스프라이트 처리: 물리 법칙 응용의 기초

이번 장에서 배울 내용

- 게임에서 쓰이는 스프라이트 처리
- 게임 조작용 인터페이스로 스프라이트 적용하기
- 물리 법칙의 응용

이 장에서는 완전한 애플리케이션이나 예쁜 디자인 기법 대신 스프라이트와 앱 인벤터 애니메이션 컴포넌트의 고급 응용 방법에 대해 배우게 될 것이다. 적용하는 블럭 수만 보면 애플리케이션 하나를 충분히 만들고 남을 양이지만 개념 소개인 만큼 완벽히 동작하는 애플리케이션을 만들지는 않을 것이다.

현실적 움직임 구현을 위한 기법 비교

첫 번째 방법은 단순한 벽돌깨기 스타일의 앱을 예로 들겠다.

> **일러두기**
>
> 벽돌깨기(BreakOut)는 퐁(Pong) 다음으로 나온 가장 초기 컴퓨터 게임 가운데 하나다. 패들(짧고 편평한 막대 모양 도구)로 공을 받아서 공이 바닥으로 떨어지지 않게 하면서, 화면 상단에 몇 겹으로 쌓여 있는 블럭을 맞추어 깨는 게임이다. 이 장에서는 앱 인벤터로 이와 같은 효과를 내는 방법을 소개한다.

스프라이트가 한 물체와 부딪치면 단순히 진행방향을 반대로 바꾼다. 이렇게 하면 두 물체가 충돌할 때 발생하는 현상과 유사하게 되는데 실제로 두 물체의 충돌 현상을 물리 법칙으로 설명하려면 꽤 복잡하다. 이를 시뮬레이션하는 일은 대단히 복잡한 계산을 필요로 한다. 가속도, 중력, 관성, 마찰력, 그리고 속도를 계산하여 근사치를 구하는 일조차도 매우 복잡한 수학 연산을 수행해야 한다. 그러나 단순히 물체의 방향을 바꾸는 것만으로도 어느정도 실제 충돌 현상을 흉내 낼 수 있다. ImageSprite 블럭의 .Bounce 메서드가 그 일을 한다. .Bounce 메서드와 negate 블럭을 함께 써서 스프라이트의 진행 방향을 뒤집는 것으로 현실의 물리법칙을 반영한 움직임을 흉내 내는 것이 가능하다.

두 번째 방법은 첫 번째 것보다 훨씬 더 복잡하며, 예전에 고등학교 시절 물리 선생님께서 나중에 살다보면 틀림없이 필요할 것이라며 가르쳐주신 그 알 수 없는 수학식을 더 많이 쓴다. 두 번째 예는 물리 연산 엔진을 적용한 것으로 실제 동작하는 예를 들지는 않겠다(이 물리 엔진의 유용성을 보이기 위한 블럭도를 들어 설명할 예정이긴 하다). 이는 운동학(Kinematics)이라는, 움직임의 묘사와 모델링을 수학식을 써서 표기하는 수학의 한 분야를 활용한 것이다.

> **일러두기**
>
> 예제에서 사용한 공식을 모두 다 설명하지는 않을 예정이다. 그렇게 하려면 책 한 권을 다 차지할 분량일 뿐만 아니라 저자도 관련 수학적 지식이 충분치 않은 처지다. 그러나 걱정할 필요는 없는 것이, 다 이해하지 못한다 해도 무언가 얻을 것은 충분히 있기 때문이다. 대신 두번째 방법인 물체 상호작용 시뮬레이션을 좀 더 복잡한 애플리케이션의 핵심 요소로 쓰이는 고급 예제로서 제시하겠다. 운동학과 객체 모델링에 관련하여 자세한 자료가 소개된 책이나 웹사이트 (http://www.physicsclassroom.com/class/1dkin/u1l6a.cfm 페이지가 첫단계로 좋음)를 참고하기 바란다.

두 번째 방법인 객체 모델링의 기초를 다지고 나면 스스로 게임 등의 애플리케이션을 만들 블럭과 핵심부를 갖추게 된다.

그림 BC-1에 BreakDroid 디자인 스케치를 보였다.

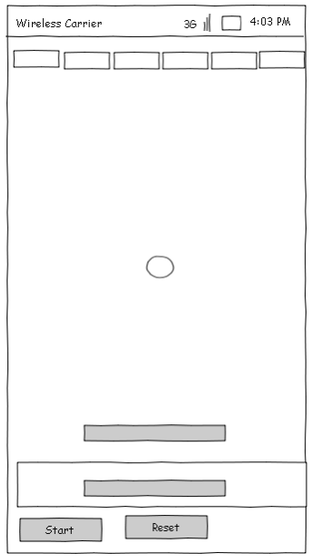

그림 BC-1 디자인 스케치

단위 목표

BreakDroid 디자인은 매우 기본적으로, 튕기며 돌아다니는 공의 움직임을 시뮬레이션하는 프레임워크를 만드는 것이다. 또한 스프라이트의 상호작용과 그 상호작용을 어떻게 활용해서 게임이나 다른 사용자 인터페이스를 만들어 내는가를 보게 해준다. 게임 내의 블럭은 공 스프라이트와 부딪치면 사라지게 된다. 점수 계산, 소리, 그리고 그 외 다른 요소들은 숙제로 남겨둔다.

- 블럭 스프라이트 한 줄
- 공 스프라이트 하나
- 충돌하는 스프라이트간의 상호작용을 처리하는 메서드
- 사용자가 조종 가능한 패들 스프라이트
- 스프라이트의 충돌 내역을 관리하는 메서드
- 게임 화면을 초기화하는 메서드
- 스프라이트 둘이 부딪칠 때 방향 전환을 처리하는 메서드
- 가장자리 중 일부만을 골라서 반동을 처리하는 메서드

새 컴포넌트

공(Ball) 컴포넌트가 프로젝트에서 처음 소개되는 유일한 컴포넌트다. 공 컴포넌트는 Animation 팔레트에서 끌어온 공처럼 생긴 단순한 스프라이트다. 다른 스프라이트와 구별되는 특별한 속성이나 특징은 없다.

- Ball

새 블럭

본 장에서 새로 소개하는 블럭은 다음과 같다.

- Negate
- Collide With

진행 단계

디자인 목표를 완수하고 BreakDroid 애플리케이션을 만드는데 필요한 논리적인 단계를 아래에 보였다.

