문제해결 및 실습: Java 과제 보고서

과제 5: Block Breaker

전공	소프트웨어학과
학번	21011778
이름	염지환
제출일	2024.12.23

세종대학교
SEJONG UNIVERSITY

[목차]

내용

1. 클래스 설계

I. 배경, 오디오, 텍스트 관련 클래스

모든 씬에 공통적으로 쓰이는 배경을 그리거나, 특정 상황에 오디오를 재생시킬 때, 그리고 외부 폰트를 불러와 사용할 때 매번 중복된 코드를 사용하지 않고 static 메소드를 이용해 편리하게 사용할 수 있도록 편의성을 증진시키는 클래스를 설계하여 프로젝트 전반에 사용하였다.

Background, ParticlePos 클래스

```
class ParticlePos {
    int x, y;
    ParticlePos() {
        x = (int)(Math.random()*800);
        y = (int)(Math.random()*800);
    }
}

class Background {
    static ArrayList<ParticlePos> points;
    Background() {
        points = new ArrayList<ParticlePos>();
        for (int i = 0; i < 100; i++) {
            points.add(new ParticlePos());
        }
    }

    static void draw(Graphics g){
        Graphics2D g2 = (Graphics2D) g;

        GradientPaint gradient = new GradientPaint(0, 0, new Color(0, 0, 20), 0, 800, new Color(50, 50, 100));
        g2.setPaint(gradient);
        g2.fillRect(0, 0, 800, 800);

        for(var p : points ) {
            g2.setColor(new Color(255,255,255));
            g2.fillRect(p.x, p.y, 5, 5);
        }
    }
}
```

Background 클래스는 배경화면을 칠하기 위한 클래스로 남색 그라데이션과 별을 표현한 입자들로 우주와 같은 배경을 표현하기 위해 만들었다. 게임 실행시 ParticlePos 의 생성자가 100 번 호출되어 위치가 무작위로 생성되어 리스트에 저장되고 draw 함수에서 그려질 별들의 위치를 저장한다. static 으로 선언된 draw 함수를 통해 각 씬의 paintComponent()함수에서 먼저 호출되어 게임 실행 중 배경화면이 유지되도록 한다.

Audio 클래스

```
class Audio {
    static Clip clip;
    Audio(){

    }
    static void playAudio(String audio) {
        try {
            if(clip != null)
                clip.stop();
            clip = AudioSystem.getClip();
            ClassLoader classLoader = Audio.class.getClassLoader(); // Audio 클래
            URL url = classLoader.getResource(audio);
            AudioInputStream audioStream = AudioSystem.getAudioInputStream(url);
            clip.open(audioStream);
        } catch(Exception e) {
            e.printStackTrace();
        }
        clip.setFramePosition(0);
        clip.start();
    }
}
```

Audio 클래스는 게임 전반에서 효과음을 실행하기 위한 클래스이다 static 메소드로 선언된 playAudio 는 매개변수로 받은 파일명에 해당하는 오디오 파일을 불러와 clip 에 저장하여 오디오를 실행한다. 이때 이미 다른 오디오가 실행중이라면 해당 클립을 멈추고 실행한다. playAudio 메소드를 이용해 어떤 씬에서도 편하게 오디오를 실행하는 코드를 작성할 수 있었다

```
class FontLoader {
    public static Font pixelFont(float size) {
        try {
            URL url = FontLoader.class.getClassLoader().getResource("PressStart2P-Regular.ttf");
            InputStream fontStream = url.openStream();
            Font font = Font.createFont(Font.TRUETYPE_FONT, fontStream).deriveFont(size);
            fontStream.close();
            return font;
        } catch (Exception e) {
            e.printStackTrace();
            return new Font("Arial", Font.PLAIN, (int) size);
        }
    }
}
```

FontLoader 클래스는 외부 폰트를 원활하게 사용하기 위한 클래스이다 static pixelFont 메소드는 게임에 사용된 PressStart2P 폰트를 불러와 매개변수로 받은

사이즈를 적용한 Font 클래스를 반환한다. 이를 통해 게임에 사용되는 모든 폰트를 외부 폰트로 편리하게 변경할 수 있었다.

GameLabel, FlickLabel 클래스

```
class GameLabel extends JLabel{

    GameLabel(String s, float size){
        super(s);
        setFont(FontLoader.pixelFont(size));
        setForeground(Color.white);
    }
}

class FlickLabel extends JLabel{

    FlickLabel(String s){
        super(s);
        setForeground(Color.red);

        Thread flick = new Thread(() -> {
            while (true) {
                try {
                    Thread.sleep(200);
                    setFont(FontLoader.pixelFont(20));
                    Thread.sleep(200);
                    setFont(FontLoader.pixelFont(0));
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });

        flick.start();
    }
}
```

게임에서 자주 출력되는 텍스트를 쉽게 출력하기 위해 JLabel 클래스를 상속받은 GameLabel 과 FlickLabel 클래스를 제작하였다. FlickLabel 클래스의 경우 스레드를 사용하여 주기적으로 글자 크기를 바꾸어 깜빡거리는 효과를 주었다.

II. 게임 오브젝트 클래스

추상 클래스 GameObject

```
abstract class GameObject {
    float x,y;
    GameObject(){

    }
    abstract void draw(Graphics g);
    void update(float dt) {};
    boolean collisionResolution(GameObject o) {return false;}
}
```

게임의 제일 핵심이 되는 클래스로 게임 씬에서 동작하는 모든 오브젝트는 해당 클래스를 상속받는다. 위치 x, y 를 변수로 갖고있으며 화면에 객체를 그리기 위한 void 함수를 추상 메소드로 정의하여 구현을 강제해두었다. 그리고 메인 루프에서 dt 시간당 변화를 적용하기 위한 update 메소드와 충돌을 처리하기 위한 collisionResolusion 메소드를 만들어두었다.

Ball 클래스

Ball 클래스는 게임의 볼을 구현한 클래스이다 외형은 다음과 같다

```
class Ball extends GameObject{
    float vx, vy;
    float prev_x, prev_y;
    float r;
    float speed;
    double angle;

    Ball(int _x, int _y, float _s){
        x = _x;
        y = _y;
        speed = _s;

        prev_x = x;
        prev_y = y;
        r = 5;

        angle = Math.random()*1.5708 + 0.7853;
        vx = (float)(Math.cos(angle))*speed;
        vy = -(float)(Math.sin(angle))*speed;
    }
```

Ball 은 x 방향과 y 방향의 이동속도와 충돌시에 이전 위치를 기억하기 위한 prev_x 와 prev_y, 반지름 r 과 공의 속도 speed, 그리고 생성 시 진행 방향을 정하기 위한 angle 을 변수로 가지고있다

첫번째 생성자는 스테이지 시작 시 볼 생성을 위한 생성자로 위치와 스피드만을 매개변수로 받아 각 변수를 초기화하며 볼의 방향은 패들을 x 축으로 보고 패들 중심이 0 이라고 하였을 때 45~135 도 사이의 각도로정해진다.

```
Ball(Ball in, float addAngle){
    x = in.x;
    y = in.y;
    speed = in.speed;
    angle = Math.atan2(in.vy, in.vx) + addAngle;

    prev_x = x;
    prev_y = y;
    r = 5;

    vx = (float) (Math.cos(angle)*speed);
    vy = (float) (Math.sin(angle)*speed);
}
```

