# 세포 자동화 도입(An Intro to Cellular Automation)

**작성자 : 존 해리스**John Harris [Game Design]
**가마수트라 등록일 : 2011년 5월4일**

[이번 편에서는 게임 제작에 유용한 알고리즘 시뮬레이션인 세포 자동화에 대해 알아보고, 마인크래프트나 드워프요세, 그 밖의 다른 게임들에서 어떻게 사용이 되었는지 설명하며, 그것이 개임 개발의 맥락에서 어떻게 유용하게 사용될 수 있는지 설명 하고자 한다.]

세포 자동화란 무엇인가?

내 이름은 존 해리스(John Harris)이고, 세포 자동화를 사용하여 인 프로펀디스(*In Profundis)* (킥스타터[Kickstarter] 프로젝트 )라는 대형 동굴의 세계 시뮬레이션 게임을 제작하고 있다. 세포 자동화는 물을 흐르게 하고 가스를 퍼지게 하며 바위를 떨어뜨리고 다른 시스템이 자라거나 시간이 지나면서 변하게 합니다. 이러한 기능을 가능하게 하기 위해서는 그리드 기반 게임세계의 각 셀마다 결정된 복잡한 시스템을 실행 해야만 한다.

세포 자동화는 게임 디자이너가 사용하는 도구들 중 큰 관심을 끌지 못해 쉽게 지나쳐 버릴 도구이기는 하지만 이 도구가 적용되어, 몇 몇의 아주 흥미로운 게임들이 만들어 졌습니다. 게임의 리스트는 다음과 같다; *Boulder Dash, SimCity*, *ADOM, Falling Sand, Dwarf Fortress* and *Minecraft*.

리스트의 게임 중 특히 사랑을 받고 있는 마지막 두 인디 커뮤니티 게임은(*Dwarf Fortress* 와 *Minecraft)* 복잡한 세계의 시뮬레이션이 인기 원인의 한 부분이라 할 수 있다.

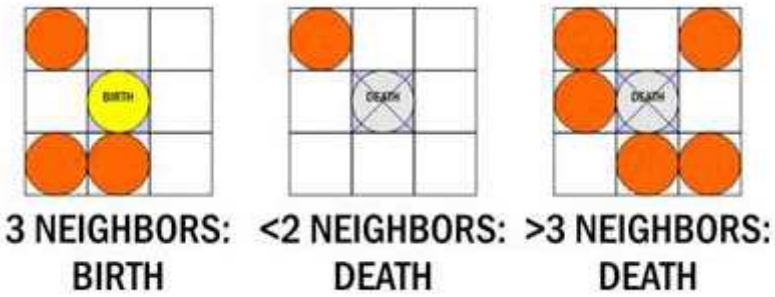
2D의 In Profundis는 조금 덜 복잡하기는 하지만 3D 의Dwarf Fortress와 비슷한 방식을 사용하고 있다. (참고: 가마수트라를 위해 저는 Dwarf Fortress의 프로그래머인 Tarn Adams와 유체 역학을 포함한 게임 구현의 세부 사항에 대하여 인터뷰 했습니다.)

세포 자동화를 크게 정의 하자면 정의된 그리드 안의 cell에 몇 가지 작업을 반복적으로 수행하는 시뮬레이션 테크닉이며, 각 결과는 각 패스의 이전 결과를 교체 하게 된다.

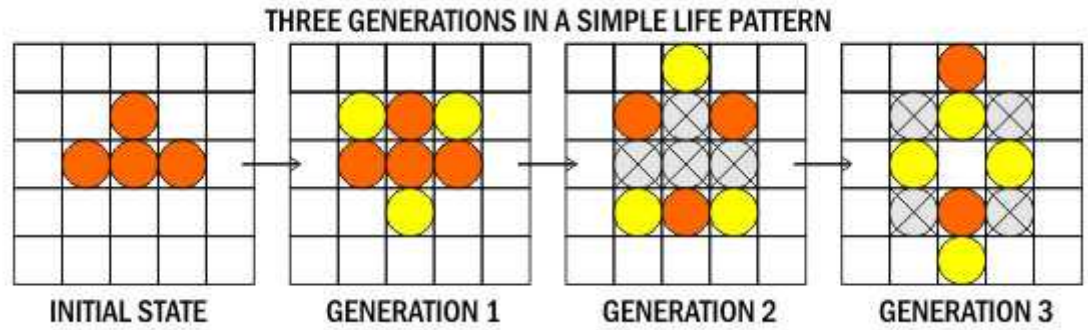
이것은 여러 가지의 이유로 그 동안 간과 되어 왔으나(조금 후에 단점들에 대해 다루겠지만) 제법 오래 전부터 존재 해 왔었다. 세포 자동화의 시작은 수학자 John Horton Conway