1. 게임 플레이 캔버스를 만든다.
2. 패들 조종 캔버스를 만든다.
3. 블럭 스프라이트와 공 스프라이트를 만든다.
4. Start, Reset 버튼을 만든다.
5. 점수 기록용 프로시저를 만든다.
6. 충돌 이벤트를 처리한다.
7. 모서리 이벤트를 처리한다.
8. Start 버튼 이벤트를 처리한다.
9. 게임 초기화 프로시저를 만든다.
10. Reset 버튼 이벤트를 처리한다.

BreakDroid 시작하기

BreakDroid 애플리케이션의 인터페이스는 매우 단순하지만 스프라이트 처리 기법을 실험해 볼 수 있는 좋은 기회가 된다. 게임 플레이 인터페이스로 캔버스 컴포넌트를 두 개 사용한다. 첫번째는 블럭과 패들, 공 스프라이트를 위한 것이고 다른 하나는 패들을 조종하는 조종용 인터페이스용이다. TwiTorial 프로젝트에서 했던 것처럼 Screen1.Title 블럭에 점수 관련 정보를 표시한다.

1. 새 프로젝트를 만들고 이름을 BreakDroid로 짓는다.
2. 홈페이지에서 받은 프로젝트 파일에서 아이콘 파일을 꺼내 Icon 값을 설정한다.
3. Scrollable 속성을 끈다.

다음으로 게임 플레이용 인터페이스에 쓸 캔버스와 스프라이트를 모두 배치한다.

1. Canvas 컴포넌트를 끌어다 뷰어에 놓는다. Height, Width 속성을 모두 Fill Parent로 한다.
2. Animation 팔레트에서 ImageSprite 컴포넌트를 모두 여섯 개 끌어다 캔버스 위에 놓는다. 그림 BC-1처럼 캔버스의 위쪽에 나란히 정렬한다.
3. ImageSprite 블럭의 이름을 sprtBlock1에서 sprtBlock6까지 다 바꾼다. 총 여섯 개의 스프라이트가 이름이 순서대로 바뀌어 있어야 한다. 이 블럭들은 공 스프라이트와 충돌하면 사라지게 된다.
4. 프로젝트 파일에 들어 있는 block.png 파일을 Media 컬럼에 올려서 스프라이트로 쓸 수 있게 준비한다.

   웹사이트에서 프로젝트 파일을 받는 방법은 이 책의 서문을 참조한다.
5. 각 sprtBlock 블럭의 Image 속성을 block.png로 한다.
6. Ball 컴포넌트를 끌어다 캔버스의 가운데에 놓는다.
7. ImageSprite 블럭을 새로 하나 끌어다 캔버스 하단에 놓고 컴포넌트 이름을 sprtPaddle로 한다. Image 속성은 프로젝트 파일에 들어 있는 paddle.png로 한다. 이 스프라이트는 공을 받아서 튕기는 패들 역할이다. 곧 추가할 조정용 패들이 이 패들을 움직여서 공을 받아 다시 블럭 쪽으로 쳐올리는 역할을 한다.
8. Canvas 컴포넌트를 새로 하나 끌어다 Canvas1 컴포넌트 밑에 놓는다.
9. 방금 만든 Canvas2 컴포넌트의 Width 속성을 Fill Parent로 바꾼다. Height 속성은 40 픽셀로 한다.
10. BackgroundColor속성을 Grey로 바꾼다. 이렇게 하면 두 번째 캔버스가 구분되어 보인다.
11. 새 이미지 스프라이트를 하나 끌어다 두 번째 캔버스에 놓는다. 이름을 sprtControlPaddle로 바꾼다.

    이 스프라이트는 패들 조종을 위한 보조 스프라이트다. 게임 사용자가 게임 화면의 패들을 바로 누르고 끌어서 조종한다면 손가락이 화면을 가려서 보이지 않을 것이다. 따라서 조종은 이 보조 패들을 대신 써서 하고, 이 패들의 움직임대로 실제 패들을 그림자처럼 따라 움직이게 하려는 것이다.
12. 프로젝트 파일에서 가져온 paddle.png을 실제 패들과 마찬가지로 sprtControlPaddle에 적용한다.

다음으로 인터페이스용 버튼을 두 개 달자. Start 버튼은 공을 움직이기 시작해서 게임을 개시하는데 쓰고 Reset 버튼은 게임 화면을 초기화해서 새로 시작하게 한다. 앞서 언급한 바와 같이 이는 기본적인 기능만 갖추게 될 것이고 실제 할만한 게임을 만들려면 다른 처리와 수정이 필요할 것이다.

1. 가로배열(HorizontalArrangement) 블럭을 끌어다 두 번째 Canvas 컴포넌트 밑에 놓는다.
2. 버튼을 하나 끌어다 방금 만든 가로배열 블럭 안에 넣고 Text 속성을 Start로, 이름을 btnStart로 바꾼다.
3. 버튼을 추가로 하나 더 끌어다 가로배열 블럭 안에 넣고 이름을 btnReset으로 바꾼 후 Text 속성은 Reset으로 한다.

지금까지는 BreakDroid 애플리케이션용 기본 인터페이스 구성이었고, 흥미로운 부분은 지금부터다. 블럭 에디터에서 시작하자.

인터페이스는 그림 BC-2와 같아야 한다.

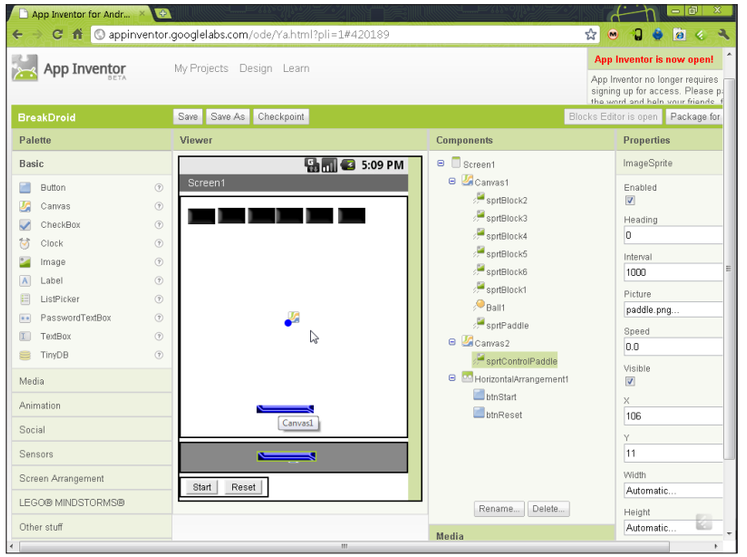

그림 BC-2 **BreakDroid 사용자 인터페이스**

다음의 단계를 밟아 스프라이트 상호 작용 처리 부분을 추가한다.

