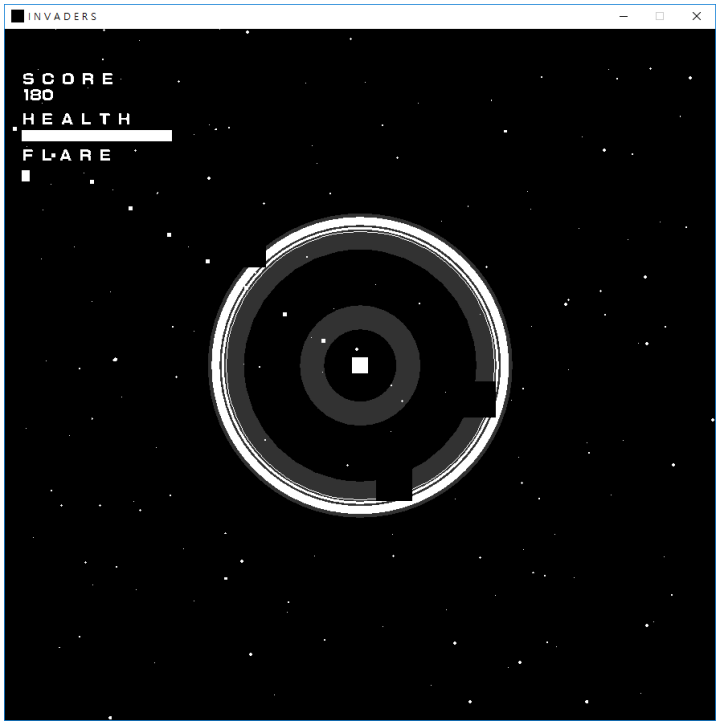
-

JAVA TOP-DOWN SHOOTING GAME

-

INVADERS



<조작법>

F Key: 총알 발사

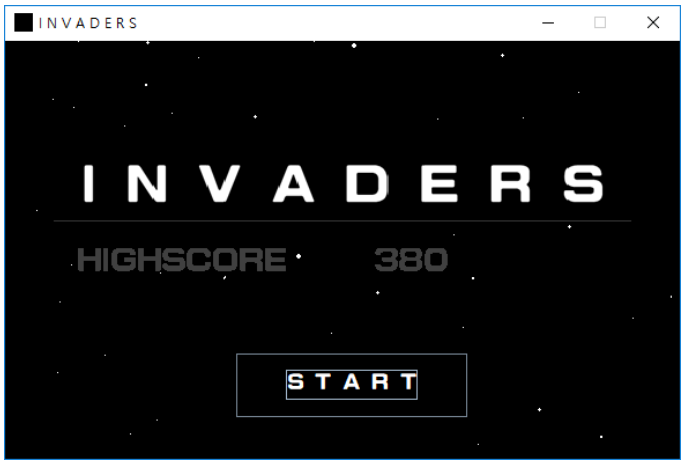
<- / -> : 발사 각도 조정

Space Key : 아이템 사용

보이지 않는 적을 탐색하고 격추하여 높은 점수를 받자!

