Block breaker 개발 과정

Goal 1. 타이틀 화면을 보여주고 사용자 입력을 기다림

일단 타이틀화면/게임화면/게임오버 화면의 세가지 화면으로 구성하고 각각 다른 클래스로 만들어야 하므로 타이틀 클래스를 만들어 주기로 했다.

`class Title`

`class PlayScreen`

`class GameOverScreen`

의 세가지 클래스를 만들어서 키를 입력하면 다음 화면으로 넘어가게 만들어야 했다.

그런데 세가지 클래스를 어떻게 해야 다음 화면으로 넘어가게 만들지 걱정이였다. 메인에서 키 입력을 받을 경우에는 다음 화면으로 넘어 갔을 때도 계속 받는 문제가 있어서 각각 클래스에서 어떤 조건을 달성하면 메인 문에서 다른 화면을 출력해줘야 겠다는 생각을 했다.

각 각의 클래스에 스레드를 주고 키가 입력 될 때까지 반복하다가 스레드가 종료되면 다음 화면을 출력하는 방식으로 생각하다가 너무 복잡하고 만들기 어려울 것 같아 포기하고 다른 방법을 생각했는데 비교적 간단한 방법으로 해결 할 수 있었다.

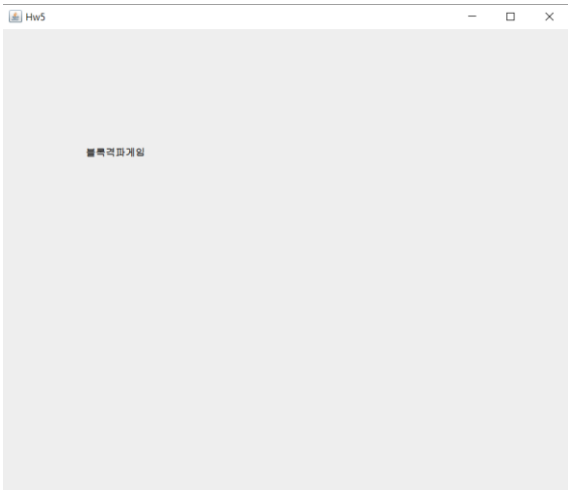
각각의 클래스에 KeyListener를 구현해서 스페이스 바를 누르면 메인 문에서 만들어준 함수로 실행한 뒤, 없어질 클래스에서 setVisible(false)를 해주었다.

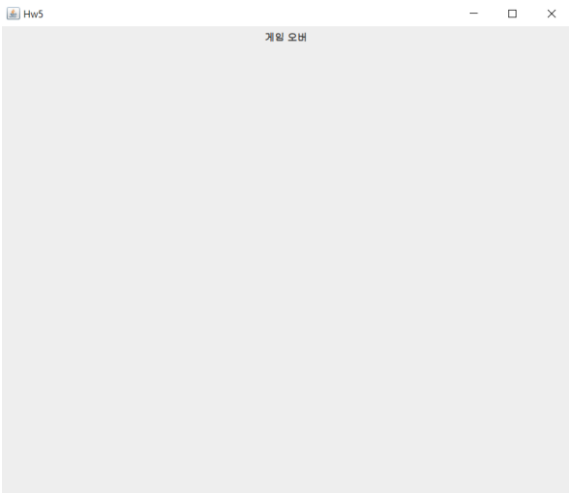
```
void ScreenChange(int screen)
{
        if(screen==0)
        {
            add(new Title());
        }

        else if(screen==1)
        {
            add(new PlayScreen());
        }
        else if(screen==2)
        {
            add(new GameOverScreen());
        }

        setVisible(true);
}
```

(메인 클래스에 만든 이 함수를)

```java
@Override
public void keyPressed(KeyEvent e) {
    if(e.getKeyCode()==KeyEvent.VK_SPACE)
    {
        ScreenChange(0);
        this.setVisible(false);
    }
}
```

(각 클래스에서 다음처럼 키를 입력 받아 바꿔 줄 수 있었다.)





(꾸미지 않아 투박하지만 타이틀 화면 구성과 화면이 넘어가는 것을 구현 했다.)

창을 변경하지 못하게 setResizable(false); 라는 메소드를 인터넷에서 찾아 사용해주었다.

Goal 2. 공이 화면에 나타나고 라켓과 벽면에 충돌하면 튕겨나감. 공이 화면 밖으로 나가면 게임오버 화면이 표시됨.

그 다음으로는 라켓과 공을 구현해야 했다.

키보드 입력을 받아야하고 스레드를 사용해야하므로

```java
class PlayScreen extends JPanel implements KeyListener, Runnable
```

로 정의를 했고, 라켓과 공을 그려야 했으므로 `public void paintComponent(Graphics g)`를 사용해야겠다고 생각했다.

먼저 라켓부터 구현을 시작했는데, 처음에는 라켓과 공을 JPanel의 클래스로 만들고 그안에 내용을 적었는데 만들다보니 굳이 클래스로 만들지 않고 PlayScreen 클래스 내의 paintComponent 함수 안에서 만들어주면 되는 것을 깨달았다.

barX(라켓의 X위치), barY(라켓의 Y위치- 고정값), barW(라켓의 넓이 – 고정값), barH(라켓의 높이 – 고정값) 의 네 가지 변수를 선언했고,

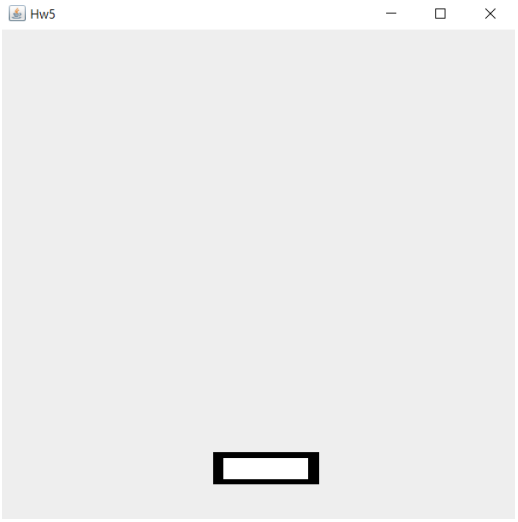
```
g.setColor(Color.BLACK);
g.fillRect(barX,barY,barW,barH);
g.setColor(Color.WHITE);
g.fillRect(barX+10, barY+5, barW-20, barH-10);
```

로 라켓을 그려줄 수 있었다.