1. 블럭 에디터를 열고 Screen1.Initialize 이벤트 핸들러를 타입블럭한다.
2. Screen1.Title [to] 블럭을 타입블럭하고 이벤트 핸들러에 끼운다.
3. text 블럭을 타입블럭하고 값을 Press start and use lowest paddle to move game paddle(스타트 버튼을 누르고 아랫쪽 패들로 게임 패들을 조종하세요)로 바꾼다. 이 설명으로 사용자가 어떻게 인터페이스를 조종하는지 알려준다. 나중에 ScreenTitle 값 대신 사용자의 점수를 넣을 것이다.

공 스프라이트가 블럭 스프라이트와 부딪칠 때마다 스코어 변수를 증가시키고 점수를 새로 표시하며 부딪친 블럭을 없앤다. 점수와 충돌 이벤트를 처리하는 기본적인 내용이다. 게임의 종류에 따라 충돌 이벤트를 기록해 두고 나중에 확인해야 할 경우도 있다. 예를 들어 스프라이트가 몇 번 부딪쳤는지 횟수를 저장해 두었다가 수가 일정량을 넘으면 그때 스프라이트가 없어지는 게임도 있을 수 있다.

다음의 단계를 밟아서 procScoreIncrement 프로시저를 만든다. 이 프로시저는 블럭과 공이 충돌할 때마다 부른다.

본 게임에서는 단순히 스코어 변수를 증가시키고 Screen1.Title에 스코어를 보여주면 충분하다.

1. 변수를 새로 하나 정의하고 이름을 varScore로 한다.
2. 숫자 0 블럭을 타입블럭하고 varScore에 끼운다.
3. 새 프로시저를 하나 타입블럭하고 이름을 procScoreIncrement로 한다.
4. varScore [to] 블럭을 타입블럭하고 procScoreIncrement에 끼운다.
5. 덧셈 (+) 블럭을 하나 타입블럭하고 varScore 블럭에 끼운다.
6. varScore 변수 블럭을 타입블럭하고 덧셈 블럭의 첫번째 소켓에 끼운다.
7. 숫자 1 블럭을 타입블럭하고 덧셈 블럭의 두번째 소켓에 끼운다.
8. Screen1.Title [to] 블럭을 타입블럭하고 procScoreIncrement 프로시저 다음 자리에 끼운다.
9. join 블럭을 타입블럭하고 Screen1 블럭에 끼운다.
10. text 블럭을 타입블럭하고 텍스트 값을 BreakDroid Score로 바꾼 후 join 블럭의 첫 번째 소켓에 끼운다.

11. varScore 변수 블럭을 타입블럭하고 join 블럭의 두 번째 소켓에 끼운다.

procScoreIncrement 프로시저를 부를 때마다 varScore 변수 값이 하나 증가하며 스크린 타이틀 바에 그 값이 나타난다.

일러두기

타이틀 바를 없애는 방법은 현재 없다. 그곳을 정보 표시 용도로 활용하는 것은 화면을 효율적으로 쓰는 좋은 방법이다.

패들로 공을 받아 치지 못하고 놓치는 경우에는 초기화 프로시저를 불러 주어야 한다. 초기화 프로시저에서는 공을 특정 X/Y 위치로 옮기고 그 방향과 속도를 초기값으로 돌려 놓는다.

1. 프로시저를 새로 하나 타입블럭하고 이름을 procBallReset으로 한다.
2. Ball1.X [to] 블럭을 타입블럭하고 프로시저에 끼운다.
3. 숫자 143 블럭을 타입블럭하고 X 위치 블럭에 끼운다.
4. Ball1.Y [to] 블럭을 타입블럭하고 프로시저에 끼운다.
5. 숫자 169 블럭을 타입블럭하고 Y 위치에 끼운다.
6. Ball1.Heading [to] 블럭을 타입블럭하고 다음 위치에 끼운다.
7. 숫자 0 블럭을 타입블럭하고 Heading 블럭에 끼운다.
8. Ball1.Speed [to] 블럭을 타입블럭하고 다음 위치에 끼운다.
9. 숫자 0 블럭을 타입블럭하고 Speed 블럭에 끼운다.

procBallReset 프로시저가 불릴 때마다 공은 주어진 X/Y 좌표로 옮겨가고 속도와 방향이 0이 되어서 사실상 공이 화면에 멈추어 서버리게 되는 효과가 난다. Start 버튼을 누르면 공의 방향과 속도가 초기화된다.

공이 블럭 스프라이트와 부딪치면 점수만 올라가는 것이 아니라 공이 튕겨 나와야 한다. 이는 negate 블럭을 활용할 좋은 기회다. collide 이벤트가 발생하면 procBounce라는 프로시저를 불러서 공의 현재 방향을 간단히 반대로 만들기만 하면 된다.

negate 블럭은 수학 연산 블럭의 하나로 정수를 입력으로 받아서 부호만 반대인 숫자를 돌려준다. 예를 들어 180은 -180이 된다. 공의 현재 방향을 나타내는 변수와 함께 쓰이면 공의 진행 방향을 바꾸는데 유용하다. 즉 공이 반사되어 나오는 효과를 낼 수 있다.

1. 프로시저를 새로 하나 타입블럭하고 이름을 procBounce로 한다.
2. Ball1.Heading [to] 블럭을 타입블럭하고 프로시저에 끼운다.
3. negate 블럭을 타입블럭하고 Ball1.Heading 블럭에 끼운다.
4. Ball1.Heading 블럭을 타입블럭하고 negate 블럭에 끼운다.

procScoreIncrement, procBallReset, 그리고 procBounce 프로시저까지 만들어서 스프라이트 충돌 이벤트 처리용으로 필요한 부분을 모두 완성했다. 공 스프라이트와 블럭 스프라이트가 충돌 했을 때 블럭 스프라이트는 사라지고, 점수는 증가하고, 공이 튕겨 나오는 효과를 내고 싶은 것이다. .CollideWith 이벤트 핸들러도 같은 방법으로 만들면 된다. 각 sprtBlock 스프라이트는 다른 스프라이트와 부딪치며 이벤트를 발생시킨다. 이 이벤트를 이용해서 원하는 효과를 내면 된다:

1. sprtBlock1.CollideWith 이벤트 핸들러를 타입블럭한다.
2. procScoreIncrement 프로시저 call 블럭을 타입블럭하고 이벤트에 끼운다.
3. procBounce 프로시저 call 블럭을 타입블럭하고 이벤트 핸들러에 끼워 넣는다.
4. 다음으로 작업하는 스프라이트의 .Visible 블럭을 타입블럭한다. 이 경우 sprtBlock1. Visible [to] 블럭을 타입블럭하고 끼워 넣는다.
5. .Visible 블럭에 false 블럭에 끼운다.
6. 각 스프라이트 블럭용 .CollideWith 이벤트 핸들러에 대해서 위의 1-5 단계를 반복한다.