Ball 의 두번째 생성자는 볼이 노란 블록에 충돌하였을 때 추가 볼을 생성하기 위한 생성자로 Ball 과 추가할 각도 addAngle 을 매개변수로 받아 원래 공의 진행방향에서 addAngle 로 전달받은 각도만큼 틀어진 각도로 이동하는 공을 생성한다. atan2 함수를 이용하기 전에는 asin, scos 등의 함수를 사용했는데 잘 되지 않아 이를 수정하는데 큰 시간을 들였다. 또한 각도 계산은 원래 메인 루프에서 진행했었는데 코드의 가독성이 매우 떨어져 Ball 의 생성자에 따로 구현하였다.

```
@Override
void draw(Graphics g) {
    Graphics2D g2 = (Graphics2D) g;
    GradientPaint gradient = new GradientPaint(x, y-r, Color.orange, x, y+r, Color.red);
    g2.setPaint(gradient);
    g2.fillOval((int)(x-r), (int)(y-r), (int)(2*r), (int)(2*r));
    g.setColor(Color.yellow);
    g.fillOval((int)(x-r+2), (int)(y-r+2), (int)(2*r-4), (int)(2*r-4));
}
@Override
void update(float dt) {
    prev_x = x;
    prev_y = y;
    x += vx*dt;
    y += vy*dt;
}
```

다음은 draw 메소드를 이용해 공을 그리는 코드와 게임 루프에서 반복적으로 호출되어 공의 이동을 구현하는 update 함수이다. 현재 공의 각 축의 속도에 dt 를 곱하여 x 와 y 에 더해주는 식으로 자연스러운 이동을 구현하였다.

```
@Override
boolean collisionResolution(GameObject o) {
    if(o instanceof Wall) {
        Wall wall = (Wall) o;
        float left = wall.x - r;
        float right = wall.x + wall.w + r;
        float top = wall.y - r;
        float bottom = wall.y + wall.h + r;


        if(x>left && x<right && y>top && y<bottom) {
            //color = color.red;
            if(prev_y < top)    { y = top - r;      vy = -vy;   return true;}
            if(prev_y > bottom) { y = bottom + r;   vy = -vy;   return true;}
            if(prev_x < left)   { x = left - r;     vx = -vx;   return true;}
            if(prev_x > right)  { x = right + r;    vx = -vx;   return true;}

        }
    }
```

다음은 collisionResolution 메소드로 충돌한 객체가 벽인지 블록인지 패들인지에 따라 충돌 처리가 달라진다. 그리고 충돌이 발생했을 경우 true 를 반환하고 아니라면 false 를 반환하도록 설계되어있다. 먼저 벽일 경우 벽의 경계를 계산하여 충돌했을 경우 공의 위치를 조정해주고 위나 아래에 충돌했을 경우 vy 를, 왼쪽이나 오른쪽에 충돌했을 경우 vx 를 반대로 변경해주었다.

```
if(o instanceof Block) {
    Block block = (Block) o;

    float left = block.x - r;
    float right = block.x + block.w + r;
    float top = block.y - r;
    float bottom = block.y + block.h + r;


    if(x>left && x<right && y>top && y<bottom && !block.broken) {
        if(o instanceof YellowBlock) {Audio.playAudio("YellowBlockHit.wav");}
        else {Audio.playAudio("BlockHit.wav");}

        //color = color.red;
        block.broken = true;
        if(prev_y < top)    { y = top - r;      vy = -vy;   return true;}
        if(prev_y > bottom) { y = bottom + r;   vy = -vy;   return true;}
        if(prev_x < left)   { x = left - r;     vx = -vx;   return true;}
        if(prev_x > right)  { x = right + r;    vx = -vx;   return true;}
    }
}
```

공이 블록에 충돌했을 경우 처리는 벽과 크게 다르지 않다. 벽과의 차이점은 block 의 broken 플래그가 false 인 블록하고만 충돌 처리를 한다는 점, 그리고 충돌했을 경우 블록에 종류에 따라 다른 효과음을 출력한다는 점, 그리고 충돌한 블록의 broken 플래그를 true 로 바꿔준다는 점이다.

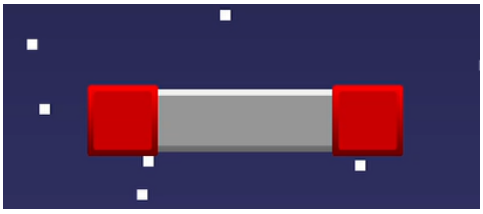
```
if(o instanceof Paddle) {
    Paddle paddle = (Paddle) o;
    float left = paddle.x - 5 - r;
    float right = paddle.x + paddle.w + 5 + r;
    float top = paddle.y - r;
    float bottom = paddle.y + paddle.h + r;


    if(x>left && x<right && y>top && y<bottom) {
        Audio.playAudio("PaddleHit.wav");
        //color = color.red;
        angle = ((x-paddle.x)/paddle.w)*1.5708 + 0.7853;
        if(prev_y < top)    { y = top - r;}
        if(prev_y > bottom) { y = bottom + r;}
        if(prev_x < left)   { x = left - r;}
        if(prev_x > right)  { x = right + r;}
        vx = -(float)(Math.cos(angle)*speed);
        vy = -(float)(Math.sin(angle)*speed);
        return true;
    }
}
return false;
```

다음은 공이 패들과 충돌했을 경우이다 패들과 충돌했을 때는 게임의 재미를 위해 방향을 바꾸는 방식을 다르게 적용하였다.

패들을 x 축이고 패들의 중심이 0 이라고 했을 때 패들의 정 중앙에 공이 맞으면 90 도 방향, 즉 똑바로 위로 공이 움직이도록 하였으며, 패들의 왼쪽으로 멀어질수록 각도가 증가해 왼쪽 끝에선 135 도로 움직이도록, 그리고 오른쪽으로 멀어질수록 각도가 감소해 오른쪽 끝에선 45 도로 움직이도록 변경하였다. 이는 패들의 길이와 위치를 이용해 볼과의 위치를 비교하여 비율을 계산해 적용했다.

공이 어디에도 충돌하지 않았을 경우에는 false 를 반환한다.