또한 윈도우 창을 W,H의 두 개의 변수에 담아 라켓의 초기 위치를

```
barX=W/2-(barW/2);                    //바 x위치
```

위처럼 설정해



다음과 같은 조촐한 라켓을 만들어 줄 수 있었다.

그 다음으로는 왼쪽과 오른쪽 방향키를 눌러 줄 때 라켓이 이동하고, 벽에 닿을 때는 이동하지 않게 설정하는 작업이 필요했다.

```
public void keyPressed(KeyEvent e) {
        if(e.getKeyCode()==KeyEvent.VK_LEFT)
        {
                if(barX>=1)
                {
                        barX-=15;
                }
                else
                {
                        barX=1;
                }
        }

        if(e.getKeyCode()==KeyEvent.VK_RIGHT)
        {
                if(barX<W-barW)
                {
```

```
                barX+=15;
                }
                else
                {
                        barX=W-barW;
                }
        }
}
```

로 미리 선언해둔 라켓의 X위치를 바꿔주는 방식을 사용했다

문제점 1.

여기서는 barX가 1씩 움직이지 않아서 벽에 닿았을 때 진동하듯 밖으로 나갔다 들어오는 현상이 있었다.

```
        if(e.getKeyCode()==KeyEvent.VK_LEFT)
        {
                if(barX-15>=1)
                {
                        barX-=15;
                }
                else
                {
                        barX=1;
                }
        }

        if(e.getKeyCode()==KeyEvent.VK_RIGHT)
        {
                if(barX+30<=W-barW)
                {
                barX+=15;
                }
                else
                {
                        barX=this.getWidth()-barW;
                }
        }
```

위처럼 범위를 조정해주어서 문제를 해결 할 수 있었다.

그 뒤로는 공을 구현 하기에 앞서 공도 라켓처럼

ballX(공의 X위치),ballY(공의 Y위치), ballW(공의 넓이 – 고정값), ballH(공의 높이 – 고정값)

의 네 가지 변수를 선언했고 추가적으로 공이 이동하는 정도를 나타내는 dx,dy 변수도 선언해주고 초기에는 y값만 변하므로 dy=10으로 두었다.

스레드 안에는 resolveCollision(),update(),repaint() 메소드를 만들어 두고 공이 아래로 떨어지면 게임이 끝나야하므로 Boolean GameOver을 만들어서 while문에 넣어주었다.

그 뒤에는 라켓과 공이, 공과 벽면이 충돌하는 내용을 담은 resolveCollision 함수를 만들어주었는데

```
if(ballX>W-ballW)     { ballX = W-ballW; dx = -dx;}
if(ballX<=0)          { ballX = 0; dx = -dx;}
if(ballY>H)           { GameOver=true; }
if(ballY<=0)          { ballY =0;  dy = -dy;}
```

다음과 같이 네 가지 벽면에 대한 충돌을 검사후 dx와 dy의 부호를 바꿔주었다.

그 뒤로 공이 라켓과 닿을 때는

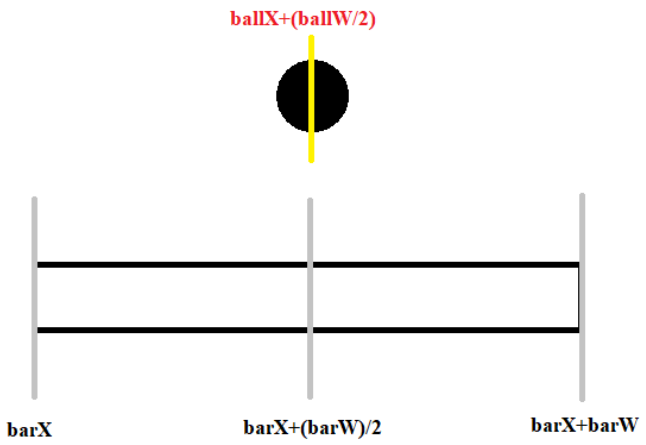
```
if(ballX>=barX-15 && ballX+ballW<=barX+barW+15 && ballY+ballH==barY) { dy=-dy; }
```

으로 공이 다시 위로 튕기게 만들어 주었다.

문제점 2. 공이 튀길 때 dx는 어떻게 변화해야 하는가?

예제 파일을 보면 라켓의 중앙을 기준으로 라켓의 가장자리 쪽으로 갈수록 공의 x좌표 위치변화가 큰 것을 알 수 있다.

따라서 공이 부딪히는 위치를 기준으로 dx값을 바꿔줘야 하겠다는 생각을 했다.



위와 같은 상황에 barX와 ballX의 좌표 값을 사용하면 좌표가 커지거나 작아질수록 dx가 변하므로 X위치를 빼주고 남은 길이를 가지고 dx를 정하기로 했다.

```
dx=((ballX+(ballW/2))-(barX+(barW/2)))/10;
```

위 처럼 ballX-barX가 되면 라켓의 중앙을 기준으로 +와 -가 정해지게 되고 중앙에서 멀어질수록 더 크게 움직이도록 만들었다. (10은 임의로 나눠준 수 이다.)