1. 실행 화면

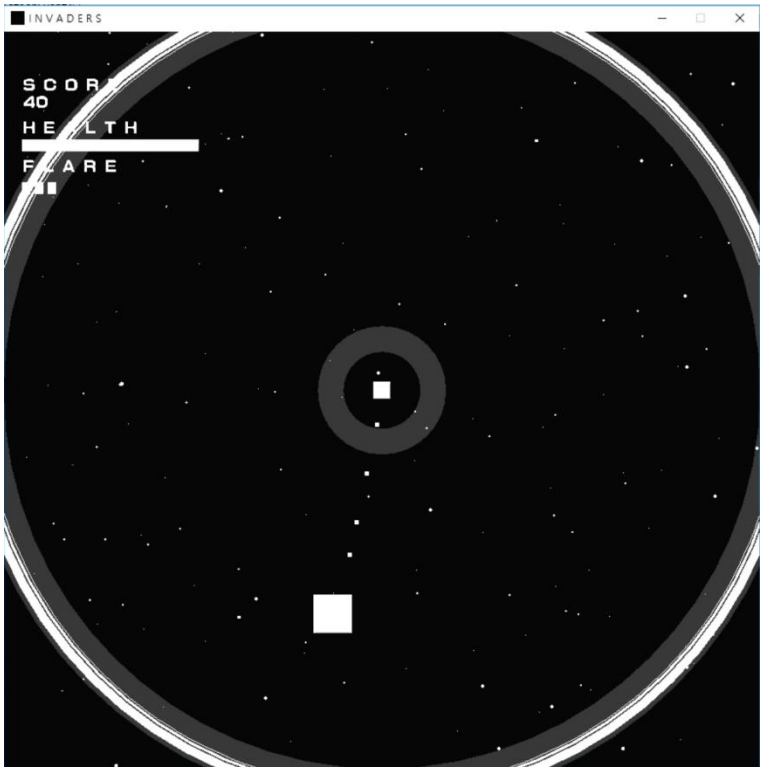
A. 타이틀 화면



타이틀 화면은 Java를 실행하면 가장 먼저 실행되는 창이다. 게임을 시작하기 전 게임의 제목과, 점수를 보여주고, START 버튼을 클릭하여 메인 화면을 실행 할 수 있다. .

- 게임 이름
- 최고기록
- 메인 화면 실행 - START 버튼

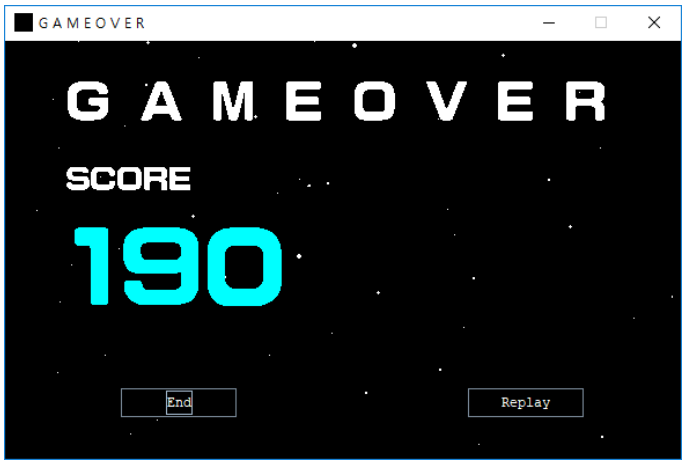
B. 메인 화면



메인 화면은 START 버튼을 누르면 실행되는 게임이 진행되는 창이다. 화면에는 점수, 체력, 아이템 개수 (Score, Health, Flare) 등 플레이어의 정보와, 배경화면, 적기, 플레이어의 위치, 발사되는 총알과, 몇 개의 동심원으로 표현되는 레이더 이펙트가 표시된다.

- 플레이어 정보 (점수, 체력, 아이템)
- 배경
- 레이터 이펙트
- 총알
- 적기
- 충돌 범위

C. 종료 화면



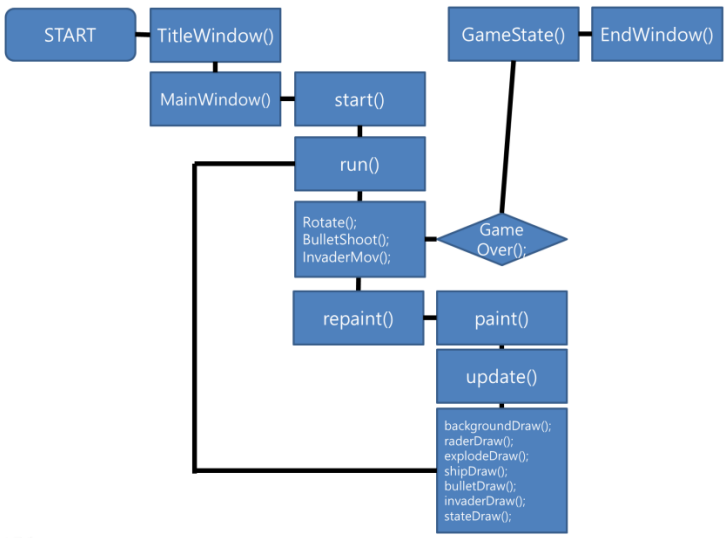
플레이어의 체력이 0이 되면 메인 화면이 닫히고, 종료 화면이 팝업 된다. 종료 화면은 게임오버 메시지와 점수를 보여주고, End버튼을 눌러 프로그램을 종료하거나, Replay 버튼을 눌러 타이틀 화면을 다시 팝업 시켜 게임을 재 진행 시킬 수 있다.. 점수는 파일에 저장되어있는 지난 최고 점수와 비교하여 현재 달성한 점수가 더 높다면 이를 파일에 기록한다.