.CollideWith 이벤트 핸들러 설정이 끝나면 그림 BC-3과 같이 .CollideWith 이벤트 핸들러가 총 여섯 개가 된다.

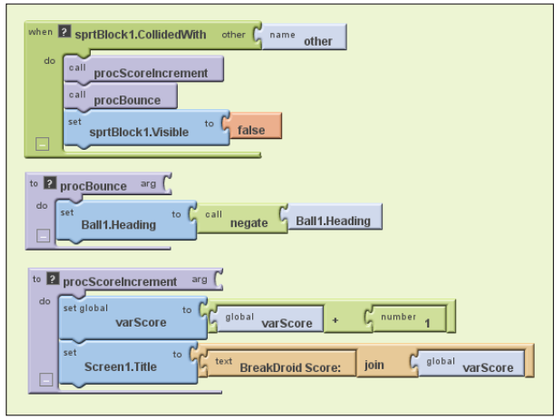

그림 BC-3 .CollidedWith 이벤트 핸들러. procBounce, procIncrement 프로시저가 삽입된 모습

다음 단계에서 만들 패널 조종용 메서드에서는 캔버스나 스프라이트 이벤트의 X/Y 좌표를 입력으로 써서 다른 스프라이트를 조종하는 중요한 기법을 배운다. 사용자가 Canvas2 상에서 sprtControlPaddle 스프라이트를 밀거나 당기면 그때 발생하는 이벤트의 X 좌표값을 얻어와서 실제 게임 패들인 sprtPaddle을 움직이는데 쓴다. 사용자의 손가락으로 게임 패들을 직접 조종하게 하면 화면이 가려서 보이지 않게 되기 때문에 이와 같은 기법을 쓰는 것이다. 이 방법이 손가락이 화면을 가리는 문제를 해결하는 유일한 방법은 아니다. 그러나 스프라이트/캔버스 이벤트로 애플리케이션의 다른 부분에 있는 컴포넌트를 움직이고 영향을 줄 수 있다는 점을 이해하는 것은 중요하다.

1. sprtPaddle.CollidedWith 이벤트 핸들러를 sprtPaddle 블럭 서랍에서 끌어다 놓는다.
2. procBounce 프로시저 call 블럭을 타입블럭하고 이벤트 핸들러에 끼운다.

이렇게 하면 공이 다시 블럭 쪽으로 반사되어 가게 된다.

벽돌깨기 스타일 게임의 주 목적은 공이 화면 바닥으로 빠지지 않게 하면서 블럭을 깨어 없애는 것이다. 지금까지의 작업으로 블럭을 없어지게 하는 것은 처리가 되었는데 아직 공이 화면 가장자리에 이르렀을 때 해당하는 처리가 남았다. 스프라이트가 캔버스의 가장자리에 이르면 .EdgeReached 이벤트가 발생하며, 이를 써서 필요한 처리를 하면 된다. 이벤트가 발생하면 어느 가장자리에 스프라이트가 와 닿았는지 정보도 함께 온다. 캔버스의 가장자리는 번호가 매겨져 있다(6장 AlphaDroid 프로젝트를 참조). 화면 위 가장자리는 1이며 바닥은 -1이다. 따라서 "어느 가장자리에 공이 이르렀나? 그리고 가장자리가 -1이면 무엇을 해야 하나?"와 같은 질문에 답할 로직을 만들어야 한다.

1. Ball1.EdgeReached 이벤트 핸들러를 타입블럭한다.
2. IfElse 블럭을 하나 타입블럭하고 이벤트 핸들러에 끼운다.
3. 등호 (=) 블럭을 타입블럭하고 IfElse 블럭의 test 소켓에 끼운다.
4. edge 값 파라미터 블럭을 타입블럭하고 등호 블럭의 첫번째 소켓에 끼운다.
5. 숫자 -1 블럭을 타입블럭하고 등호 블럭의 두번째 소켓에 끼운다. 여기서 숫자 -1은 캔버스의 아래쪽 가장자리를 뜻한다.
6. procBallReset 프로시저 call 블럭을 타입블럭하고 IfElse 블럭의 then-do 소켓에 끼운다. 공이 닿은 가장자리가 -1이면 볼의 상태가 초기화된다.
7. Ball1.Bounce 블럭을 타입블럭하고 IfElse 블럭의 else-do 소켓에 끼운다.
8. edge 값 파라미터 블럭을 타입블럭하고 Ball1.Bounce 블럭에 끼운다.

이제 조건식이 참이 아니면 공은 내장 함수 .Bounce를 부른다. .Bounce 함수는 의도와 기능 모두 procBounce 프로시저와 동일하다.

이제 남은 것은 버튼 이벤트 두 개다. Start 버튼은 공 스프라이트에 방향과 속도 초기치를 설정한다.

1. btnStart.Click 이벤트 핸들러를 타입블럭한다.
2. Ball1.Heading [to] 블럭을 타입블럭하고 이벤트 핸들러에 끼운다.
3. 숫자 250 블럭을 타입블럭하고 .Heading 블럭에 끼운다.
4. Ball1.Speed [to] 블럭을 타입블럭하고 이벤트 핸들러 다음 위치에 끼운다.
5. 숫자 10 블럭을 타입블럭하고 .Speed 블럭에 끼운다.

사용자가 btnStart.Click 버튼을 누르면 공이 바닥을 향해 왼쪽으로 비스듬하게 내려오게 된다.

이제 마지막으로 Reset 버튼을 처리하자. Reset 버튼은 없어졌던 블럭들도 다시 다 화면에 나타나게 하고 점수도 초기화한다. 또한 공의 위치도 procBallReset 프로시저를 불러서 초기화한다.