문제점 3. 공이 라켓의 왼쪽이나 아래 쪽을 맞아도 위로 올라가는 문제

```
b.dx=((b.bx+(b.bw/2))-(barX+(barW/2)))/10;

if(b.by+b.bh<barY+(barH/2))
b.dy=-b.dy;
```

공은 항상 위에서 떨어지므로 공이 라켓의 중앙 보다 위에 있으면 다시 튕기고 아니라면 그대로 떨어지게 만들었다. (게임을 쉽게 하기 위해 걸쳐 맞아도 위로 올라가도록 만듦)

그리고 공이 밖으로 나가면 미리 만들어둔 충돌 검사 함수에서

```
if(ballY>H)          { GameOver=true; }
```

가 되어

```
public void run() {
        while(!GameOver) {
                resolveCollision();
                update();
                repaint();
        }

        ScreenChange(2);
        this.setVisible(false);
}
```

와 같이 스레드에서 while문을 벗어나 게임 오버 화면으로 전환이 되게 구현했다.

문제점 4. 공과 벽돌의 충돌이 불안정함.

추가적으로 공과 벽돌의 충돌이 불안정하다는 문제점이 있었는데, 벽돌의 위쪽을 맞거나 양옆을 맞을 때 dx와 dy가 잘 변하지 않는 문제가 있었다. 구현을 하기 위해서 기존에 쓰던 Balls 의 bw 와 bh 대신 반지름 변수 r과 이전의 x,y 좌표인 prex와 prey를 만들어서 구현했다.

```
if(b.blockcollision==false)
{
        if(b.prex <= x1-2) {b.bx = b.prex; b.dx = -Math.abs(b.dx);}
        if(b.prex >= x2+2) {b.bx = b.prex; b.dx = Math.abs(b.dx);}
        if(b.prey <= y1-2) {b.by = b.prey; b.dy = -Math.abs(b.dy);}
        if(b.prey >= y2+2) {b.by = b.prey; b.dy = Math.abs(b.dy);}

        b.blockcollision=true;
}
```

(x1-2 등으로 추가적인 값을 빼주었는데 공이 블록이랑 살짝 겹쳐보이는 경우가 있어서 임의적으로 충돌 검사 범위를 늘려주었다.)

Goal 3. 블록이 화면에 나타나며 공을 조작해 블록을 없앨 수 있음, 블록 중 특별한 블록은 없어지며 공을 3개로 만듦.

일단 블록은 여러 개이므로 따로 객체를 만들어서 여러 개를 한번에 그려야 한다고 생각했다. 그래서 `class Blocks` 을 만들어 준 뒤 배열 형식으로 구현 하였다.

블록은 행과 열로 정렬 되어있으므로 이차원 배열을 사용해서 만들어 주었고,

```
int blockx;                    // 블록의 x좌표 위치
int blocky;                    // 블록의 y좌표 위치
int blockw;                    // 블록의 가로 길이
int blockh;                    // 블록의 세로 길이
int blocks;                    // 블록의 종류
```

의 다섯가지 변수를 선언했다.

그리고 생성자에서 `blocks=(int)(Math.random()*10+1);` 로 sort값을 (1~10)으로 정해두었는데 이는 이벤트 블록이 랜덤하게 생성되야 하므로 랜덤으로 값을 구해준 값이다.

그리고 class Blocks 내에서

```
void setblock(int x,int y,int w,int h)
{
	blockx=x;
	blocky=y;
	blockw=w;
	blockh=h;
}
```

다음과 같이 블록의 값들을 정해주는 함수를 만든 뒤, 게임 플레이스크린 클래스 내에서

```
public Blocks[][] block= new Blocks[blockrow][blockcol];

void makeblock()
{
	for(int i=0;i<blockrow;i++)
	{
		for(int j=0;j<blockcol;j++)
		{
		block[i][j]=new Blocks();
		block[i][j].setblock((i*bw)+blockspace*i,bh*j+blockspace*j,bw,bh);
		}
	}
}
```

다음과 같이 만들어 주었다.

Bw와 bh는

```
int bw=(W-blockspace*(blockrow-1))/blockrow;
int bh=H/15;
```

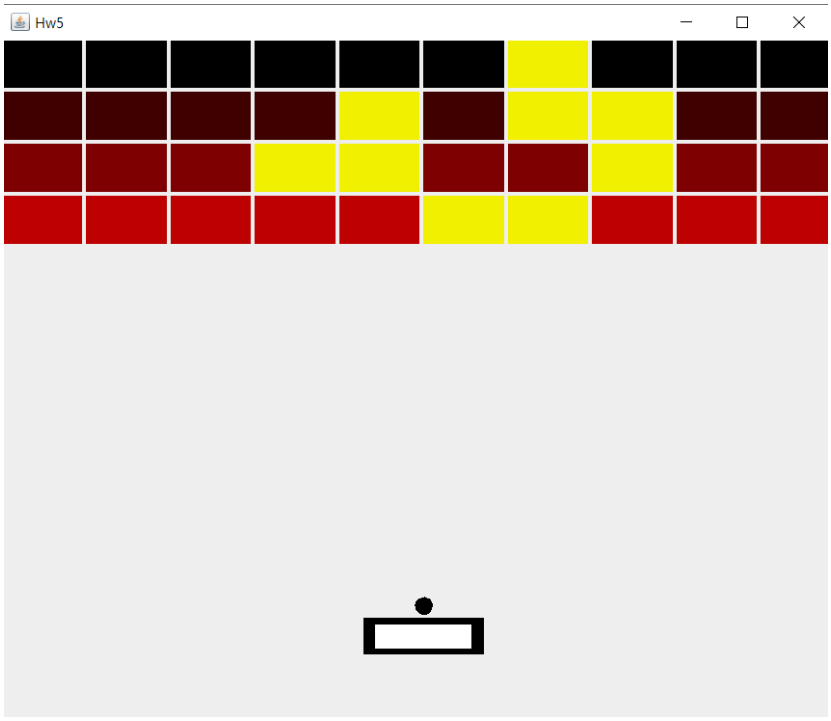
로 윈도우 창(W,H)과 블록의 개수에 맞춰서 만들었고 (blockrow , blockcol 는 블록의 행과 열)

