoader().getResource("itemBlock.wav");
    URL url4 = getClass().getClassLoader().getResource("balltouch.wav");
    URL url5 = getClass().getClassLoader().getResource("next.wav");
    URL url6 = getClass().getClassLoader().getResource("fail.wav");
    URL url7 = getClass().getClassLoader().getResource("stageClear.wav");

    AudioInputStream backstream =
            AudioSystem.getAudioInputStream(url1);
    AudioInputStream breakstream =
            AudioSystem.getAudioInputStream(url2);
    AudioInputStream itemstream =
            AudioSystem.getAudioInputStream(url3);
    AudioInputStream touchstream =
            AudioSystem.getAudioInputStream(url4);
    AudioInputStream nextstream =
            AudioSystem.getAudioInputStream(url5);
    AudioInputStream failstream =
            AudioSystem.getAudioInputStream(url6);
    AudioInputStream clearstream =
            AudioSystem.getAudioInputStream(url7);
    background.open(backstream);
    breakBlock.open(breakstream);
    itemBlock.open(itemstream);
    balltouch.open(touchstream);
    next.open(nextstream);
    fail.open(failstream);
    stageClear.open(clearstream);
} catch (LineUnavailableException e) {
    e.printStackTrace();
} catch (IOException | UnsupportedAudioFileException e) {
    e.printStackTrace();
```

반복작업

해당 방식은 계속 반복되어 일어나야한다. 그렇기 때문에 계속해서 다시 만들어주기 위해서 메인 패널에서 부숴졌을 경우 스코어 초기화, 배경음악 정지, 끝난것을 알리는 bool값 설정 오브젝트들 모두 삭제, 스테이지수 초기화 등의 작업을 이루었다,

```
if(gball.y > 1000) {
    obj.remove(i);
    balls--;
    if(balls == 0) {
        background.stop();
        background.setFramePosition(0);
        fail.setFramePosition(0);
        fail.start();
        isOver = true;
        stage = 0;
        obj.clear();
    }
}
```

또한 해당 작업을 위하여 모든 패널의 이동에서 new연산자로 만들어 이동해주었고 이것으로 인해 새로 생성되어 쓰레드가 다시 작동하여 끝나도 재시작 될 수 있도록 설계해주었다.

어려웠던 점

문제점이 발견되었따 가끔 시작할 때 블록이 보이지 않는 오류가 있었으나 이것은 실행시 패널의 크기가 정해지지 않은 상태에서 블록이 생성되다보니 생긴 오류로 아예 블록의 패널의 크기를 가져오는 getwidth().가 아닌 585의 크기를 가지고 완성해주었다.

쓰레드와 쓰레드 간의 연결과 키보드 리스너를 변경하는 점이 힘들었고, 패널이 교환되는 것 또한 여간 어려운 것이 아니었다. 그러나 페인트 컴포넌트에서 교환해주는 방식, 패널에서 교환해주는 방식등 다양하게 시도해본 결과 해당 방식을 도출해낸 것이 뿌듯했다.