1. btnReset.Click 이벤트 핸들러를 타입블럭한다.
2. varScore [to] 블럭을 타입블럭하고 이벤트 핸들러에 끼운 후 숫자 0 블럭을 넣는다. 이렇게 하면 score 변수값이 0으로 초기화된다.

3. Screen1.Title [to] 블럭을 타입블럭하고 이벤트 핸들러 다음 위치에 끼운다.
4. join 블럭을 하나 타입블럭하고 .Title 블럭에 끼운다.
5. text 블럭을 하나 타입블럭하고 내용을 BreakDroid Score:로 바꾼다. 그리고 join 블럭의 첫번째 소켓에 끼운다.
6. varScore 변수 블럭을 타입블럭하고 join 블럭의 두번째 소켓에 끼운다.

다음으로 sprtBlock 블럭 모두 Visible 속성을 true로 한다. 현재 구현상으로는 게임이 끝나면 블럭들이 모두 안 보이게 된다. 새로 게임을 시작하려면 이 블럭들을 모두 다시 보이도록 바꾸어 주어야 한다.

1. sprtBlock1.Visible [to] 블럭을 타입블럭하고 이벤트 핸들러의 다음 위치에 끼운다.
2. true 블럭을 하나 타입블럭하고 .Visible 블럭에 끼운다.
3. 나머지 sprtBlock 블럭들에 대해서도 위의 1-2 단계를 반복한다. 그러면 그럼 BC-4와 같이 sprtBlock들이 모두 보이는 상태로 바뀌게 된다.

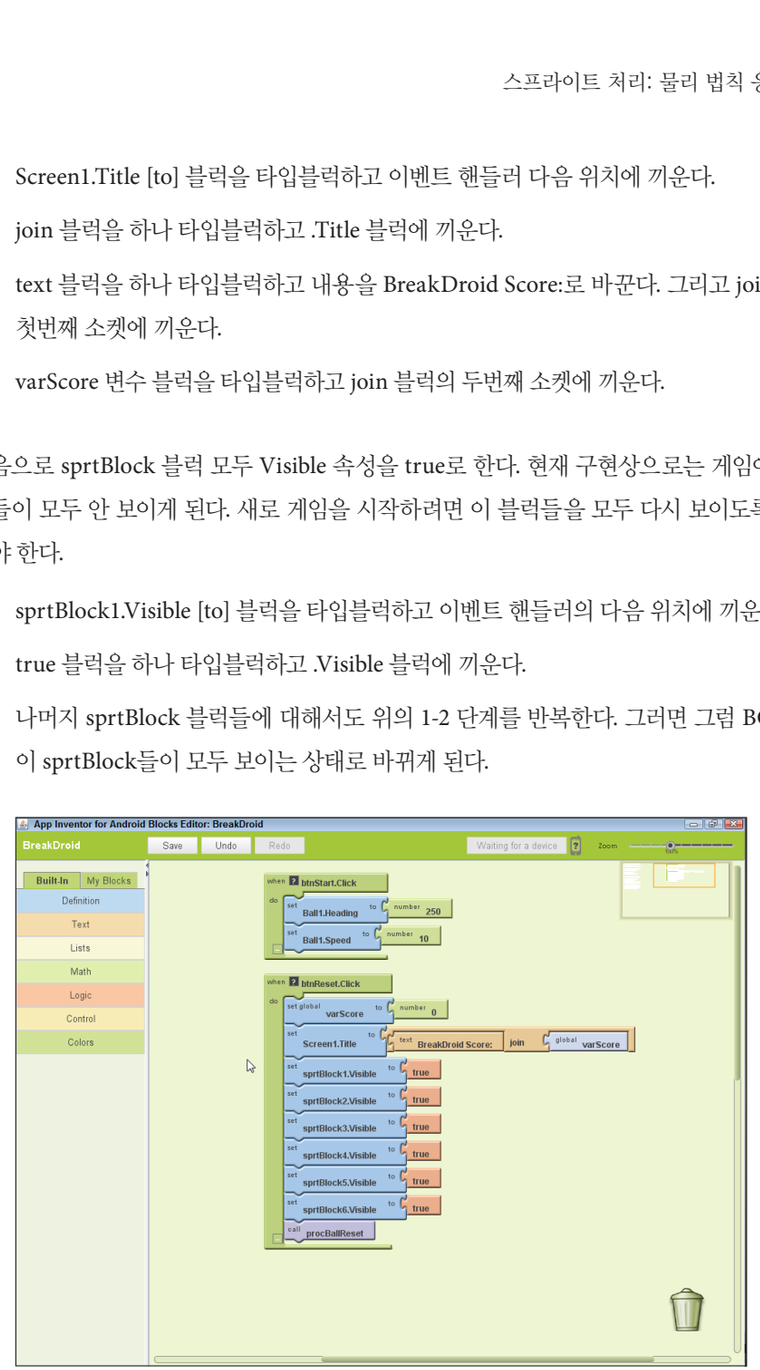

그림 BC-4 Reset 버튼과 Start 버튼 .Click 핸들러

4. procBallReset 프로시저 call 블럭을 타입블럭하고 이벤트 핸들러의 마지막 .Visible 블럭 아래에 끼운다.

이제 BreakDroid 애플리케이션에서 발생 가능한 이벤트는 모두 다루었다. 애플리케이션을 패키지로 만들어서 폰에 올리고 벽돌깨기를 한번 해보자(패키지 만들기와 설치 방법은 1장을 참고한다). 보면 알수 있듯 현재 완벽한 게임이라고는 할 수 없다. 아래 열거한 기능을 추가해서 좀더 완벽한 게임으로 만들어 보기 바란다.

횟수 제한 (공을 못 받아쳐서 아래로 빠지면 기회가 한 번 줄어듬)

화면을 여러개 쓰지 않고 게임 레벨 바꾸기(보이지 않은 벽돌을 많이 미리 만들어 놓고 게임 레벨이 올라갈 때마다 더 보여주는 방법도 있다).

- 공이 부딪칠 때 음향 효과 내기
- 게임 완료 시나리오
- 물리법칙 엔진 만들기

여기서 만들려고 하는 물리 법칙 엔진으로, 실제 게임을 만들지는 않을 것이다. 그러나 예제 프로그램이 프로젝트 파일에 들어 있으므로 먼저 웹사이트에서 받아 두기 바란다. .zip 소스 파일을 프로젝트로 올려서 동작하는 것을 볼 수도 있다.