Block의 x와 y좌표에 따라 달라지게 만들어주었고, blockspace는 블록 사이의 값으로 임의적으로 3으로 두었다.

그 뒤로는 PlayScreen 클래스 내의 PaintComponent 함수에서

```
for(int i=0;i<blockrow;i++)
{
    for(int j=0;j<blockcol;j++)
    {
        if(block[i][j].blocks<=7)                    //이벤트 블럭
        {
            g.setColor(new Color(j*(255/blockcol),0,0));
        }
        else
            g.setColor(new Color(240,240,0));
        g.fillRect(block[i][j].blockx,block[i][j].blocky,block[i][j].blockw,block[i][j].blockh);
    }
}
```

다음과 같이 출력했는데 이벤트 블록의 sort 값이 8~10 인경우, 즉 30%의 확률로 이벤트 블록이 생성되고 노란색 블록으로 출력되게 만들었고 나머지 블록은 열과 비례해서 색이 바뀌도록 설정했다.



(블록을 생성한 화면)

그 뒤로는 블록에 공이 부딪힐 때 파괴되고 공을 튀겨내게 만들어야 했는데

for문을 두 번 돌려서 공이 바에 부딪히는 것처럼 블록에 부딪히면 반사되고 블록이 없어지게 만들면 된다고 생각했다.

다만 바와 다른 점은 블록의 양 옆과 위 아래를 맞을 때 dx dy 바뀌는 것이 다르기 때문에 범위를 나누어서 처리를 해야 했다.

문제점 4.

두 블록이 동시에 맞을 때는 dx dy가 두 번 연속 바뀌어 그대로 공이 진행 될 수 있으므로 두 블록이 동시에 맞는 상황을 처리해야 했다.

Boolean `blockcollision = false;`

를 만들어서 `blockcollision`이 `false`일 때만 `dx dy` 값을 바꿔주되, 한번 값을 바꾸면 `blockcollision = true;` 로 값을 바꿔주고 다음 충돌 함수가 실행될 때 다시 `blockcollision = false`로 만들어 주었다.

그리고 `int count` 변수를 두어 충돌 할 때마다 `count`를 올려주었는데

`Count = blockcol*blockrow` 일 때는 스테이지를 클리어 한 것 이므로 `GameOver=true`로 만들어주고

초기값을 1로둔 `int stage`에 맞춰 게임을 초기화 하는 `init` 함수를 만들었다.

```
void init()
{
    ballList = new LinkedList<Balls>();

    barX=W/2-(barW/2);                                                    //바 x위치
    ballList.add(new Balls(W/2-(ballW/2),(H/9)*7,ballW,ballH,4,stage+2,bindex));

    blockrow=2+stage*3;
    blockcol=stage+2;

    bw=(W-blockspace*(blockrow-1))/blockrow;
    bh=(H/3)/blockcol;

    block = new Blocks[blockrow][blockcol];
    ballcount=1;
    count = 0;

    GameOver = false;

    makeblock();
}
```

`Init` 의 내용은 이러한데, 먼저 공들과 리스트를 초기화하고 라켓과 공의 위치를 초기화했다. 그 뒤로는 스테이지에 비례해서 블록의 개수가 증가하고 공의 속도 또한 스테이지에 비례해서

점점 빨라진다. 또한 공이 빨라지면 기존 속도로는 라켓으로 공을 받기 어렵기 때문에 라켓의 속도도 stage에 따라 비례해서 증가하도록 했다. Stage가 무한히 늘어나므로 게임은 게임오버 되기 전까지 계속 진행 된다.

그 뒤로는 공이 이벤트 블록에 부딪혔을 때 세 개로 갈라지는 기능을 만들어야 했는데 이벤트 블록은 랜덤이고 공이 생기고 없어지는 수를 세기가 굉장히 어렵다고 생각했다.

많은 시간을 고민하고 찾아본 결과 LinkedList를 사용해서

`class Balls` 과 `LinkedList<Balls> ballList;` 를 사용해서 만들기로 했다.

Class Balls에는 기존에 공을 위해서 만들었던 변수들을 모두 넣어

```
int index;                                  // 공의 인덱스

int bx;                                     // 공의 x좌표
int by;                                     // 공의 y좌표
int bw;                                     // 공의 가로
int bh;                                     // 공의 세로

int dx;                                     // 공의 x변화량
int dy;                                     // 공의 y변화량

boolean blockcollision=false;               // 블록에 두 개이상 겹침을 검사
```