Paddle 클래스



Paddle 클래스는 플레이어가 조종하는 바를 구현한 클래스이다 외형은 Alkanoid 게임의 바우스를 참고하였다.

```
class Paddle extends GameObject{
    float w, h;
    int mode;
    Paddle(int _x, int _y, int _w, int _h){
        x = _x;
        y = _y;
        w = _w;
        h = _h;
        mode = 0;
    }
```

Paddle 의 필드 및 생성자는 다음과 같다. 너비와 높이 w h 와 아이템 적용 상태를 나타낼 mode 가 있으며 생성자에서는 위치와 크기를 받아 초기화하고 mode 를 0 으로 만든다.

```
@Override
void draw(Graphics g) {
    Graphics2D g2 = (Graphics2D) g;
    int bright = 0;
    if(mode == 1) {
        bright = 50;
    }
    if(mode == 2) { if(w < 250) {w += 10;} }
    else { if(w > 150) {w -= 10;} }

    GradientPaint gradient = new GradientPaint(0, y, new Color(250,250,250), 0 , y+h, new Color(100,100,100));
    g2.setPaint(gradient);
    g2.fillRect((int)x, (int)y, (int)w, (int)h);
    g2.setColor(new Color(150,150,150));
    g2.fillRect((int)x+3, (int)y+3, (int)w-6, (int)h-6);

    gradient = new GradientPaint(0, y-2, new Color(255,0,0), 0, y+h+2, new Color(100 + bright,0 + bright,0 + bright));
    g2.setPaint(gradient);
    g2.fillRoundRect((int)(x-2), (int)y-2, 34, 34, 3, 3);
    g2.fillRoundRect((int)(x+w-34), (int)y-2, 34, 34, 3, 3);
    g2.setColor(new Color(200 + bright,0 + bright*2,0 + bright*2));
    g2.fillRoundRect((int)(x+1), (int)(y+1), 28, 28, 3, 3);
    g2.fillRoundRect((int)(x+w-31), (int)(y+1), 28, 28, 3, 3);
}
```

다음은 paddle 의 draw 메소드이다 기본적인 외형을 그리는 부분을 제외하고 주목할 점은 모드에 따른 변화이다. Mode 가 1, 미사일 모드일 경우 bright 값이 50 이 되어 패들 양쪽의 빨간색 사각형이 밝게 변한다.

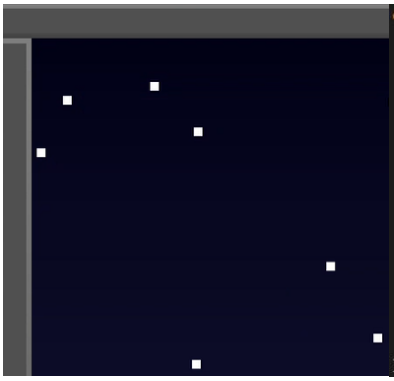
Mode 가 2, 확장 모드일 경우 크기가 250 이 될 때 까지 패들의 크기를 조금씩 늘리며, 그려질 때 마다 호출되어 아이템을 먹었을 경우 부드럽게 패들의 크기가 커진다. 만약 Mode2 에서 벗어났다면 크기는 다시 서서히 줄어들어 원래 패들의 크기가 된다.

```
void move(int dx) {
    x += dx;
    if(x < 20)
        x = 20;
    if(x > 765-w)
        x = 765-w;
}

void init() {
    mode = 0;
    w = 150;
    x = 320;
}
```

Paddle 을 움직이기 위한 move 메소드와 라운드 종료 시 초기화를 위한 init 메소드를 추가하였다. move 메소드는 GameScene 의 플래그 상태에 따라 메인 루프에서 반복적으로 호출되어 패들이 서서히 움직이도록 해준다.

init 함수는 원래 존재하지 않았으나 라운드가 끝났을 때 패들이 벽쪽에 붙어있을 경우 첫 공을 치지도 못하고 게임이 끝나는 경우가 많아 이를 조정하기 위해 라운드가 끝나면 패들의 위치를 초기화하기 위해 추가하였다.

Wall 클래스



```
class Wall extends GameObject{
    float w, h;
    Wall(int _x, int _y, int _w, int _h){
        x = _x;
        y = _y;
        w = _w;
        h = _h;
    }
    @Override
    void draw(Graphics g) {
        Graphics2D g2 = (Graphics2D) g;

        GradientPaint gradient = new GradientPaint(0, 0, Color.gray, 0, h, Color.darkGray);
        g2.setPaint(gradient);
        g2.fillRect((int)x, (int)y, (int)w, (int)h);

        g.setColor(new Color(80,80,80));
        g2.fillRect((int)x+3, (int)y+3, (int)w-6, (int)h-6);
    }
}
```

Wall 클래스는 게임의 외벽을 나타내는 클래스로 높이와 너비를 필드로 가지고 draw 메소드만 오버라이딩 하여 가지고 있다. 모든 충돌 처리는 ball 에서 하고 있으므로 구현이 간단했다.

Block 클래스

Block 클래스는 공과 충돌하면 부서지는 블록을 구현한 클래스이다

```
class Block extends GameObject{
    float w, h;
    boolean broken;
    int opacity = 255;
    Block(int _x, int _y, int _w, int _h){
        x = _x;
        y = _y;
        w = _w;
        h = _h;
        broken = false;
    }
    @Override
    void draw(Graphics g) {
        if(broken == true) {
            opacity -= 20;
            if(opacity < 0)
                opacity = 0;
        }
        Graphics2D g2 = (Graphics2D) g;

        GradientPaint gradient = new GradientPaint(0, y, new Color(250,100,250, opacity), 0, y+h, new Color(100,40,100, opacity));
        g2.setPaint(gradient);
        g2.fillRect((int)x, (int)y, (int)w, (int)h);

        g.setColor(new Color(200,80,200, opacity));
        g2.fillRect((int)x+3, (int)y+3, (int)w-6, (int)h-6);
    }
}
```

멤버 필드로 너비와 높이, 그리고 broken 플래그를 가지고 있으며 투명하게 사라지는 효과를 사용하기 위해 opacity 를 255 로 초기화하여 가지고있다. draw 에서는 broken 이 true 일 경우 opacity 를 계속 20 씩 줄고 이것이 모든 컬러에 적용되어 블록이 서서히 사라지는 것처럼 보이도록 디자인하였다.

YellowBlock 클래스

YellowBlock 은 일정 확률로 생성되어 충돌시 공을 3 개로 만드는 블록을 구현한 클래스이다

```
class YellowBlock extends Block {
    int bright;
    int time;
    YellowBlock(int _x, int _y, int _w, int _h){
        super(_x,_y,_w,_h);
        bright = 0;
        time = (int)(Math.random()*3000);
        new Thread(()->{
            try {
                Thread.sleep(time);

                while(true) {
                    bright = 50;
                    Thread.sleep(250);
                    bright = 0;
                    Thread.sleep(2750);
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }).start();
    }
```

YellowBlock 은 Block 을 상속받아 만들어졌고 생성자에서 Block 의 생성자를 먼저 호출한다. 이후로는 주기적으로 주기 사이의 블록간 반짝이는 순서와 시간 간격을 랜덤으로 하기 위해 초기 시간 time 을 랜덤 0~3 초 사이으로 초기화해주고 Thread 를 이용하여 time 만큼 대기 후 3 초마다 반짝이도록 bright 값을 50 으로 설정하고 다시 0 으로 되돌려 주었다. draw 메소드는 색상이 변경되고 bright 가 적용된 것 말고는 block 과 동일하므로 생략한다.