일러두기

이 물리법칙 엔진에 쓰인 운동학 관련 복잡한 수식을 다 설명할 계획은 없다. 구글 앱 인벤터 포럼에 조쉬 터너씨가 물리 법칙 응용 데모 애플리케이션으로 핵심 로직과 블럭을 만들어서 공개했는데, 본 책에 그 내용을 인용할 수 있게 동의해 준데 대해 감사의 뜻을 표한다. 그 결과로 두개의 물체가 관성과 중력을 바탕으로 서로 상호작용 하는 것을 시뮬레이션하였다. 이는 다소 복잡한 수식을 기반으로 해서 정의한 다양한 수치로 물체의 방향, 속도, 그리고 좌표를 지속적을 바꿈으로써 구현한 것이다.

물체의 속도와 방향, 위치를 계속 변경하려면 타이머를 쓴다. 주어진 예제 애플리케이션에서 공 스프라이트는 움직임이 실제와 근사하게 되도록 상수를 몇개 정의해서 썼다. 그러나 실제 애플리케이션이라면 이러한 상수 대신 사용자가 입력한 값이나 스프라이트 간의 상호작용에 근거해서 변수를 썼을 것이다.

디자인

그림 BC-5에 운동학 엔진의 핵심 블럭과 수식을 보였다. 애플리케이션에 쓰이는 스프라이트의 움직임을 물리적으로 시뮬레이션하기 위한 것이다.

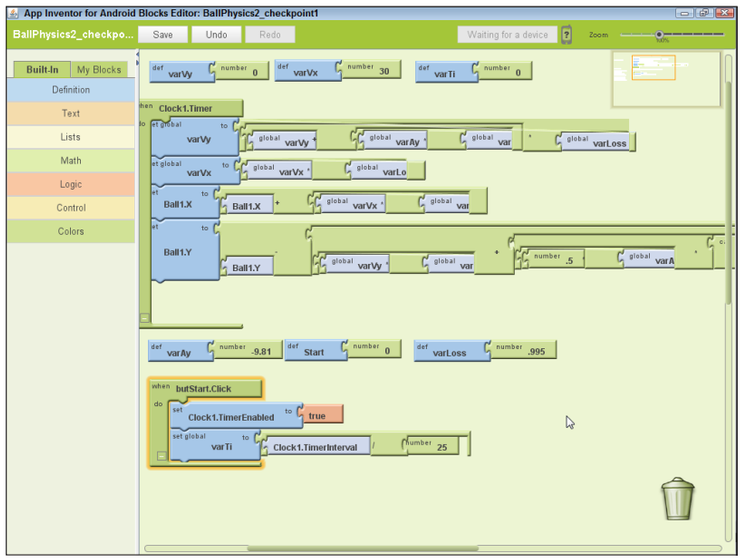

그림 BC-5 운동학 엔진 블럭의 완성도

단위 목표

운동학 엔진과 같은 물리 법칙 엔진의 기본적 목표는 아래와 같은 것들을 시뮬레이션하는 것이다.

- 물체간의 상호 작용
- 관성 효과
- 중력의 영향
- 직전 시간의 상태에 기초 속도와 방향의 변화

진행 단계

실제 애플리케이션을 만들려면 이 단순한 예제에서보다 훨씬 더 신경 쓸 일이 많다. 기본적으로 핵심부에 준하는 부분을 만들기 위한 기본 단계는 다음과 같다.

1. X, Y, 시간, 중력 등을 나타내는 변수를 정의한다.
2. 시간에 따라 주기적으로 스프라이트의 속도와 방향을 갱신하는 메서드를 만든다.

물리 엔진 개발 시작하기

우선 애플리케이션의 골격을 만들어야 하는데, 이는 캔버스와 공 스프라이트, 클럭 컴포넌트 등으로 이루어진다. 아래 과정은 물리 엔진을 개발하기 위한 것이며 이를 실제 애플리케이션에 적용하는 것은 독자의 창의력과 상상력에 달려 있다.

캔버스와 공 스프라이트, 그리고 공과 상호 작용하는 다른 스프라이트들을 만들었으면 이어서 운동학 엔진을 만든다. 먼저 다음의 변수들을 모두 정의한다.

varTi - 초기값은 Clock1.TimerInterval 값을 25로 나눈 것으로 한다(.TimerInterval/25)

일러두기

여기 보인 예제에서 varTi 변수는 Start 버튼 이벤트에서 초기화된다. 이는 .TimerInterval 속성을 Clock1.Timer를 가지고 조정하기 위한 것이다. 처리과정의 부하와 수식의 정확도를 조절하고자 한다면 Clock1.TimerInterval 값을 늘리거나 줄이면 된다. 하지만 원한다면 varTi 값을 상수로 정의해도 무방하다.

- varVy - 초기값은 0. 이는 Y축방향 초기 속도다
- varVx - 초기값은 30. X축방향 초기 속도다(이 초기값은 측면 방향으로 공을 움직여서 공이 반동하여 나오는 것을 보여주고자 한 것이다).
- varLoss - 초기값은 .995. 이는 마찰로 인한 운동의 감속효과를 나타낸다.
- varAy - 초기값은 -9.81. 초기 가속도를 나타낸다.

마지막 상수 두개는 물체나 스프라이트에 가해지는 외부의 힘을 나타내는 것이다.

변수들을 모두 정의했으면 Clock1.Timer 이벤트를 만든다. 처리 부하를 높여서 좀 더 부드러운 움직임을 보려면 5밀리초로, 대신 부하를 줄이려면 15밀리초로 한다.

처음 나오는 블럭 두개는 변수 Vy, Vx의 값을 설정한다. 이들 변수는 나중에 스프라이트의 X/Y 위치를 변경시키는 데 쓰인다. Vy 변수의 식은 다음과 같다.

```
(Vy + (Ay*Ti)) * varLoss
```

팁

위 수식에 대한 설명은 관련 웹사이트 www.physicsclassroom.com/class/1dkin/u1l6a.cfm를 참조한다.