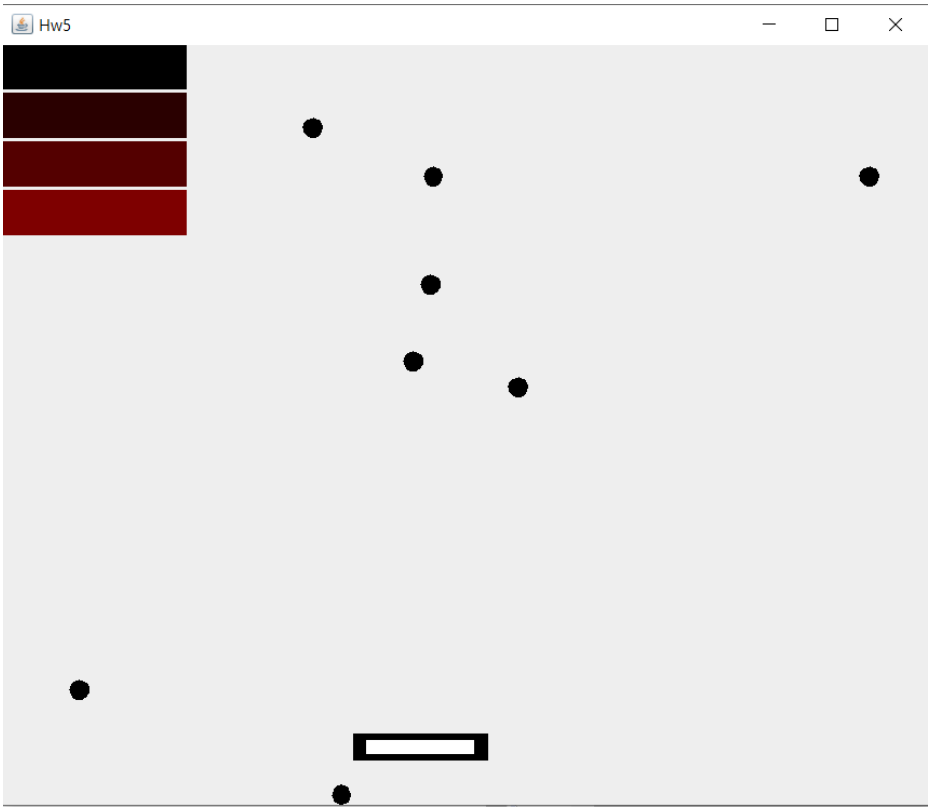
로 만들어 주었고,

PaintComponent와 resolveCollision에서는 for-each문을 사용해서 구현을 했다.

그리고 이벤트 블록에 닿을 때 실행 할 함수로

```
void blockevent(int x,int y)
{
        ballList.add(new Balls(x,y,ballW,ballH,-1,3,bindex+1));
        ballList.add(new Balls(x,y,ballW,ballH,0,3,bindex+2));
        ballList.add(new Balls(x,y,ballW,ballH,1,3,bindex+3));

        ballcount+=3;
        bindex+=3;
}
```

를 만들었는데, 스페이스바를 눌렀을 때 이 기능을 실행하게 해보니 공이 그려지는 것, 공들과 벽들의 충돌, 공들과 벽돌의 충돌 모두 정상적으로 이루어졌다.



문제점 5.

그런데 이벤트 블록에 공이 닿았을 때 이 함수를 실행하면

```
Exception in thread "Thread-0" java.util.ConcurrentModificationException
        at java.base/java.util.LinkedList$ListItr.checkForComodification(Linl
        at java.base/java.util.LinkedList$ListItr.next(LinkedList.java:892)
        at Hw5$PlayScreen.resolveCollision(Hw5.java:136)
        at Hw5$PlayScreen.run(Hw5.java:124)
        at java.base/java.lang.Thread.run(Thread.java:835)
```

다음과 같은 에러가 발생하면서 실행이 멈추는 문제가 생겼다. 바로 for-each문에서 값을 추가하는 식으로 구현했기 때문이었는데, 한참 해결을 못하다가 이터레이터를 사용해서 해결해야 함을 알았다. 그런데 이터레이터 사용이 익숙하지 않은 탓에 재대로 사용을 하지 못하는 건지 이터레이터로 벽돌 충돌 검사를 하면 bx와 by를 제대로 받음에도 충돌 검사를 제대로 하지 못하는 문제가 있었다.

그래서 차라리 다른 방법으로 해결을 했는데, 정수형 배열 event와 event의 현재 위치를 저장할 eventindex를 선언했다. 그 뒤로 blockcol*blockrow 만큼의 충돌 검사 중 이벤트 블록에 충돌했을 시 에는 event 배열에 블록의 중앙에 해당 하는

```
int ex=block[i][j].blockx+((block[i][j].blockw)/2);
```

```
int ey=block[i][j].blocky+((block[i][j].blockh)/2);
```

값을 넣어주었다.

```
event[eventindex]=ex;
event[eventindex+1]=ey;
eventindex= eventindex+2;
```

(이런 방식이다.)

그 뒤로 이벤트 블록의 충돌을 뜻하는 Boolean eventcheck 을 false에서 true로 바꾸어 준 뒤, 모든 벽돌의 충돌 검사가 끝났을 때

```
if(eventcheck==true)
{
	for(int i=0;i<ballcount*2;i+=2)
	{
		if(event[i]!=0)
		{
			blockevent(event[i],event[i+1]);
		}

		else
		{
			break;
		}
	}

	eventcheck=false;
}
```

다음과 같은 기능으로 for-each문 밖에서 blockevent를 실행 시켜주게 만들었더니 잘 구현 되었다.