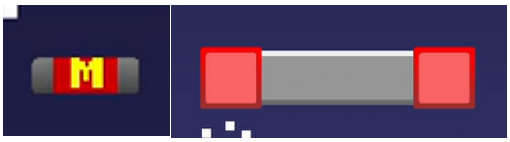
```java
class Item extends GameObject{
    float w, h;
    Item(int _x, int _y, int _w, int _h){
        x = _x;
        y = _y;
        w = _w;
        h = _h;
    }

    @Override
    void draw(Graphics g) {}

    @Override
    void update(float dt) {y += 400*dt;}
    @Override
    boolean collisionResolution(GameObject o) {
        if(o instanceof Paddle) {
            Paddle paddle = (Paddle) o;
            float left = paddle.x - 5 - w/2;
            float right = paddle.x + paddle.w + 5 + w/2;
            float top = paddle.y - h/2;
            float bottom = paddle.y + paddle.h + h/2;

            if(x>left && x<right && y>top && y<bottom) {
                return true;
            }
        }
        return false;
    }
}
```

추가 기능 구현을 고민하였고 Alkanoid 게임에 등장하는 2 개의 아이템, Laser 와 Enlarge 를 참고하여 구현해보기로 하였다. 이를 위해 Item 클래스를 하나 만들어 이를 상속하여 아이템을 만들기로 생각했다.

Item 은 생성되면 빠른 속도로 아래방향으로 떨어진다.

Item 은 위치에 더해 너비와 높이를 가지고 있으며 패들과 충돌할 경우 true, 아니면 false 를 반환한다.

MissileItem 클래스

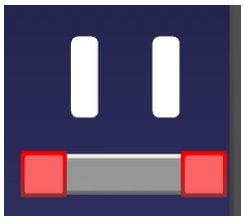


MissileItem 은 Alkanoid 의 Laser 아이템을 참고하여 만든 아이템으로 패들에 접촉하여 획득 시 미사일 모드가 되어 스페이스바를 눌러 미사일을 발사할 수 있다. 다른 아이템 획득시에는 비활성화된다.

```java
class MissileItem extends Item{

    MissileItem(int _x, int _y) {
        super(_x, _y, 30, 10);
    }

    @Override
    void draw(Graphics g) {
        Graphics2D g2 = (Graphics2D) g;
        GradientPaint gradient = new GradientPaint(0, y, Color.gray, 0, y+h, Color.darkGray);
        g2.setPaint(gradient);
        g2.fillRoundRect((int)(x), (int)y, (int)w, (int)h, 3, 3);
        g2.setColor(new Color(200,0,0));
        g2.fillRect((int)(x+6), (int)(y), 18, 10);

        g2.setFont(FontLoader.pixelFont(10.0f));
        g2.setColor(Color.yellow);
        g2.drawString("M", x+w/2-4, y+h);
    }
}
```

미사일아이템 클래스는 외형을 그리기 위한 draw 메소드만 구현되어있다

Missile 클래스



Missile 클래스는 미사일 모드에서 스페이스바를 누르면 나가는 미사일을 구현한 클래스로 블록과 충돌하면 블록이 사라지며 미사일 역시 사라진다

```java
class Missile extends GameObject{

    float w, h;
    Missile(int _x, int _y){
        x = _x;
        y = _y;
        w = 80;
        h = 60;
    }

    @Override
    void draw(Graphics g) {
        g.setColor(Color.white);
        g.fillRoundRect((int)x, (int)y, 20, 60,10,10);
        g.fillRoundRect((int)(x+60), (int)y, 20, 60,10,10);
    }

    @Override
    void update(float dt) {
        y -= 800*dt;
    }
```

미사일은 위치와 너비, 높이를 가지고 있으며 긴 사각형 2 개를 나란히 세워놓은 외형으로 구현 하였다. 미사일은 생성된 뒤 게임 루프에서 지속적으로 호출되는 update 메소드에 의해 빠른 속도로 위 방향으로 진행한다.

```
@Override
boolean collisionResolution(GameObject o) {
    if(o instanceof Wall) {
        Wall wall = (Wall) o;
        float left = wall.x;
        float right = wall.x + wall.w;
        float top = wall.y;
        float bottom = wall.y + wall.h;


        if(x+w/2>left && x-w/2<right && y<bottom) {
            return true;
        }
    }
    if(o instanceof Block) {
        Block block = (Block) o;

        float left = block.x;
        float right = block.x + block.w;
        float top = block.y;
        float bottom = block.y + block.h;

        if(x+w/2>left && x-w/2<right && y>top && y<bottom && !block.broken) {
            block.broken = true;
            return true;
        }
    }
    return false;
}
```

미사일은 벽과 블록, 두가지 오브젝트와 충돌처리를 진행하며 만약 벽에 충돌했을 경우 사라지기만 하며, broken 플래그가 false 인블록에 충돌했을 경우 블록의 broken 을 true 로 활성화 시킨다. 역시 충돌했을 경우 true, 아닐 경우 false 를 반환한다.

EnlargeItem 클래스

EnlargeItem 클래스는 Alkanoid 의 Enlarge 아이템을 참고하여 만든 아이템으로 획득 시 패들의 길이가 늘어난다. 역시 미사일 아이템을 획득하면 다시 길이가 짧아지며, 중첩은 불가능하다. Enlarge 아이템의 효과는 패들 mode 를 2 로 바꾸어 draw 함수의 코드를 통해 패들의 w 값을 늘려준다.

```
class EnlargeItem extends Item{

    EnlargeItem(int _x, int _y) {
        super(_x, _y, 30, 10);
    }

    @Override
    void draw(Graphics g) {
        Graphics2D g2 = (Graphics2D) g;
        GradientPaint gradient = new GradientPaint(0, y, Color.gray, 0, y+h, Color.darkGray);
        g2.setPaint(gradient);
        g2.fillRoundRect((int)(x), (int)y, (int)w, (int)h, 3, 3);
        g2.setColor(new Color(0,0,200));
        g2.fillRect((int)(x+6), (int)(y), 18, 10);

        g2.setFont(FontLoader.pixelFont(10.0f));
        g2.setColor(Color.yellow);
        g2.drawString("E", x+w/2-4, y+h);
    }
}
```

III. 메인 클래스 및 게임 씬 클래스

추상 클래스 Scene

```
abstract class Scene extends JPanel{
    Scene(){

    }
    abstract void init();
}
```

추상 클래스 Scene 은 씬 변경을 편하게 하기 위해 만든 추상 클래스이다 추상 메소드로 씬을 초기화하는 init 메소드를 만들어 Scene 을 상속받은 클래스들이 초기화 함수를 가지고 있도록 하였다.