■ 게임 오버 메시지

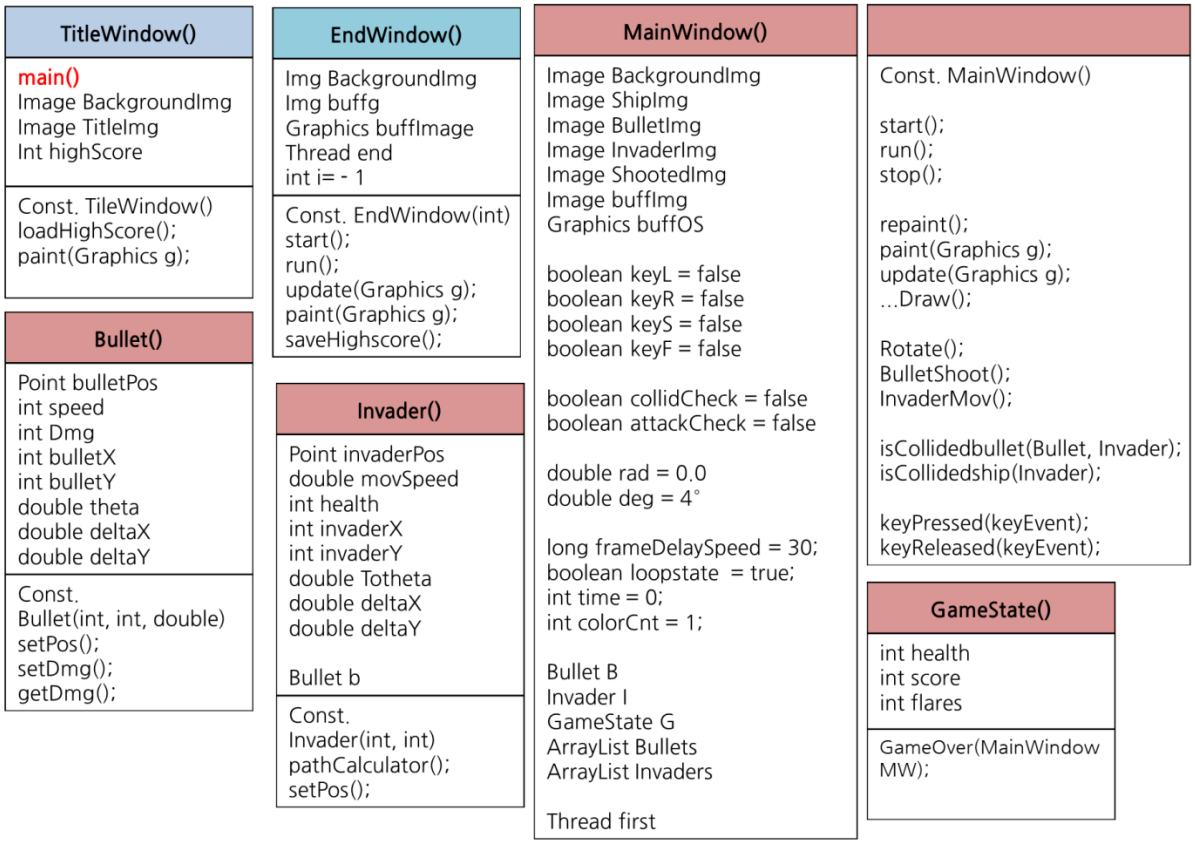
■ 점수 (점수를 표현 할 때, 계수기처럼 연속적으로 올라가는 효과를 Thread 지연을 사용하여 표현함)

■ End 및 Replay 버튼

2. Flow chart



3. 클래스차트



■ TiltleWindow class 내부의 main()에서 TitleWindow 객체를 성하여 프로그램이 동작한다.