문제점 6. 벽돌 카운트의 문제

벽돌을 카운트 할 때 마다 count를 증가시켜서 blockcol * blockrow 값이 되면 끝나게 구현하니까 벽돌이 다 깨지기 전에 게임이 끝나는 경우가 있었다. 충돌 시 벽돌 값과 위치를 0으로 초기화 하지만 필요 이상으로 count가 되는 것 같았다. 따라서 정확하게 구현 하기 위해서는 block에서 벽돌 파괴 상태를 뜻하는 Boolean broken을 만들어 주고 모든 블록이 broken 이면 다음 스테이지로 넘어가게 만들어야 한다고 생각했다. 만약 이렇게 만들어 준다면 충돌검사와 draw를 할 때도 if문으로 broken=false 일 때만 실행하면 되게 조건을 달 수 있다는 장점이 있다고 생각했다. 단점으로는 검사를 한번 더 해야 한다는 단점이 있었다.

```
if(block[i][j].broken==false)
{
    count++;
    score+=30;
    block[i][j].broken=true;
}
```

블록 충돌 검사에서 위를 추가 해서 성공적으로 구현됨을 확인 할 수 있었다.

문제점 7. 스테이지 넘어가기 전 마지막 벽돌이 이벤트 블록인 경우.

스테이지 넘어가기 전 마지막 블록이 이벤트 블록인 경우 다음 스테이지에서 시작 하자마자 그 위치에서 이벤트가 발생하는 문제가 있었다. 처음에는 BallList가 삭제가 안되어서 그런 줄 알고 Clear 해주었으나 해결되지 않았다.

계속 찾아보다가 충돌 검사를 하면서 게임이 끝난 후에 event 함수가 실행되는 것을 발견해서

```
if(ballcount<=0)
{
	soundStop();
	GameOverSound.setFramePosition(0);
	GameOverSound.start();
	GameOver=true;
	stage=-1;

	try {
		Thread.sleep(1500);
	} catch (InterruptedException e) {
		e.printStackTrace();
	}

	ballList.clear();

	if(maxscore<score)
		maxscore=score;


	return;
}

if(count>=blockcol*blockrow)
{
	soundStop();
	stage++;
	GameOver=true;
	score+=1000;
	ballList.clear();
	Clear.setFramePosition(0);
	Clear.start();

	try {
		Thread.sleep(1500);
	} catch (InterruptedException e) {
		e.printStackTrace();
	}


	init();
	return;
```

```
}

if(eventcheck==true && GameOver==false)
{
	Event.setFramePosition(0);
	Event.start();

	for(int i=0;i<ballcount*2;i+=2)
	{
		if(event[i]!=0)
		{
			blockevent(event[i],event[i+1]);
		}

		else
		{
			break;
		}
	}

	eventcheck=false;
}
```

다음과 같이 게임 오버가 되거나 다음 스테이지로 넘어가게 되면 return 해주는 방식으로 해결할 수 있었다.

Goal 4. 그림이나 사운드 및 여러 시도를 통해 비쥬얼 및 게임성을 높임

이제 게임 기능적으로는 어느 정도 구현이 되었으니 그림이나 사운드 사용을 할 차례였다. 먼저 사운드 기능부터 만들기로 했는데, 사운드 기능을 구현 해야겠다고 생각했다.

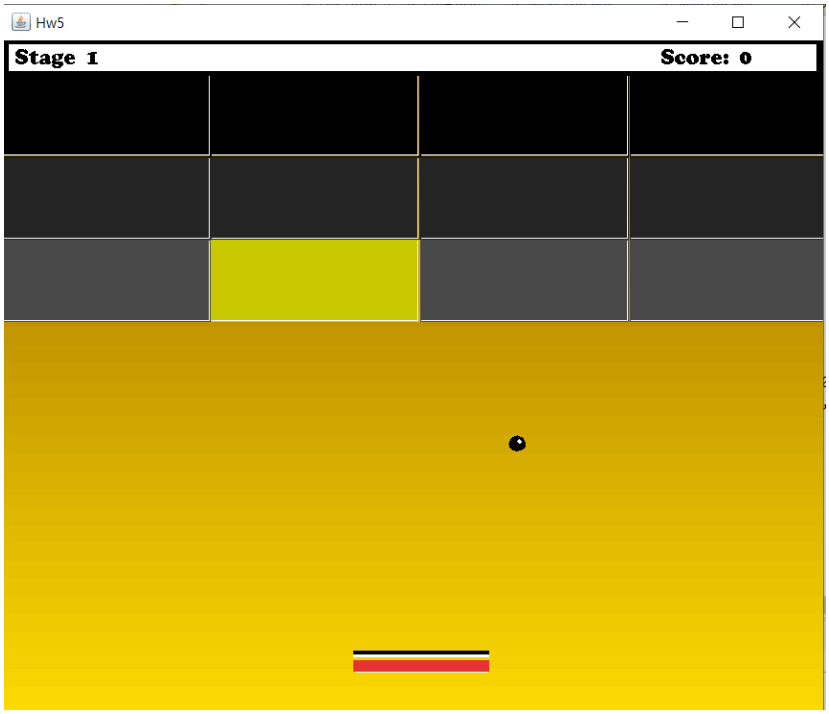
```
Clip PlayBGM1;        // 게임BGM1
Clip PlayBGM2;        // 게임BGM2
Clip PlayBGM3;        // 게임BGM3
Clip bounce;          // 공이 튀길 때 나는 효과음
Clip Broken;          // 일반 벽 부숴질 때 효과음
Clip Event;           // 이벤트 벽 부숴질 때 효과음
Clip GameOverSound;   // 모든 공이 떨어질 때 효과음
Clip Clear;           // 스테이지를 클리어 했을 때 효과음
Clip TitleBGM;        // 타이틀 패널 배경음악
Clip GameStart;       // 타이틀 패널에서 스페이스바 눌렀을 때 효과음
Clip GameOverBGM;     // 게임오버 패널 배경음악
Clip Regame;          // 게임오버 패널에서 스페이스바 눌렀을 때 효과음
```