TitleScene 클래스

```
class TitleScene extends Scene implements KeyListener{
    BlockBreaker frame;

    TitleScene(BlockBreaker frame){
        setLayout(null);
        this.frame = frame;

        setFocusable(true);
        addKeyListener(this);

        GameLabel l1 = new GameLabel("Java Programming", 40);
        GameLabel s1 = new GameLabel("Java Programming", 40);
        GameLabel l2 = new GameLabel("Homework #5", 40);
        GameLabel s2 = new GameLabel("Homework #5", 40);
        GameLabel l3 = new GameLabel("BlockBreaker", 60);
        GameLabel s3 = new GameLabel("BlockBreaker", 60);

        FlickLabel l4 = new FlickLabel("Press Spacebar to play!");
        FlickLabel s4 = new FlickLabel("Press Spacebar to play!");

        l1.setBounds(80, 120, 800, 40);
        l2.setBounds(170, 200, 800, 40);
        l3.setBounds(40, 350, 800, 80);
        l4.setBounds(170, 600, 800, 20);
        add(l1);
        add(l2);
        add(l3);
        add(l4);
```

TitleScene 클래스는 frame 을 매개변수로 받아 정보를 일부 수정할 수 있도록 하였다. GameLabel 을 이용하여 게임 타이틀 글씨를 출력하도록 하였으며 키보드 입력을 위해 KeyListener 인터페이스를 구현하였다.

```
void init() {
    requestFocus();
    Audio.playAudio("Game Start.wav");
}

public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Background.draw(g);
}

public void keyTyped(KeyEvent e) {}
public void keyPressed(KeyEvent e) {
    if(e.getKeyCode() == KeyEvent.VK_SPACE) {
        frame.changeScene(2);
    }
}
public void keyReleased(KeyEvent e) {}
```

TitleScene 의 init 메소드는 패널의 포커스를 요청하며 게임 시작 효과음을 출력한다. paintComponent 메소드에서는 Background 를 이용해 배경을 그린다. KeyPressed 메소드에서는 스페이스바가 눌렸을 때 게임 씬으로 씬을 교체한다

GameScene

```
class GameScene extends Scene implements KeyListener,
                                         Runnable{
    BlockBreaker frame;
    LinkedList <GameObject> walls;
    LinkedList <GameObject> balls;
    Item item;
    Paddle paddle;
    Thread gameLoop;
    int level;
    int count;
    float ballspeed;

    GameScene(BlockBreaker frame) {
        setLayout(null);
        this.frame = frame;
        level = 0;
        frame.yourScore = 0;
        walls = new LinkedList<>();
        balls = new LinkedList<>();

        paddle = new Paddle(320, 700, 150, 30);

        gameLoop = new Thread(this);

        setFocusable(true);
        addKeyListener(this);
    }
```

GameScene 은 게임의 가장 핵심인 게임 씬을 구현한 클래스이다. 키보드 입력과 게임 루프를 구현해야 하므로 KeyListene Runnable 인터페이스를 구현하였다 충돌을 구현한 공이나 미사일 클래스들을 저장할 balls 와 충돌하는 벽이나 블록, 패들을 저장할 walls 리스트를 선언하였다. GameObject ㄴ리스트 하나로 아이템까지 구현하기엔 어려운 부분이 있어 두개의 리스트를 사용하게 되었다. 필드에 아이템은 한번에 하나만 나오므로 아이템은 변수 하나로 저장하였다. 패들을 따로 저장하여 조종하기위한 패들 변수와 게임 메인 루프를 실행시키기 위한 변수 gameLoop, 그리고 게임의 현재 레벨을 저장할 level, 부서진 블록 수를 저장하는 count, 현재 공의 스피드를 저장하는 ballspeed 변수가 있다.

생성자에서는 레벨을 0 으로 초기화하고 frame 의 yourScore 를 0 으로 초기화한다. 그 후 다른 로컬 변수들을 초기화하고 패들을 생성한다. 또한 gameLoop 에 run 함수를 실행하는 스레드를 생성하여 저장한다.

```
void init() {
    requestFocus();
    Audio.playAudio("Round Start.wav");

    level++;
    ballspeed = 350.0f + 50.0f*level;

    balls.clear();
    walls.add(paddle);
    walls.add(new Wall(0,0,790,20));
    walls.add(new Wall(0,20,20,745));
    walls.add(new Wall(767,20,20,745));
    balls.add(new Ball(400,690,ballspeed));

    item = null;
    paddle.init();

    count = 0;
    int num = level * 3;
    int bw = (751 - (5 * (num + 1))) / num;
    int bh = (400 - (5 * (num + 1))) / num;
    for (int i = 0; i < level * 3; i++) {
        for (int j = 0; j < level * 3; j++) {
            if((int)(Math.random()*1000)%5 == 0) {walls.add(new YellowBlock(25 + (bw + 5) * i, 25 + (bh + 5) * j, bw, bh));}
            else {walls.add(new Block(25 + (bw + 5) * i, 25 + (bh + 5) * j, bw, bh));}
        }
    }
```

GameScene 의 init 메소드는 게임 씬으로 전환되었을 때와 새 라운드가 시작될 때 호출된다.

포커스를 가져오고 라운드 시작 효과음을 출력한다. 그후 레벨을 1 증가시켜 스테이지레벨을 상승시키고 그에 따른 공의 스피드를 조정한다. 그 후 balls 를

초기화하고 walls 에 패들과 게임의 외벽을 추가한 뒤 패들의 중앙에 공이 오도록 생성하고 패들 역시 중앙에 존재하도록 초기화한다. Item 은 null 로 초기화 시켜 이전 스테이지의 아이템이 남아있지 않도록 처리한다.

다음은 블록 생성 부분으로 블록이 부서진 수를 0 으로 초기화한다, 그리고 주어진 예제와 레벨에 따라 블록을 늘려 3*level x 3*level 의 블록 배열을 생성한다. 이때 5 분의 1 확률로 노란색 블록이 생성되도록 설정하였다.

```
new Thread(() -> {
    GameLabel l = new GameLabel(String.format("Stage %d", level), 80);
    l.setBounds(120, 320, 800, 80);
    try {
        for(int i = 0; i < 5; i++) {
            add(l);
            repaint();
            Thread.sleep(300);
            remove(l);
            repaint();
            Thread.sleep(300);

        }

        if(level == 1)
            gameLoop.start();
        Audio.playAudio("PaddleHit.wav");
    }catch (InterruptedException e) {
        e.printStackTrace();
    }
}).start();
```

다음은 스테이지 시작 전 대기를 위한 스레드이다 현재 스테이지 현황을 화면에 깜빡이는 라벨로 표현하였으며 3 초 뒤에 라벨이 사라지고 게임이 시작되도록 구현하였다.

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Background.draw(g);

    var it1 = balls.iterator();
    while(it1.hasNext())
        it1.next().draw(g);

    var it2 = walls.iterator();
    while(it2.hasNext())
        it2.next().draw(g);
    if(item != null)
        item.draw(g);
}
```

paintComponent 메소드는 배경을 그린 뒤 iterator 를 이용해 balls 와 walls 를 순회하며 오브젝트들을 화면에 그린다. 이때 item 이 생성되어있을 경우 아이템도 그리도록 하였다.