의 Game of Life 에서 유래 되었다고 보여 집니다. 여기서 생명은 보드의 카운터 숫자와 몇 가지의 규칙으로 셀의 생사를 결정하게 된다.

플레이어는 시작 구성이 된 보드를 가지고 순서대로 턴을 실행한다. 각 턴마다 세 이웃을 가지고 있는 빈 공간은 출생이 되고 다음 턴에 새로운 카운터를 받게 된다. 카운터가 둘 이하 이거나 셋 이상인 공간은 사망하고 다음 턴에서 카운터가 사라지고 제거된다. 이웃이 되는 공간은 인접한 공간과 대각선으로 인접한 공간을 합하여 모두 여덟 개 이다. 이 여덟 개가 라이프(Life)의 전부이다.




자, 이것이 Life의 전부다 라고 할 수 없지만, 세포적 자동화는 의외의 복잡성을 표현 하기 위한 훌륭한 수단이다. 간단한 패턴과 몇 개의 카운터는 많은 계산을 통해 폭발적으로 늘어나고 수 천개의 세포을 채워 엄청난 양의 표를 만들어 낼 수 있습니다.




Life 의 규칙은 간단하지만 그 결과는 오늘날 까지도 완전히 밝혀지지 않았다.(Life에 대한 자세한 내용검색은 위키 페이지에서 시작하시는 것이 좋겠다. 또한 처음으로 대중에게 공개된 Scientific American 지의 Martin Gardner의 수학 게임 컬럼도 참고할 수 있다. 자신이 한번 직접 해보고 싶다면 오픈 소스에 여러 플렛폼을 지원하는 CA 시뮬레이터인 Golly 를 강력히 추천 한다.)

Life는 컴퓨터 없이도 플레이를 할 수 있지만 최근에 발견된 Life의 행동은 모두 컴퓨터의 Life 시뮬레이터에 의존하여 얻어진 것이다. 최초로 작성된 엔터테인먼트 소프트웨어 프로그램들 중 하나는 라이프 심(Life Sims)이다.

모든 세포 자동화가 Conway의 Life처럼 심오한 것은 아니다. 사실 풍부한 행동성 대비 계산시간을 위해 많은 노력을 하겠지만 실제로 Life 급의 우아함이나 그 정도 수준이 안 된다고 해서 세포 자동화를 잘 사용하지 못한다는 것은 아니다.

세포 자동화의 어떤 점이 게임 개발자에게 유용할까?

¡Ü그것은 동적으로 변화 하는 시스템을 만드는데 큰 도움을 줄 수 있다. 흐르는 액채는 아주 훌륭한 예 라고 할 수 있다. 한 세계에 생물체를 퍼뜨리는 것도 한 예가 될 수 있겠습니다. 연료가 깔려 있는 필드를 따라 불이 붙어 번져 나가는 것은 CA 기반의 시뮬레이션에 아주 잘 맞습니다. *Dwarf Fortress* 와 *Minecraft* 모두는 흥미롭게 (그리고 위험하게) 불이 번져 나가는 행동을 구현합니다.

> ¡Ü잘 설계되어있는 세포 자동화는 드러내지 않게 작업을 할 수 있다. World Sim에, platformer 엔진 같은 다른 시스템을 여러분은 매우 쉽게 덮어 씌울 수 있다. 약간의 추가 작업으로 CA 프로세스를 자체의 프로세서가 구동하고 있는 스레드에 맡겨 버릴 수도 있다. 타일 기반 게임은 이미 셀룰라 자동화가 사용하는 일반, 사각형 그리드에 자연스럽게 호스팅이 된다. 특히, 하드웨어 타일 구조를 사용하는 콘솔은 세포 자동화를 표시하기에 매우 적합합니다.

세포 자동화 시스템은 모든 경우에 유용한 것은 아니며, 일부의 경우에는 어느 정도 용도에 따라 작업을 해 두어야 한다. 게임 개발자가 세포 자동화를 사용할 때 반드시 주의 해야 할 사항은 무엇일까?