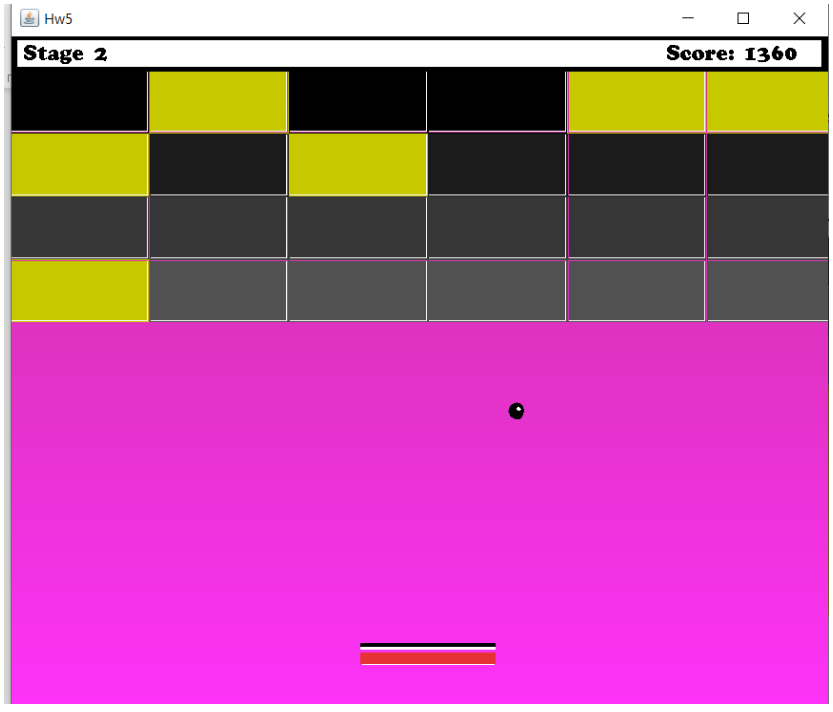
으로 각각의 상황에 넣어주었다. 단순한 게임이기 때문에 효과음과 배경음악을 8비트 느낌으로 골라 게임에 잘 어울리도록 넣었다.

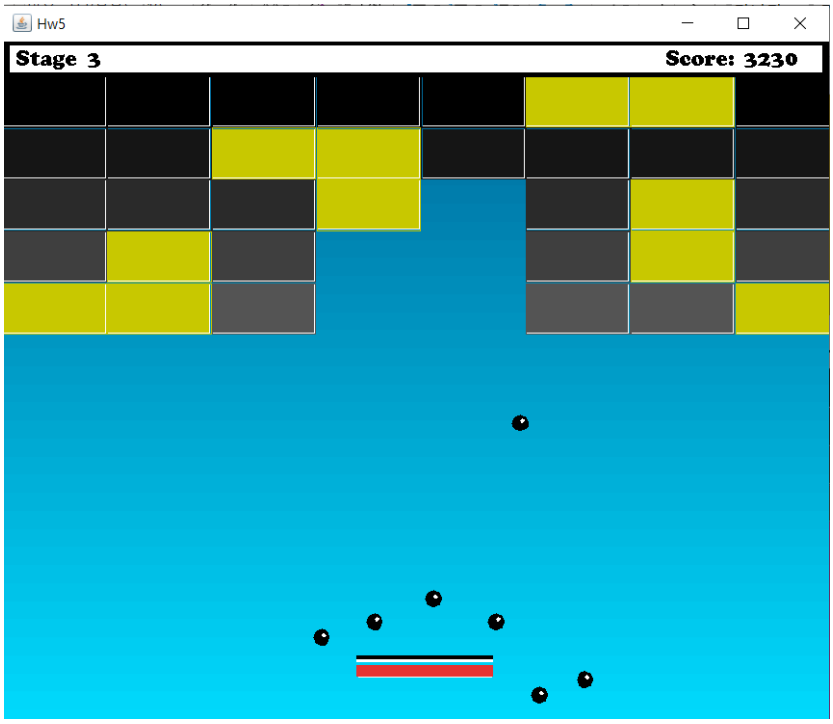
그 뒤로 각각의 paintComponent를 재정의해서 게임을 꾸몄는데

(타이틀화면)

타이틀 화면에서는 배경색을 fillRect로 그라데이션 주고 drawString으로 문자열을 출력했으며 정수형 변수인 time을 선언하고 Thread.sleep을 추가해 시간이 지나면서 깜빡이는 애니메이션을 추가했다.





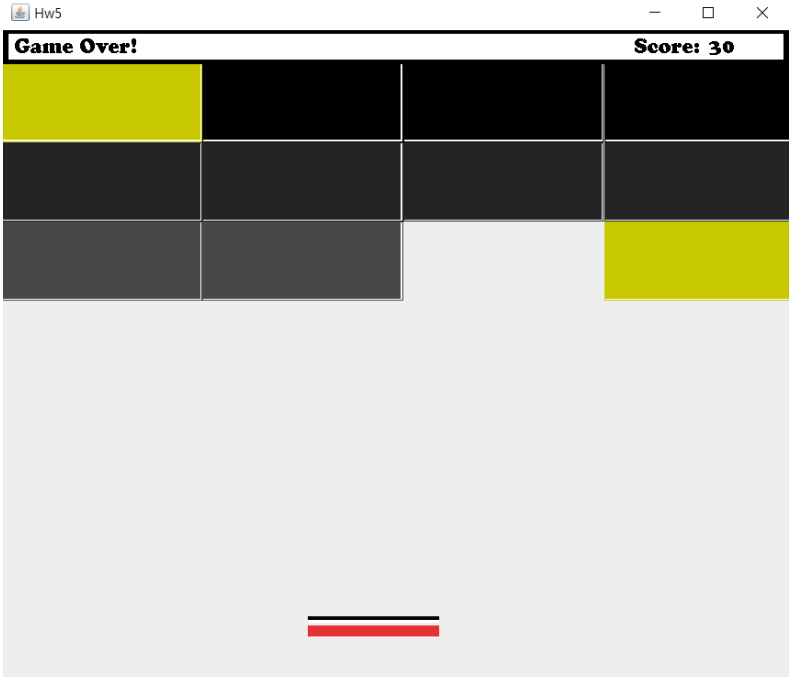


(게임 플레이 화면)