```
boolean moveLeft = false;
boolean moveRight = false;
public void keyTyped(KeyEvent e) {}
public void keyPressed(KeyEvent e) {
    if (e.getKeyCode() == KeyEvent.VK_LEFT) {
        moveLeft = true;
    }
    if (e.getKeyCode() == KeyEvent.VK_RIGHT) {
        moveRight = true;
    }
    if (paddle.mode == 1 && e.getKeyCode() == KeyEvent.VK_SPACE) {
        Audio.playAudio("Missile.wav");
        balls.add(new Missile((int)(paddle.x+35), (int)(paddle.y - 60)));
    }
}

public void keyReleased(KeyEvent e) {
    if (e.getKeyCode() == KeyEvent.VK_LEFT) {
        moveLeft = false;
    }
    if (e.getKeyCode() == KeyEvent.VK_RIGHT) {
        moveRight = false;
    }
}
```

다음은 KeyListener 의 추상 메소드를 구현한 부분이다. 좌 우 방향키를 눌렀을 때 각각 moveLeft, moveRight 함수를 활성화하여 메인 루프에서 부드럽게 패들이 움직이도록 설정해주었다. 원래는 방향키를 누르는 즉시 패들을 일정 방향 이동시키려고 하였으나 움직임이 부드럽지 않아 키보드 입력으로는 플래그만 설정해주고 움직임은 메인 루프에서 구현하는 방식으로 구현했따

패들의 모드가 미사일 모드일 경우 스페이스바를 누르면 효과음과 함께 미사일이 생성되도록 구현하였다.

게임 메인 루프는 설명이 길기에 게임 메인 루프 문단에 자세히 서술하였다.

GameOverScene

```
class GameOverScene extends Scene implements KeyListener{
    BlockBreaker frame;

    GameOverScene(BlockBreaker frame){
        setLayout(null);
        this.frame = frame;

        setFocusable(true);
        addKeyListener(this);

        GameLabel l1 = new GameLabel("Game Over", 80);
        GameLabel s1 = new GameLabel("Game Over", 80);
        GameLabel l2 = new GameLabel(String.format("Your Score : %d", frame.yourScore), 20);
        GameLabel s2 = new GameLabel(String.format("Your Score : %d", frame.yourScore), 20);
        GameLabel l3 = new GameLabel(String.format("High Score : %d", frame.highScore), 20);
        GameLabel s3 = new GameLabel(String.format("High Score : %d", frame.highScore), 20);

        FlickLabel l4 = new FlickLabel("Press Spacebar");
        FlickLabel s4 = new FlickLabel("Press Spacebar");

        l1.setBounds(40, 200, 800, 80);
        l2.setBounds(250, 350, 800, 20);
        l3.setBounds(250, 400, 800, 20);
        l4.setBounds(250, 600, 800, 20);
        add(l1);
        add(l2);
        add(l3);
        add(l4);


        s1.setBounds(45, 205, 800, 80);
        s1.setForeground(Color.black);
        s2.setBounds(255, 355, 800, 20);
        s2.setForeground(Color.black);
```

GameOverScene 의 경우 GameTitle 씬과 크게 다른 점이 없다. GameLabel 을 이용하여 GameOver 텍스트와 현재 점수와 현재까지 게임의 최고점수를 함께 출력한다. FlickLabel 을 이용하여 스페이스바를 누르라는 안내 메시지까지 아래쪽에 출력하였다.

```
void init() {
    requestFocus();
    Audio.playAudio("Game Over.wav");
}

public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Background.draw(g);
}

public void keyTyped(KeyEvent e) {}
public void keyPressed(KeyEvent e) {
    if(e.getKeyCode() == KeyEvent.VK_SPACE) {
        frame.changeScene(1);
    }
}
public void keyReleased(KeyEvent e) {}
```

init 메소드에서는 포커스를 가져오고 게임 오버 효과음을 출력한다. paintComponent 메소드는 먼저 Background 를 그리며 keyPressed 메소드에서는

스페이스바가 입력되면 씬을 TitleScene 으로 교체한다 TitleScene 과 거의 똑 같은 동작을 하는 클래스로 설계하였다.

메인 클래스

```
public class BlockBreaker extends JFrame{
    Scene scene;
    int yourScore;
    int highScore;

    BlockBreaker() {
        setTitle("Java Homework5 : Block Breaker");
        setSize(800,800);
        setResizable(false);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        new Background();
        new Audio();
        scene = new TitleScene(this);

        highScore = 0;

        add(scene);
        scene.init();

        setVisible(true);
    }
```

메인클래스는 필드에 scene 변수와 두 score 변수를 가지고 있다. 게임이 실행되면 생성자가 실행되고, 생성자에선 JFrame 을 초기화해주고 Background 와 Audio 클래스의 생성자를 실행시켜준다.

현재 씬을 저장하는 scene 변수엔 TitleScene 이 생성되어 저장되고 프레임에 scene 을 출력하며 scene 의 init 메소드를 호출하여 초기화한다.

```
public void changeScene(int sceneNum) {
    remove(scene);
    if(sceneNum == 1) {scene = new TitleScene(this);}
    if(sceneNum == 2) {scene = new GameScene(this);}
    if(sceneNum == 3) {scene = new GameOverScene(this);}
    add(scene);
    scene.init();
    revalidate();
    repaint();
}
```

다음은 씬을 교체하는 changeScene 메소드로 메소드가 실행되면 매개변수로 입력된 숫자에 맞게 해당 씬으로 이동한다. 이때 해당 씬은 Frame 에서 지워지고 새로운 씬이 생성되어 scene 에 저장된 뒤 Frame 에 추가된다. 원래는 각 씬마다 변수를 만들어 관리하려 했으나 게임을 재시작하는 과정에서 게임 메인 루프가 제대로 종료되지 않는 문제를 지속적으로 해결하지 못해 아예 씬을 지우고 다시 생성하는 방식으로 해결하였다.

2. 게임 루프 동작

GameScene 설명에서 보았듯이 게임 씬으로 전환되면 3 초 뒤 게임이 시작되고 gameLoop 가 실행되어 run 함수가 동작하기 시작한다.

```
public void run() {

    while (true) {
        try {
            //패들 이동 처리
            if (moveLeft) {paddle.move(-15);}
            if (moveRight) {paddle.move(15);}

            // 아이템이 패들과 닿았을 때 아이템 효과 적용
            if(item != null) {
                item.update(0.033f);
                if(item.collisionResolution(paddle)) {
                    if(item instanceof MissileItem) paddle.mode = 1;
                    if(item instanceof EnlargeItem) {
                        Audio.playAudio("Enlarge.wav");
                        paddle.mode = 2;
                    }
                    item = null;
                }
                else if(item.y > 800) {
                    item = null;
                }
            }
```

run 함수안에선 게임 루프 while 이 돌고있고 이 루프의 반복 주기는 0.033ms 으로 게임은 약 30 프레임으로 동작한다.

루프가 시작되면 가장 먼저 패들의 이동처리 부분이 있다. moveLeft 플래그가 true 면 paddle 은 계속 -15 만큼 이동하고 오른쪽은 그 반대로 작동한다.

패들 이동처리가 끝나면 아이템의 위치 업데이트가 발생한다. 업데이트 결과 아이템이 패들과 충돌했을 경우 패들의 모드를 바꿔준다. EnlargeMode 의 경우 효과음을 함께 출력해준다. 아이템이 적용되면 Item 은 null 이 되어 사라진다.

그리고 아이템이 맵 밖으로 나갔을 경우에도 Item 은 null 이 되어 사라진다.