> ¡Ü세포 자동화는 프로세서 시간과 메모리 모두를 요구하고 있다. 시뮬레이션의 모든 턴마다 각 그리드별로 계산한 값을 모두 메모리에 보관 해야만 한다. 3D 시뮬레이션에서는 이것이 합쳐져서 큰 저장소를 요구하고 있다. *Minecraft* 는 그 격자의 "입자"를 각 측면당 미터로 매우 크게 제작함으로써 문제를 해결했다. 물 한 블럭을 긴 도랑을 따라 여러 갈래로 복잡하게 갈라지도록 흘려 보낼 때에 물의 흐름이 계산되는 동안 버벅거림이 일어남을 볼 수 있을 것이다. (현재~1.2). *Dwarf Fortress* 의 경우 세션의 시작과 종료 시에 긴 로딩과 저장 시간을 요구한다. 그리고 한 번에 많은 양의 액체가 쏟아졌을 경우에는 속도가 크게 느려진다. 설계와 최적화를 신중하게 수행한다면 프로세스의 양은 크게 줄일 수 있을 것이다. 그러나 CA 기반 시뮬레이션은 세상을 역동성 있게 바꿔나갈 수 있다는 점이 흥미로운 사실이기는 하지만 매번 순서마다 변화가 늘어날수록 처리 시간이 늘어나는 것은 불가피 하게 보인다.
>
> ¡ÜCA는 큰 사이즈 격자의 변화를 시뮬레이션 하기에 적합하다. 그러나 그리드 표면의 특성과 (배열이 효과적으로 계산할 수있을만큼 작을 경우), 각 공간에서 단순함이 요구되는 작업으로 인해 진정한 뉴턴의 물리 시뮬레이션을 쉽게 구현 하기가 힘들다. 예를 들면, 낙하물의 속도는 격자 크기에 의해 제한된다. 그것은 어느 정도 속임수를 쓸 수도 있겠지만 반드시 작업이 동반되어야 한다. 그러나 실제로 액체를 부을 때의 광경은 물리학의 충실도가 일반적으로 필요한 것은 아니다
>
> ¡Ü시뮬레이션이 균등하게 수행되도록 어느 정도의 작업이 필요하다. 다시 말하자면, 초기 시작 부분에 프레임에 도입된 효과는 다음 계산이 끝난 프레임에서는 영향을 미치지 않는다는 것이다. 가장 쉽게 처리 절차를 반복 할 수 있는 방법은 셀룰라(세포적인) 세계를 루프 안에 정착시켜 매 턴마다 셀 각각의 인덱스 변수 계산이 순차적으로 계산 반복되어 배열을 정하게 하는 것이다. 그러나 y 루프를 적용 되어 있는

공간 안에서 바위가 떨어지는 과정을 보면 바위의 위치는 매번의 x 루프가 완료 될 때 마다 y값이 하나씩 낮아지게 된다. 특단의 조치를 취해 놓지 않는다면 다음 번의 y루프에서는 바뀌어진 바위의 위치에서 한 칸 더 떨어지게 되어 결국에는 바위가 순간 이동하여 바닥으로 떨어진 것처럼 보이게 된다. 같은 맥락으로 x 루프를 사용할 때에도 같은 문제가 생기게 된다.

이와 비슷하게 Life 에서도 셀의 탄생이나 죽음을 업데이트 하는 과정에서 변화된 값이 세계의 그리드에 바로 적용이 될 경우 그 프레임에서 생성되는 탄생이나 죽음이 영향을 받게 된다. 이것을 해결할 수 있는 방법들 중 하나는 변화를 기록 하기 위한 월드 그리드를 따로 분리해 두었다가 프레임이 끝날 때 포인트를 교체 해 주는 것이다. 다만 이 방법은 변화를 만들 수 있는 다른 스레드가 더 존재할 경우에는 수정해야 하는 프레임을 찾아 내는 것이 쉽지 않다. 또 다른 해결 방법은 특수한 자리 표시자 값을 사용하여 출생과 죽음의 진행을 지정하는 과정을 따로 하나 추가 하는 것이다. 문제를 해결 할 수 있는 또 하나의 방법이 있다. 저는 이것을 '부분 업데이트 " 문제 라고 부른다. *In Profundis 은* 자체적으로 다소 핵키한 솔루션을 사용하고 있다. 아래쪽으로 더 내려가면 자세한 설명을 하고 있다.