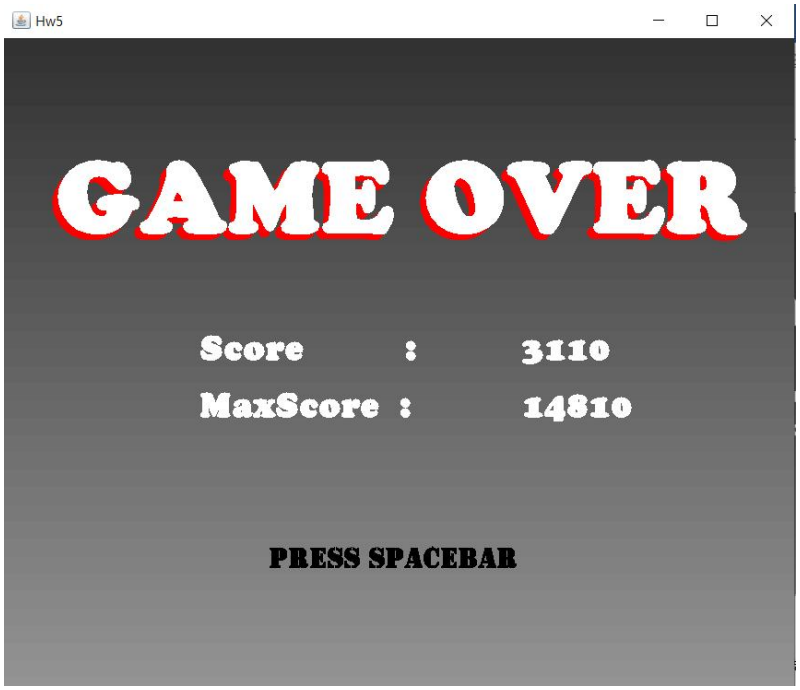
게임 플레이 화면에서는 우선 라켓과 공을 다시 그렸고, 벽돌 겉면에 흰색으로 선을 그려 더 명확하게 만들었다. 배경색은 스테이지가 바뀜에 따라 바뀌게 만들었고 벽돌 위에는 현재 스테이지와 점수가 나오도록 만들었다.

(벽돌을 다 깨서 스테이지를 클리어 했을 때)

ballList.clear로 모든 공을 지워 화면에서 없어지게 만들었다.



(공이 다 떨어졌을 때)



(게임 오버 화면)

게임 오버 화면에서는 Title과 비슷한 방식으로 그렸고 현재 스코어와 최고 기록을 출력하게 만들었다.

Goal 5. 객체지향의 관점에서 프로그램을 디자인하였으며, 이를 보고서에 잘 기술함

객체 지향의 관점으로 프로그램을 디자인 하려고 노력했는데, 일단 **게임 화면에 나오는 물체(벽돌과 공)**을

```
public abstract class object
	{
		public abstract void draw(Graphics g);
	}
```

(class Balls에서)

```
		@Override
		public void draw(Graphics g)
		{
			g.setColor(Color.BLACK);
			g.fillOval(bx,by,r*2,r*2);
			g.setColor(Color.WHITE );
			g.fillOval(bx+(bw/2),by+3,(r*2)/3,(r*2)/3);
		}
```

(class Blocks에서)

```
		@Override
		public void draw(Graphics g)
		{
			if(broken==false)
			{
				g.drawRect(blockx-1,blocky-1,blockw+2,blockh+2);
				g.fillRect(blockx,blocky,blockw,blockh);
			}
		}
```

라는 **추상 클래스 object를 상속 받아 추상 메소드를 다르게 구현**하도록 만들었다. 그리고 각각 블록과 공은 class 내의 각기 다른 값들을 가지고 있어 각 객체마다 다르게 출력되고 다르게 충돌되며 다르게 움직이게 설계했다. Block은 init()으로 처음에 정해진 값이지만 Balls 같은 경우에는 상황에 따라 늘어나고 줄어들 수 있기 때문에 LinkedList를 사용했으며 개수가 늘고 줄음에 따라 for each문을 통해 필요한 만큼만 검사 할 수 있었다.

또한 화면의 크기, 블록의 크기, 블록의 개수, 공의 속도, 공의 크기 등등 많은 것을 변수로 설정 해 놓았고 위와 같이 객체 지향적으로 디자인했기 때문에 후에 공마다 각기 다른 크기와 속도, 블록의 각기 다른 크기도 설정 할 수 있다는 장점을 알았고 지금은 draw 메소드 하나밖에 만들지 못해 아쉬웠지만 다음에 프로그램을 짤 때는 충돌 검사를 넣는 등 더 계획적으로 짜야겠

다는 생각을 했다.

자바를 사용해서 더 복잡한 프로그램은 처음 만들어보아 서투른게 많았다. 만들면서도 비 효율적으로 프로그래밍 한 부분이 많아 이제 자바로 다른 프로그램을 만들 때는 처음부터 더 효율적이게 프로그램 할 수 있을 것 같고 처음에는 막막했으나 하나 하나 시행착오를 거치면서 해결 하면서 즐겁게 만들 수 있었다.