■ ...Window class 들은 JFrame을 상속 받는다.

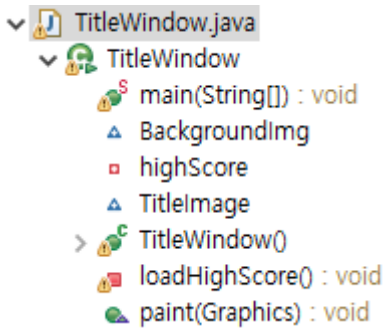
■ 상속 관계에 있는 클래스는 없다.

■ TitleWindow는 MainWindow를 MainWindow는 EndWindow를 종료시 호출 한다.

■ Bullet, Invader, GameState class들은 MainWindow에 객체 생성을 통해 fleid와 method에 접근하고, MainWindow에서 직접 사용된다. 변경된 값들을 각 클래스 객체에 저장되고, constructer를 통해 값을 전달 받는다.

4. 코드 설명

A. `public class TitleWindow extends JFrame`



타이틀 화면을 보여주는 클래스. JFrame을 상속하여 GUI를 표현한다. 메인 함수를 표함하고 있으며,

```
public static void main(String ar[]) {
	TitleWindow TW = new TitleWindow();
}
```

를 통해 TitleWindow 오브젝트 TW를 생성하여, 자바 실행 버튼을 눌렀을 때 가장 먼저 실행되는 클래스이다. `public TitleWindow()` {...} 컨스트럭터를 통해 창의 기본적인 틀을 마련하고

`private void loadHighScore()`메서드는 FileInputStream과 Scanner를 통해 save 디렉토리에 저장된 HighScore.txt 파일의 값을 읽고 창에 `public void paint(Graphics g)` 를 통해 화면에 그려준다.

`JButton startBT` 을 화면에 `getContentPane().add(startBT);` 로 추가해준 뒤, ActionListener를 통해 버튼을 누르면 `MainWindow MW = new MainWindow();` 와 같이 `MainWindow` 객체 `MW`를 생성해준다. 이후 `dispose();`로 현재 창을 닫아준다.

B. `public class MainWindow extends JFrame implements Runnable, KeyListener`

메인클래스에서는 JFrame을 상속받고, Runnable과 KeyListener method를 Override 합니다.

28 ~ 32 : 클래스에서 사용할 Image 변수를 지정해 줍니다. ImageIcon의 getResouse().getImage를 통해 파일에서 이미지를 얻고 이를 ImageIcon 형식으로 받아 Image 변수에 저장합니다.

33 : 더블 버퍼링에 사용할 Image 변수와 Graphics 변수를 지정해 줍니다.

34 : 클래스에 사용되는 폰트 지정

36~ 48 : 클래스 method 에 사용되는 Boolean 값, int 값 초기화

50~55 : 클래스에 사용되는 객체 생성을 위해 각 클래스의 객체 생성 및 관리를 위한 배열 생성, 스레딩을 위한 first 스레드 선언.

57~ 68 : new MainWindow()로 클래스 객체 생성시 호출되는 Constructor 설정. JFrame의 method를 사용하여 화면에 표시할 GUI를 꾸며줍니다. 이후 start() method를 호출하여 스레딩 제어를 시작합니다

`start()`: MainWindow 객체에 KeyListener를 추가하고, 위에서 선언한 first 스레드를 MainWindow의 스레드로 지정합니다.