¡ÜCA 시스템은 백지에서 시작하여 디자인 하기에는 어려움이 있을 수 있다. 유체 시뮬레이션 같은 실제 물리학을 구현하려는 경우에는 그리 어렵지 않다. *SimCity* 처럼 좀 더 복잡한 많은 목적을 위해 그것을 사용하려 한다면 심리적으로 부담이 좀 될 것이다. 커다란 규모의 CA 규칙을 시각화 하는 것이 어렵다고 하는 의미는 유용하거나 재미있는 동작을 찾아내야 하기 때문이다. 그러기 위해서는 프로토 타입을 만들어 디자인 공간을 탐험해 보아야 한다. 이것은 엄격히 정해진 제품 생산의 일정에 걸림돌이 될 수 있다. 자신의 개인 시간에 미리 CA 규칙들을 시도 해보고, 검증된 룰셋(ruleset) 을 가지고 프로젝트에 임하는 것이 좋을 것 같다.

¡Ü때로는 CA를 예측할 수 없다! 그것은 매력의 일부이지만, 한편으로는 디자이너와 동등한 창의력을 발휘 할 때도 있다. 그래서 많은 CA 게임들이 절차적 또는 사용자 생성 콘텐츠를 사용하는 이유일 것이다.

이것은 단점이 많은 것으로 보일지도 모르지만, 심오한 장점들이 있으며, 그것이 별도로 구현하기 어려운 수준의 뉘앙스를 주는 세계 시뮬레이션을 가능하게 한다.

### Profundis의 유체 시스템 고찰 (A Look at *In Profundis'* Fluid System)

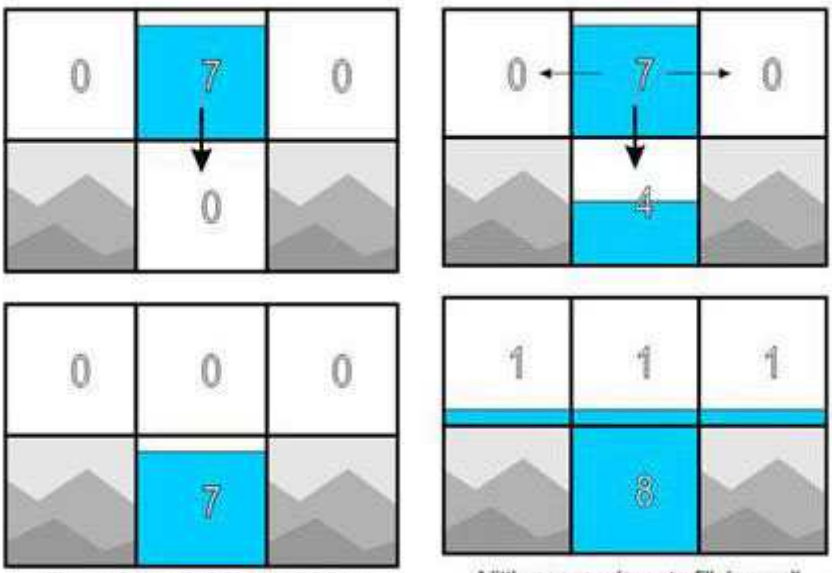
세포 자동화가 프로세서 시간을 최소화 하는 쪽으로 촛점을 맞추어 설계 되어야 하지만, 그렇다고 꼭 이것이 느리다는 것은 아니다. *In Profundis'에*서는 상당히 복잡한 구조에도 불구하고 상당한 양의 액체를 흐르게 하고 가스를 퍼지게 하면서도 프레임 속도를 45fps을 상회하면서 유지 할 수 있다. 이것은, 게임 엔진으로서 Python을 사용하여 작성되고, Psyco로 가속을 해 준 상태이다.

또한, *In Profundis'* 에서는 부분 프레임들이 시뮬레이션을 보존하면서 실행을 할 수 있도록 디자인되어 있다. 각 라인은 별도로 계산될 수 있으며, 각 게임 프레임들 마다 CA 필드의 필요한 일부만이 계산이 된다.

