1. 각 클래스 명세

1) BlockBreaker

- 프로그램의 시작점인 메인클래스입니다.

- 스페이스바를 눌렀을때 changeScene( )을 통해 씬 전환을 담당합니다.

- 키 리스너를 통해 라켓 조작을 담당합니다. keyboardPressedThread 를 이용하여 기존 KeyPressed 이벤트보다 딜레이를 작게 설정함으로서, 그냥 이벤트리스너를 사용할때보다 input delay를 줄였습니다.

- Play( )에는 AudioSystem을 통한 오디오 입력 메서드가 정의되어 있습니다.

2) Scene

- 모든 Scene들의 부모 클래스이자, 추상 클래스입니다.

- 모든 씬이 공유하는 배경화면의 렌더링을 담당하고 있습니다.

3) SceneStart

- 첫번째 씬입니다. 게임 시작시 해당 씬으로 시작합니다.

- setLayout(null)과 setBound를 통해 각 label의 위치를 지정하였습니다.

- titleAnimationThread 는 타이틀 화면의 줌인 / 줌아웃 애니메이션을 담당하는 쓰레드입니다.

- textBlinkThread는 하단 메시지를 깜빡거리게 하는 쓰레드입니다.

4) SceneGame

- 두번째 씬입니다. 게임에 관한 대부분의 작업을 담당하는 클래스입니다.

- 처음에는 게임에 관한 여러 정보와 렌더링 화면을 초기화합니다.

- blockCollisionDetector는 폴링 방식으로 모든 block에 대한 충돌을 감지합니다.

- processCollisionBallAndBlock 을 통해 공이 블럭에 어느 부분에 맞았냐에 따라 진행 방향을 바꾸고, 블럭을 없앱니다.

- 공이 블럭과 충돌했을 때 해당 블럭이 스페셜 블럭이라면 splitBall( )을 통해 공을 세개로 나눕니다. 기존 공을 삭제하고, 새로운 공 세개를 만듭니다. 그리고 각 공에다가 기존 공의 MovingContext를 적용시키고 각도는 25도의 차이를 두었습니다. 이 작업은 ballArr 벡터에 add하는 작업을 수반하므로 synchronized 를 통해 다른 ballArr을 사용하는 쓰레드들과 동기화를 시켜주었습니다.

- 블럭의 수가 0이 되면, setNextStage( )를 통해 블럭의 수를 늘리고 새로운 스테이지를 만듭니

다. 이 과정에서 ballArr이 비게 되는데 이러면 GameoverThread가 게임을 종료시키므로 nextStageFlag를 통해 그런 상황을 방지하였습니다.

- wallCollisionDetector는 공과 벽의 충돌을 감지하고 공을 반대 방향으로 보냅니다.

- barCollisionDetector는 공과 라켓의 충돌을 감지하고 getBallDegInCollisionWithBar( ) 을 통해 공의 방향을 바꿔줍니다. 공의 x값과 라켓의 x값을 빼면 공이 라켓의 좌측으로 부터 떨어진 x값이 나오고(0~150) 이 값을 각도로 하여 공을 반대방향으로 날립니다. 일정한 값이 더해진 이유는 컴퓨터 상의 좌표계는 180~360이 위로 날리는 각도이기 때문입니다.

- ballEnterIntoLowerBoundDetector는 공이 아래로 떨어졌을 때 공의 개수를 감소시키는 역할을 합니다.

- GameoverDetector는 공의 개수가 0개가 되었을 때 씬을 전환시켜 게임을 종료시키는 역할을 합니다.

5) SceneOver

- 마지막 씬입니다.

- 점수를 출력하고, 최고 점수를 갱신합니다.

- getResourceAsStream( )을 통해 이미지를 입력받습니다.

6) Ball

- 공 클래스입니다.

- 공의 x, y, deg(진행 각도), dist(한번에 이동하는 거리) 등에 대한 정보를 가지고 있습니다.

- ballMoveThread를 통해 공이 이동합니다. 공은 BALL_REPAINT_DELAY마다 dist만큼 이동합니다. 공의 이동좌표는 각도와 이동 거리를 통해 구해집니다.

7) Bar

- 라켓 클래스입니다. 라켓에 대한 정보를 가지고 있습니다.

- 라켓은 백그라운드가 그려집니다.

8) Block

- 블럭 클래스입니다. 블럭에 대한 정보를 가지고 있습니다.

- 스페셜 타입의 블럭은 Blink하며, 다르게 그려집니다.

- stage마다 블럭의 scale이 달라집니다. (row와 column이 3배수로 늘어납니다.)

9) Wall

- 벽 클래스입니다.

10) Variable

- 글로벌 변수 및 함수를 정의하는 클래스입니다.

- 이 게임에 설정(config)창이 존재한다면 여기있는 값들을 통해 옵션을 조작할 수 있습니다.