`run()`: 스레딩을 제어하기 위해 while문을 선언하고 loopState Boolean 값으로 루프를 .제어합니다. 한 루프당 Rotate, BulletShoot, InvaderMov method와 repaint 로 화면에 표시되는 내용을 수정하고, GameState 클래스의 객체 G의 GameOver method를 호출하여 플레이이어의 체력을 검사합니다.

<GameState>

```
public void GameOver(MainWindow MW) {
	if(health <=  0) {
		MW.stop();
		MW.dispose();
		EndWindow EW = new EndWindow(score);
	}
}
```

<MainWindow>

```
public void stop() {
	loopState = false;
}
```

위와 같이 만약 0과 같거나 낮다면 현재 MainWinodw의 스레딩을 중단, 창을 닫고, 현재 점수를 EndWindow에 Constructor를 통해 전달합니다.

루프를 돌 때마다 `time++;` `Thread.sleep(frameDelaySpeed);`를 통해 게임의 객체를 제어하기위한 카운터 time을 수정해주고, 화면에 내용이 표시될 수 있도록 스레딩 속도를 조절합니다.

`repaint()/paint()/update()`: 이중 버퍼링을 통해 화면에 새로운 그림을 그릴 때 화면의 깜빡이는 현상을 해결합니다. Update() method에서 Graphics 를 통해 Graphics 객체인 buffOS에 먼저 그림을 그려줍니다. 루프가 한번 돈 후 이후 메모리에 저장된 이미지를 덧씌워 수정된 이미지가 깜빡임 없이 출력되도록 합니다.

`backgroundDraw()`:배경화면을 받아 buffOS에 그려줍니다. `clearRect(0, 0, 900, 900)`로 화면을 전체를 지워서 초기화 합니다.

`raderDraw()` 고수준의 이미지 제어를 위해 Graphics buffOS를 Graphics2D로 캐스팅 해줍니다. 이후 time 카운터를 사용하여 시간에 따라 달라지는 레이더 이펙트를 그려줍니다.
화면 전체를 흰색으로 밝혀주는 flare 아이템의 사용을 위해, flare의 남은 개수가 1개보다

많고(>=100), 스페이스바가 눌렸을 때의 동작을 설정합니다. 스레딩 중 if문이 반복 실행될 때 투명도를 점점 높이는 쪽으로 수정할 수 있도록 설정해 주었습니다. if 문이 끝나면 객체 G의 아이템(flares)의 값을 줄여줍니다.

bulletDraw():for 반복 구문을 사용하여 ArrayList에 저장된 Bullet 클래스 객체들에 접근 하여 화면에 그려줍니다.

InvaderDraw():위의 bulletDraw()와 같은 방식으로 동작합니다. Invader class의 I.shooted 에 접근하여 피격 시 피격이미지를 대신 그립니다.

stateDraw():GameState의 값들을 플레이어가 확인 할 수 있게 그립니다. .drawRect를 이용해 시각적으로 나타냅니다.

Rotate():방향키의 Boolean 값을 얻어 라디안 각도값을 수정합니다. 양 음에 제한이 없기 때문에

```
if(rad < -6.28319 || rad > 6.28319) {
	rad = 0;
}
```

의 if 문을 통해 360도가 약 6.28319인점을 통해 rad 값을 초기화 해줍니다.

위의 run()에서 루프를 돌면서 다시 호출 될 때마다 현재 생성된 모든 객체에 접근하여 값을 수정해줍니다.
BulletShoot():총알의 발사와 이동에 관련한 method 입니다. F키가 수정하는 Boolean 값에 따라 Bullets 객체에 하나하나 접근 할 수 있도록 ArrayList에 Bullets 객체 B를 추가해 줍니다. setPos() method를 통해 각각의 위치 값을 조정해줍니다. 화면 밖으로 나가 보이지 않는 총알은 삭제 처리를 해줍니다. 이후 Invader의 ArrayList와 같이 for문을 탐색하면서 isCollidedbullet(B, I)함수를 호출하여 적기와 총알의 충돌의 계산합니다. 총알을 배열에서 제거합니다. 적기의 체력이 모두 소진되면 점수를 10점, 아이템 점수를 5점얻고 적기를 배열에서 제거합니다.
InvaderMov():위와 동일한 방식으로 Invader의 객체 I의 값을 수정해줍니다.
isColliedship(I)함수를 통해 플레이어와 부딪혔는지 조사합니다.
이후 만약 time 카운터가 250의 배수가 되면, 객체를 가장자리의 랜덤한 좌표위치를 할당해주고, 배열에 추가합니다. 이후 카운터를 0으로 되돌립니다.