이제 이 시스템에서 어떻게 물이 다루어 지는지 설명하겠다. 전체 시스템은 여러 유체를 서로 다른 확산 규칙과 밀도로 시뮬레이션 할 수 있지만, 그 중 물은 시뮬레이션 시작하기에 훌륭한 액체이며, 그 자체로 재미있는 게임 역학을 제공할 수 있다.

유체를 포함하는 각 공간은 공간을 나타내는 높이의 변수가 0에서 8까지 있다. 자, 그러면 *In Profundis에서 업*데이트 루프가 물을 만나면 어떻게 되는지 알아 보기로 하겠다. (사실, 정확하게 따지자면 이 상황은 일어나지 않는다. – 그것은 이전 버전의 게임 코드에 대한 설명이다. 현재 버전은 쉽게 설명하기에는 너무 복잡하다.)

¡첫째, 현재 셀 아래의 공간은 비어 있는가 아니면 일부 채워져 있는가? 비어있는 경우, 현재 셀과 아래 셀의 내용물을 서로 교환함으로 "낙하"를 만들어 낸다.
완료하고, 루프를 계속한다. 일부 채워져 있다면, 어느 정도 공간이 남아 있는가? 공간이 액체 보다 많거나 같다면 현재 공간의 액채를 아래 공간으로 더한다. 완료.
공간이 액체보다 적을 경우, 아래 쪽에 남은 공간 만큼 물을 빼서 아래 쪽 공간에 더하고 나머지 물은 옆 칸으로 흘러 들어간다.

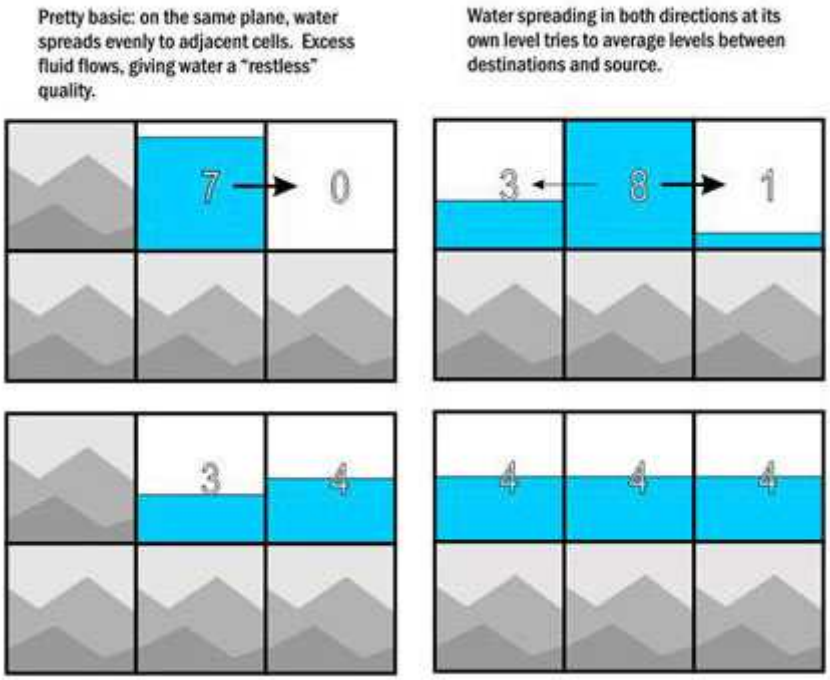


Simplest case: unsupported water simply falls down.

A little more complex: water fills lower cell, then spreads evenly at its own layer.

낮은 쪽의 세포가 벽이나 지나갈 수 없는 기능이 있거나 물로 가득 차 있다면 오른쪽이나 왼쪽에 액체가 들어갈 공간이 있는가? 양쪽 모두 액체가 이동할 수 있는 공간이 없다면 액체가 더 이상 움직일 수 있는 공간이 없다. 완료.

¡Ü만약 단 한쪽만이 비어 있거나, 반대편은 현재보다 더 많은 물이 있다면, 공간이 더 많이 비어있는 쪽으로 움직일 수 있는 만큼의 액체를 이동 시킨다. 이렇게 할 때에 방향 깃발을 셀에 설정해두어 나중에 셀이 액체의 흐름을 결정할 때에 기울어진 방향으로 인식하게 한다. 이것을 흐름 방향의 깃발 이라고 할 수 있겠다. 이것은 어떤 양의 액체가 양쪽 방향 중 어느 쪽으로 흘러가야 할지를 결정할 때에 중요하다. 방향에 대한 검사에서 액체는 깃발이 가리키는 기울어진 방향으로 움직이게 된다.