2. 어려웠던 점

1) 벽과의 충돌 감지

- wallCollisionDetector를 비롯하여 충돌 감지 쓰레드에서는 원래 모두 intersect( )를 사용하였습니다. 그런데 이상하게도 intersect가 자꾸 중복으로 처리되어서 공이 어떤 곳에 부딪히면 무한히 방향이 바뀌며 진동하는 현상이 발생했습니다. 그래서 flag를 두거나 delay용 thread를 만들었는데도 역시 효과적이지는 않았습니다. 결국 intersect가 아닌 단순히 다른 물체와 공의 x, y 값 비교를 통해서 충돌 처리를 하고, 공의 프레임당 이동 거리를 줄여서 detecting resolution을 조금 더 높이니 그제서야 다른 부분은 해결이 되었는데, wallCollisionDetector 만큼은 끝내 해결하지 못했습니다. 그래서 아직도 가끔씩 벽에 공이 부딪히면 공이 벽 너머로 날아가버리는 현상이 발생합니다.

2) 계층 구조의 모호함

- 스윙에는 add와 remove라는게 존재해서 부모랑 자식 관계가 조금 혼란스러웠습니다. 그리고 생성자에서만 작업을 해야되는 것도 좀 어색했습니다. 그러다보니 Variable이라는 이름도 이상한 static 클래스를 만들어서 이상한 방식으로 변수를 공유하게 되었습니다.

3) jar 파일 생성

- 이미지와 사운드를 jar 파일에 적용시키기 위해서 많은 구글링이 필요했습니다. 결과적으로는 stream을 이용해서 입력받아 처리하는 것이었습니다.

4) 쓰레드의 사용

- 쓰레드가 점점 늘어나면서 어려웠던 점은 첫째로 프로그램의 부하가 점점 커진다는 것이었고, 둘째로는 동기화 문제였습니다. 특히 동기화 문제는 프로그램이 커질수록 더욱 어려워질 것 같습니다.

5) 소멸자의 부재

- C++에서는 소멸자를 통해 객체가 없어질 때 해야할 일을 부여해줬는데, 자바에는 소멸자가 없어서 객체를 다루기가 조금 더 까다로웠습니다.

3. 보완할 부분

1) blockCollisionDetector

전체 block의 충돌을 감지하게 하는 기존의 방식은 많은 딜레이를 발생시켜서 특히 다음 스테이지로 넘어갈 때 에러를 발생시킵니다. 더 많은 쓰레드가 필요하겠지만, 각 블럭마다 detector를 두는 방식이 더 나은 것 같습니다.

또한 쓰레드 안에서 볼이 나눠지는 부분이나, 다음 스테이지로 넘어가는 부분 등 너무 많은 작업을 담당하고 있어서 분리가 필요해보입니다.

2) processCollisionBallAndBlock

조건문이 불필요하게 복잡하여서 리팩토링이 필요합니다.

3) getBallDegInCollisionWithBar

공식이 부정확합니다.. 그리고 각도의 개념과 기울기 값을 어중간하게 섞어써서 잠재적 위험이 존재합니다.

4) GameoverDetector

씬이 전환되도 쓰레드가 종료되지않고 넘어갑니다. 때문에 게임을 계속 다시 실행할수록 쓰레드가 누적될 수 있습니다.

5) 플래그의 남용

플래그를 안쓰고 해결해야 하는데, 자꾸 플래그에 의존하다보니 점점 프로그램이 꼬이는 상황이 발생하였습니다.

6) 긴 조건문

조건문을 boolean을 리턴하는 함수로 모듈화시키면 좀 더 깔끔해질 것 같습니다.

7) 잔여하는 쓰레드

씬이 전환되도 씬 컴포넌트만 사라지고 쓰레드들은 남아있어서 계속 씬을 전환하다보면 쓰레드가 누적되어 위험을 초래합니다.

8) 네이밍 컨벤션

똑같은 기능을 하는데도 클래스마다 이름이 다르게 지어지거나 형식이 다른 것들이 있어서, 코

드를 읽을때 혼란이 가중될 수 있습니다.

9) 중복 코드

조건문을 중복하여 쓴 경우가 많아서, 코드를 수정하는 경우에 더 많은 부분에 신경써야 합니다.

10) enum 사용

direction이나 type 등에 매직넘버를 사용했습니다. 이를 알아보기 위해서는 주석에 의존해야 하므로 enum을 통해 의미를 부여해야 합니다.

4. 느낀 점

이번 과제에서 가장 크게 느낀 점은 설계의 중요성입니다. 개념에 대해서 잘 이해하지 못하거나, 급하게 기능을 추가하다보니 잘못된 방식으로 코드를 작성하게 되었고 그게 쌓이다보니 생각보다 깔끔하지 못한 코드가 작성되었습니다.

이를 방지하기 위해서는 코드를 짤 때 기능 구현에만 초점을 맞추지말고 최대한 넓은 시야에서 보며 신중하게 짜는 습관이 중요하다고 느꼈고, 설계 원칙에 위반되는 방향으로 밖에 짤 수 없을 때에는 차라리 구현하지 않는게 나은 것 같습니다.

규모가 큰 프로젝트를 설계해야 될때 이런 실수를 했다가는 돌이킬 수 없는 일이 벌어지므로, 클린 코드에 대한 책을 읽고 잘 설계된 오픈소스를 보며 다음에는 좀 더 잘 설계된 프로그램을 만들어야 겠습니다.