```
// 충돌 처리 루프
ArrayList<Ball> newBalls = new ArrayList<>(); // 추가 공을 저장할 임시 배열
var it1 = balls.iterator();
while (it1.hasNext()) {
    GameObject o1 = it1.next();
    o1.update(0.033f);

    // 범위를 넘어간 공 삭제
    if(o1.y > 800) {
        it1.remove();
    }

    var it2 = walls.iterator();
    while (it2.hasNext()) {
        GameObject o2 = it2.next();

        // 충돌 발생시 처리
        if (o1.collisionResolution(o2)){

            // 미사일이 블록이나 벽에 맞았을 때 처리
            if(o1 instanceof Missile) {
                if(o2 instanceof Wall) {
                    it1.remove();
                    if(it1.hasNext())
                        it1.next();
                }

                if(o2 instanceof Block) {
                    count++;
                    frame.yourScore += 10;
                    it1.remove();
                    if(it1.hasNext())
                        it1.next();
                }
```

다음은 공 및 미사일과 블록, 벽의 충돌 처리이다. 먼저 노란색 블록에 닿으면 공이 늘어나는데 이 과정에서 리스트 순회 중 공이 추가되어 에러가 발생하는 경우가 잦았기에 공을 임시 배열에 저장하고 리스트 순회가 끝나면 추가할 수 있도록 임시 배열 newBalls 를 선언해주었다.

바깥쪽 반복문은 balls(collisionResolution 이 구현된 오브젝트)를 순회하고 안쪽 반복문은 walls(collisionResolution 이 구현되지 않은 오브젝트)를 순회한다.

순회중 o1 과 o2 사이에 충돌이 발생하면 어떤 오브젝트끼리 충돌했는지에 따라 동작이 달라진다. 위 사진에선 먼저 미사일이 블록이나 벽에 닿았을 때를 처리한다

미사일이 벽에 닿았을 경우엔 바로 미사일을 삭제하고 반복문을 종단한다. 블록에 닿았을 경우 count 와 yourScore 을 올리고 미사일을 삭제하고 반복문을 중단한다.

```
// 볼이 블록에 맞았을 때 처리
if(o2 instanceof Block) {
    count++;
    frame.yourScore += 10;
    Block block = (Block)o2;

    // 보라색 블럭의 경우 아이템이 필드에 없는 경우만 아이템 생성
    if(item == null && !(o2 instanceof YellowBlock)) {
        if((int)(Math.random()*10000)%5 == 0) {
            item = new MissileItem((int)(block.x + block.w/2 - 15) ,(int)(block.y + block.h/2 - 5));
        }
        else if((int)(Math.random()*10000)%5 == 1) {
            item = new EnlargeItem((int)(block.x + block.w/2 - 15) ,(int)(block.y + block.h/2 - 5));
        }
    }

    // 노란 블록 맞았을 시 처리
    if(o2 instanceof YellowBlock) {
         Ball b1 = (Ball)o1;
         newBalls.add(new Ball(b1, 0.5236f));
         newBalls.add(new Ball(b1, -0.5236f));
    }
}
```

이후 볼과 블록의 충돌에 대한 처리를 진행한다. 볼이 블록에 맞았을 경우 count 를 1 올리고 yourScore 를 10 올린다. 만약 보라색 블록과 충돌했을 경우 필드에 아이템이 이미 존재하지 않다면, 보라색 블록의 중앙에 각각 20%의 확률로 아이템을 생성한다

노란색 블록과 충돌했을 경우 충돌 뒤 나아가는 방향에서 +30 도 -30 도 방향으로 나가는 공을 2 개 더 생성하여 newBalls 리스트에 저장한다.

```
// 추가된 공 적용
balls.addAll(newBalls);
repaint();
```

순회가 끝나면 newBalls 의 볼들을 balls 에 전부 넣어주고 패널을 다시 그린다.

```
// 클리어 처리
if(count == (level*3)*(level*3)) {
    Audio.playAudio("StageClear.wav");
    Thread.sleep(2000);
    walls.clear();
    init();
    repaint();
    Thread.sleep(3000);
}
```

만약 순회가 끝나고 count 가 블록의 개수랑 똑같아 졌다면 블록이 존재하지 않는다는 의미이므로 게임이 클리어된다. 클리어 효과음이 나오며 2 초동안 게임이 멈추고 walls 가 초기화된 뒤 init 메소드를 통해 씬이 초기화되어 다음 스테이지가 시작된다. 이때는 루프 자체적으로 3 초를 대기한 뒤 다시 동작을 시작한다.

```
// 공이 없지만 미사일이 남았을 경우 처리
var check = balls.iterator();
while(check.hasNext()) {
    if(check.next() instanceof Ball)
        break;
    balls.clear();
}

// 게임 오버 처리
if(balls.isEmpty() && level > 0) {
    Audio.playAudio("PaddleBroken.wav");
    Thread.sleep(2000);
    if(frame.yourScore > frame.highScore)
        frame.highScore = frame.yourScore;
    frame.changeScene(3);
    break;
}