Vy 변수와 위의 식을 쓰면 어떤 클럭 사이클 시점에서 외부에서 가해지는 힘을 계산할 수 있다:

1. Clock1.Timer 이벤트 핸들러를 타입블럭한다.
2. varVy [to] 블럭을 타입블럭하고 이벤트 핸들러에 끼운다.
3. 곱셈 (*) 블럭을 타입블럭하고 varVy 블럭에 끼운다.
4. varLoss 값 블럭을 타입블럭하고 곱셈 블럭의 두 번째 소켓에 끼운다.
5. 덧셈 블럭을 타입블럭하고 곱셈 블럭의 첫 번째 소켓에 끼운다.
6. varVy 값 블럭을 타입블럭하고 덧셈 블럭의 첫 번째 소켓에 끼운다.
7. 두 번째 곱셈 블럭을 타입블럭하고 덧셈 블럭의 두 번째 소켓에 끼운다.
8. varAy 값 블럭을 타입블럭하고 두 번째 안쪽 곱셈 블럭의 첫 번째 소켓에 끼운다.
9. varTi 값 블럭을 타입블럭하고 두 번째 안쪽 곱셈 블럭의 두 번째 소켓에 끼운다.

Clock1.Timer의 varVy 블럭이 완성된 모습은 그림 BC-6과 같다.

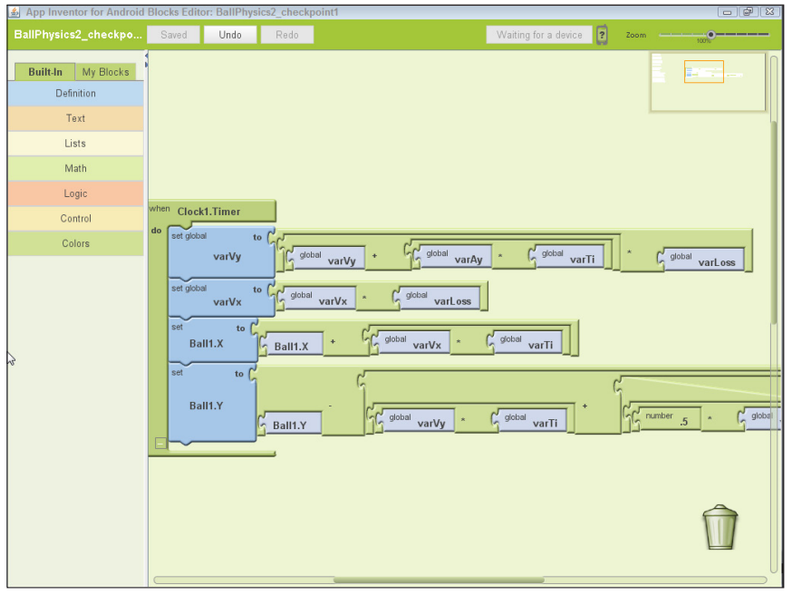

그림 BC-6 varVy 수식 블럭

다음으로 varVx 변수를 설정한다. X 방향 속도 수식은 더 간단하다.

```
Vx * varLoss
```

1. varVx [to] 블럭을 타입블럭하고 varVy 블럭 아래에 끼운다.
2. 곱셈 (*) 블럭을 타입블럭하고 varVx 블럭에 끼운다.
3. varVx 값 블럭을 타입블럭하고 곱셈 블럭의 첫 번째 소켓에 끼운다.
4. varLoss 값 블럭을 타입블럭하고 곱셈 블럭의 두 번째 소켓에 끼운다.

varVx 블럭은 그림 BC-7과 같다.

클럭이 바뀔때마다 스프라이트의 좌표가 변수의 값에 따라 바뀌게 될 것이다.

X 좌표 수식은 varVx 변수값을 이용하면 되며 매우 간단하다.

```
Ball1.X = Ball1.X + (Vx * Ti)
```

위의 수식을 Clock1.Timer 이벤트에 만들어 넣는다. 이는 Ball1 스프라이트의 X 좌표값을 계산하는 것이다. 이 물리 엔진을 애플리케이션 개발에 적용할 때는 Ball1 스프라이트를 실제 움직이고자 하는 스프라이트로 바꾸어 넣으면 된다.

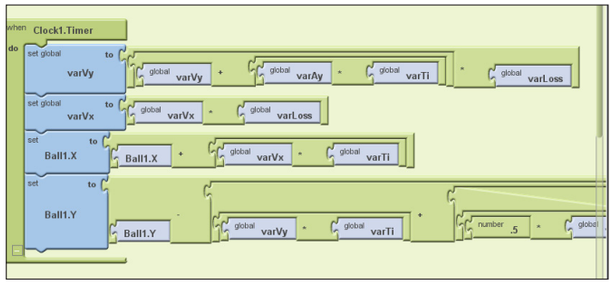

그림 BC-7 varVx 변수 설정용 블럭의 완성된 모습

1. Ball1.X [to] 블럭을 타입블럭하고 Clock1.Timer 이벤트내 varVx 블럭 아래에 끼운다.
2. 덧셈 (+) 블럭을 타입블럭하고 Ball1.X 블럭에 끼운다.
3. Ball1.X 값 블럭을 타입블럭하고 덧셈 블럭의 첫 번째 소켓에 끼운다.
4. 곱셈 (*) 블럭을 타입블럭하고 덧셈 블럭의 두 번째 소켓에 끼운다.
5. varVx 값 블럭을 타입블럭하고 덧셈 블럭 안에 넣은 곱셈 블럭의 첫 번째 소켓에 끼운다.
6. varTi 값 블럭을 타입블럭하고 곱셈 블럭의 두 번째 소켓에 끼운다.

Ball1.X [to] 블럭의 완성된 모습은 그림 BC-8과 같다.

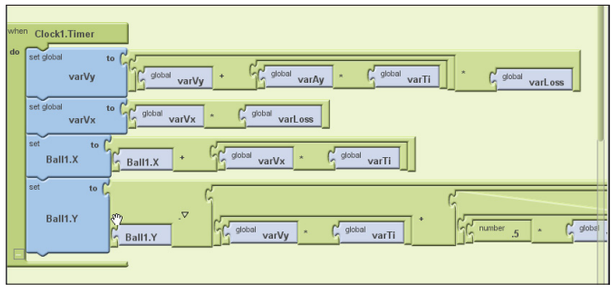

그림 BC-8 X 좌표 처리 블럭 완성된 모습

Y 좌표 계산식은 수식 블럭이 중첩되어 좀 더 복잡하다. 도중에 혼동이 오면 그림 BC-9를 참조한다.