286 ~ : x,y 좌표값, Poistion 값에 기반한 충돌함수 2개와 KeyListener로 구성.

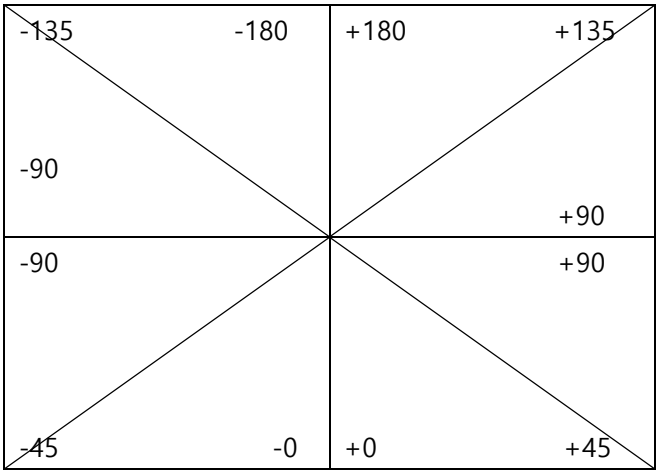
C. GameState 클래스에서는 게임의 점수와 플레이어의 체력, 플레이어의 아이템의 개수를 관리합니다. 이때 플레이어의 체력이 0 이하로 떨어진다면 stop()을 통해 loopState를 수정하여 스레딩을 중단하고, 창을 닫습니다. 이후 interrupt()를 통해 스레드를 안전히 종료하고, EndWindow 객체를 생성합니다. 이를 실행하지 않았을 시 점수가 계속 오르고, EndWindow 객체가 무한히 생성되는 오류가 있었습니다.

D. Bullet 클래스에서는 총알의 이동을 관리해 줍니다. 총알이 날아가야 할 각도를 Rotate()에서 수정된 rad 값으로 Bullet(int bX, int bY, double rad)를 통해 받고, 포인트 객체를 설정하여 Bullet 객체의 위치에 접근할 수 있도록 합니다. 이후 다음 식을 이용하여 화면에 한번 표시 될 때마다(update()가 한번 호출 될 때마다) 이동해야하는 단위 거리를 계산하고, setPos()를 통해 계속 수정해줍니다

```
theta = rad * 57.2958;
```

```
deltaX = (double) Math.sin(rad) * speed;
deltaY = -(double) Math.cos(rad) * speed;
```

E. Invader 클래스에서는 랜덤하게 생성된 Invader 객체의 체력과 격추상태, 이동을 관리해 줍니다. 객체가 이동해야 할 각도를 TOtheta각을 구하여 pathCalculator() method를 통해 단위 이동 좌표를 구하고, setPos() 를 통해 좌표를 이동시켜 줍니다. 다음은 TOtheta의 각도 값에 따른 단위 이동 좌표의 부호값을 결정하기 위해 나타낸 GUI 좌표 평면입니다. 왼쪽 위 모서리가 (0,0), 오른쪽 아래 모서리가(900,900)입니다.



F. `public class EndWindow extends JFrame implements Runnable`

EndWindow 클래스에서는 Runnable을 Override하여 스레딩 제어를 통해 계수기처럼 점점 올라가는 점수표시 효과를 보여줍니다.

```
public void run() {
	while(true) {
		repaint();
		i++;
		if(showScore == i) {
			break;
		}
		try {
			Thread.sleep(3);
		} catch (InterruptedException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}
	saveHighScore();
}
```

Break를 통해 무한 루프에서 탈출하고, saveHighScore();로 파일에 최고점을 기록합니다.