Thread.sleep(33);
```

마지막으로 게임 오버 처리이다, 미사일 모드의 경우 미사일이 balls 에 저장되기 때문에 balls.isEmpty 로 게임오버 처리를 할 경우 미사일을 계속 발사하면 공이 다 사라져도 게임 오버 처리가 되지 않는 문제가 있었다.

따라서 공이 없지만 미사일만 남았을 경우를 검사하여 공이 하나도 없을 경우 balls 를 강제로 비워버리는 코드를 추가하였다.

이후 볼이 하나도 없고, 게임 중이라면 게임 오버 코드가 실행된다. 효과음이 재생되며 2 초간 게임이 멈추고 하이스코어 갱신이라면 하이스코어가 갱신된다. 이후 게임 오버 씬으로 씬을 전환한다.

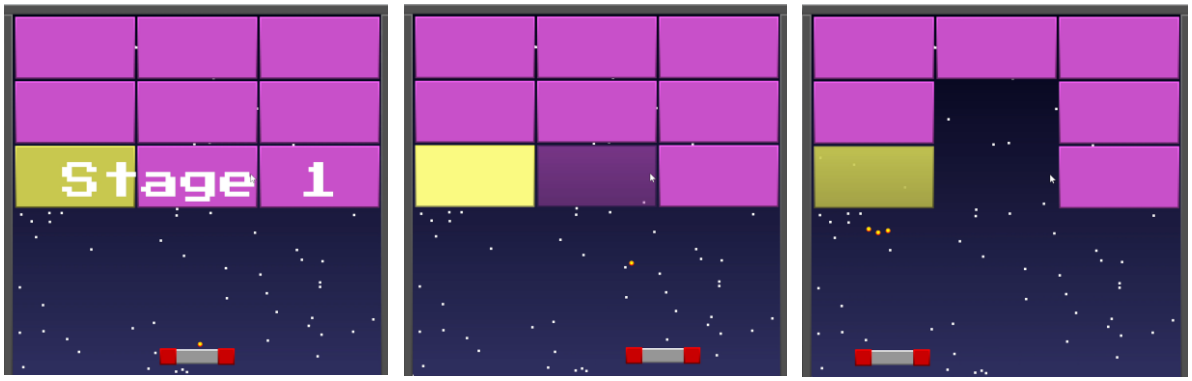
3. 결과

I. 게임 진행 요약

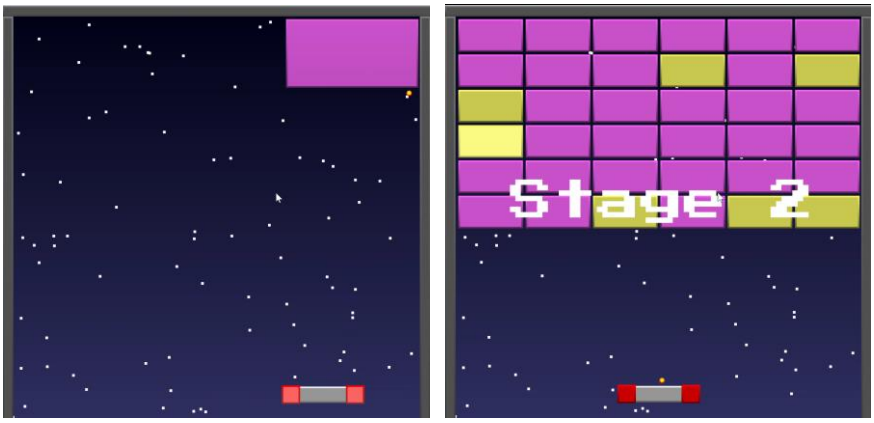
Game Scene

게임 타이틀과 깜빡이는 안내문 출력, 스페이스바 입력 시 게임 씬으로 이동한다

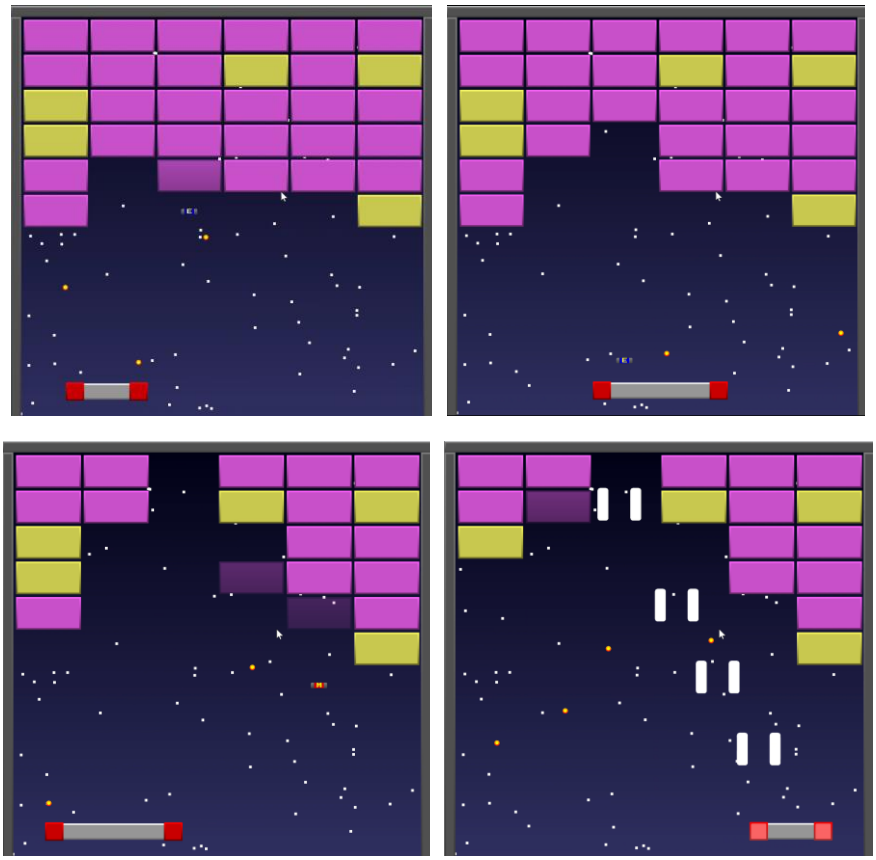
Game Scene



스테이지 시작 시 3 초동안 대기 후 게임이 시작하며, 패들을 좌 우 버튼으로 움직여 공을 튀겨 블록을 파괴해야 한다. 노란 블록은 파괴 시 공이 3 개로 증가한다

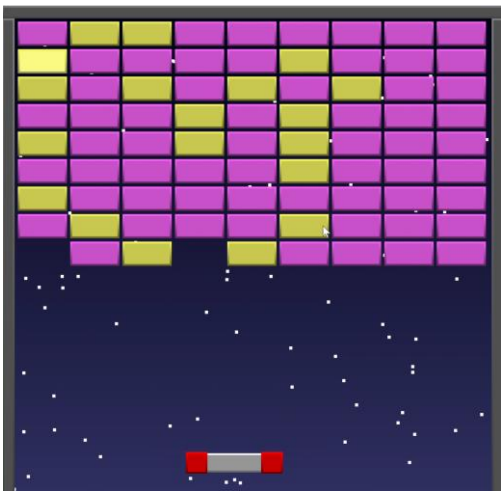


블록을 전부 파괴했을 경우 다음 스테이지로 변경된다.



Enlarge 아이템을 획득할 경우 패들 길이가 증가함.

Missile 아이템을 획득할 경우 스페이스 바로 미사일을 발사하여 블록 파괴 가능



공이 전부 사라지면 게임 오버 처리.

게임 오버 화면에서 스페이스바 입력 시 다시 타이틀 화면으로 복귀.

II. 개선할 점

게임을 참고할 때 알카노이드를 제일 많이 참고하였는데 다양한 아이템을 더 많이 만들지 못하여 아쉽다. 알카노이드의 Doh 와 같은 보스 스테이지도 만들면 더 게임성이 좋아질 것 같다.

그리고 미사일을 잘못 발사하면 오디오 문제로 게임이 멈추는 경우가 있는데 이를 해결하지 못하였다. 이를 해결하여 좀 더 매끄러운 게임을 만들고 싶다.

객체지향적으로 설계를 해야 하나 리스트를 2 개로 나눈 시점에서 코드가 많이 더러워졌고 balls 에 저장된 객체들의 역할이 너무 무거워졌고, 객체 필드를 직접 참조하는 일이 많아졌다. 설계 당시에는 두개로 나누는 것이 편해보였지만 다 구현을 한 지금 리스트를 하나로 통일하지 못해 아쉽다. 객체지향적인 코드를 잘 설계하기 위한 노력을 더 해야겠다고 생각했다.

III. 후기

비록 완전 객체지향적인 코드를 작성하지는 못했지만 객체를 기반으로 코드를 작성하다 보니 몇백줄 짜리 코드에 다른 기능을 추가할 때 시간이 기하급수적으로 단축되었다 게임을 위한 라이브러리가 아닌 스윙 라이브러리를

이용해 게임을 만들고 나니 다른 어떤 라이브러리나 프로그램으로 무언가를 만드는데 더 자신 있게 도전할 수 있을 것 같다.