1. Ball1.Y [to] 블럭을 타입블럭하고 Ball1.X 블럭 아래에 끼운다.
2. 뺄셈 (-) 블럭을 타입블럭하고 Ball1.Y 블럭에 끼운다.

3. Ball1.Y 값 블럭을 타입블럭하고 뺄셈 블럭의 첫 번째 소켓에 끼운다.
4. 덧셈 (+) 블럭을 타입블럭하고 뺄셈 블럭의 두 번째 소켓에 끼운다.

다음의 세 단계는 방금 만든 덧셈 블럭의 첫 번째 소켓에 들어가는 내용이다.

1. 곱셈 (*) 블럭을 타입블럭하고 끼운다.
2. varAy 변수 값 블럭을 타입블럭하고 곱셈 블럭의 첫번째 소켓에 끼운다.
3. varTi 변수값 블럭을 타입블럭하고 곱셈 블럭의 두번째 소켓에 끼운다.

다음 단계는 처음에 만든 뺄셈 블럭에 끼운 두번째에 들어가는 내용이다.

1. 곱셈 (*) 블럭을 타입블럭하고 소켓에 끼운다.
2. 곱셈 (*) 블럭을 하나 더 타입블럭하고 첫 번째 곱셈 블럭의 첫 번째 소켓에 끼운다.
3. 숫자 0.5 블럭을 타입블럭하고 안쪽 곱셈 블럭의 첫 번째 소켓에 끼운다.
4. varAy 값 블럭을 타입블럭하고 안쪽 곱셈 블럭의 두 번째 소켓에 끼운다.
5. expt 블럭을 타입블럭하고 바깥쪽 곱셈 블럭의 두 번째 소켓에 끼운다.

   expt 블럭은 base 소켓에 든 값을 exponent 소켓에 든 수만큼 제곱하여 돌려준다.
6. varTi 값 블럭을 타입블럭하고 expt 블럭의 base 소켓에 끼운다.
7. 숫자 2 블럭을 타입블럭하고 exponent 소켓에 끼운다.

완성된 전체 Ball1.Y 블럭은 그림 BC-9와 같다.

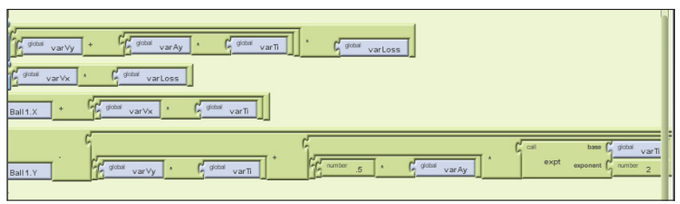

그림 BC-9 Y 좌표 계산 블럭의 전체 모습

기억해두기

Ball1 블럭은 나중에 애플리케이션에서 실제 시뮬레이션하고자 하는 스프라이트로 바꾸어 준다.

이 엔진을 활용하는 테스트 애플리케이션을 만든다면 스프라이트를 하나 정의하고 (위의 예에서는 Ball1) negate 블럭을 원하는 좌표 변수에 적용해서 실제 공이 반동하는 효과를 낼 수 있을 것이다. 스프라이트가 부딪친 후 반동하는 물체가 바닥에 있다면 varVy 변수 값에 negate 블럭을 쓰면 되고, 물체가 양옆에 있다면 varVx 변수에 적용하면 된다. 예제 애플리케이션에서 이를 적용하는 방법을 볼수 있다. .CollidedWith 이벤트로 varVy [to] 혹은 varVx [to] 블럭을 가져와 negate 블럭에 넣고 다시 변수 값 블럭에 넣으면 된다. 그림 BC-10에 예를 보였다. 그림에서는 수평 및 수직 방향에 있는 물체에 부딪치며 반사되어 나오는 것을 볼 수 있다.

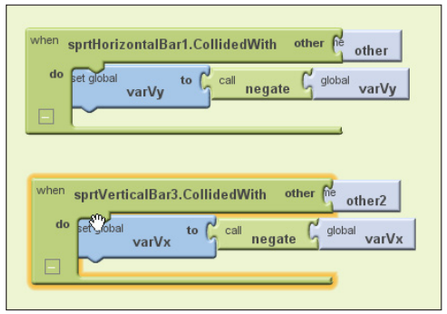

그림 BC-10 물리 엔진이 충돌 이벤트에 적용된 모습

스프라이트가 화면 가장자리에 이르렀을 때도 마찬가지다. 가로 방향 가장자리에서는 varVy 값 부호를 바꾸고 세로 방향 가장자리에서는 varVx 값 부호를 바꾸면 된다. 그림 BC-11에서 Ball1.EdgeReached 이벤트를 처리해서 가장자리 관련 처리를 하는 것을 보였다.

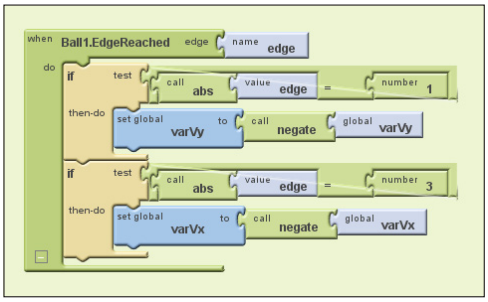

그림 BC-11: 물리 엔진이 가장자리 이벤트에 적용된 모습

물리 엔진은 계산량이 매우 많으므로 오래된 기기상에서는 매우 느리게 돌 수도 있다는 점을 염두에 둔다. 최근에 출시된 1GHz 이상의 기기에서는 별다는 문제가 없다. 어떤 기기에서 애플리케이션이 무리없이 도는지 알아내는 제일 좋은 방법은 패키지로 만들어서 실제 돌려 보는 것이다.

도전 과제

여기서 보인 물리 엔진은 다양한 방법으로 확장과 변경이 가능하다. 가장 간단한 방법은 사용한 상수와 변수 값을 조절해서 스프라이트의 움직임이 어떻게 달라지는지 보는 것이다. 에뮬레이터나 연결된 기기에서 돌려보면 이러한 변경 내용의 효과를 매우 쉽게 확인할 수 있다.

다음의 과제에도 한번 도전해 보자:

- 만든 물리 엔진을 앞의 BreakDroid 애플리케이션에 적용해보기

일러두기

패들의 collide 이벤트에서 공의 가속도를 증가시킨다.

- 달이나 목성의 중력을 시뮬레이션해보기
- 바람의 영향을 시뮬레이션해보기