이것은 물이 빠르게 확산 되는 것을 도와 주고 있다. 하지만 이것은 물이 웅덩이를 만드는 대신 물 한 칸 단위의 조각들이 이리 저리 움직이게 된다. (실제로 현재 빌드에는 가끔 이것을 방지하는 버그가 있다.) 어떤 경우이든지 우리는 현재 셀이 목적지 셀보다 얼마나 많은 물을 가지고 있는지 알아보고 양 차이의 절반에 반올림 한 만큼의 양을 이동 시킨다.

¡Ü만약 양측 물의 양이 현재의 셀보나 낮다면, 우리는 수위가 모두 동일해 질 수 있도록 흐름을 조절하려고 한다. 물은 흐름 방향 쪽 면으로 이동하거나 임의로 방향을 선택한다. 우리는 또한 각 방향으로 깃발을 설정한다.

이러한 모든 경우에, 물이 계산한 방향으로 흘러갈 때에 물이 순간 이동을 하는 것을 막기 위해 Nocalc 이라는 깃발을 세운다. (위에서 언급한 "부분 업데이트" 문제와도 관련되어 있는 이것에 대한 자세한 내용은, 아래를 참조하시오.)

중요한 점은 제가 물이 들어있는 셀과 관련된 모든 상황을 커버했고 각 상황 별로 어떤 일이 일어나야 하는지 분류 하였다는 것이다.

나는 물이 본체로서는 어떻게 해야 하는지 걱정하지 않는다. 그냥 작은 한 부분을 시뮬레이션하고 반복하여 스크린 전체를 루프 안에 두는 것에 중점으로 두었다

그것은 복잡한 알고리즘이 아니라, 수백 번의 루프를 반복하여야 하기 때문에 좋은 CA 코드는 장황하지 않는다. 간단한 코드를 작성하고, 모든 셀들이 서로 협동하여 의외의 행동을 만들어 내고 있다.

이 코드는 규모로 볼 때에는 Tarn Adams의 Dwarf fortress의 물이 흐르는 시스템의 2D 버

전입니다, 그리고 공통적인 결함이 하나 있다. 그것은 수압이 잘 처리되지 않은 점 이다. 사실 제 게임은 수압을 전혀 처리하지 않았다. 지금까지 제 게임에 실제로, 수압을 전혀 처리하지 않았다. 아담스 코드는 시간을 절약할 해킹의 비트(Bit)를 사용하여 시뮬레이션 한다.

그 결과, 제 게임에는 U 자로 꺽인 터널의 한 쪽에 물을 부어 넣어도 다른 한 쪽으로 물이 차오르지 않는다.

여기를 클릭하여 *In Profundis'* 의 물 시뮬레이션을 볼 수 있다. 엔진이 사용하는 최적화들이다:

> ¡Ü유사한 "*Falling Sand*" 시뮬레이션과 달리, *In Profundis'* 는 시스템의 픽셀이 아닌 타일을 계산한다. 계산으로만 따지자면 "*Falling Sand*" 보다 훨씬 복잡하다. (여러 유형의 액체가 한 셀에 존재할 수 있기 때문이다.), 각 타일은 24개의 픽셀로 이루어져 있으며 "*Falling Sand*" 에 비해 훨씬 탁월한 인지 속도를 낼 수 있다.
> ¡Ü*In Profundis 는* 화면에 보여지는 부분과 그 테두리 주변의 작은 지역만을 시뮬레이션 한다. 테스트 베드에서도 테두리 주변 지역은 가끔 보일 뿐이다. 시뮬레이션이 시작되기 전에 몇 가지 예비 주기를 전 세계에 실행하여, 액체가 분지를 찾아내어 채우고 시뮬레이션이 균형을 찾아 더욱 완전한 환상을 만들어 낼 수 있다. 완성된 게임은 시야에 제한이 더해지면서 게임 세계의 제한된 업데이트 본성을 감추는데 도움이 될 것이다.

위에서 언급한 "부분 업데이트" 문제에 대한 해결책은 좀 특이하다. 나는 별도로 그리드 복사본을 보관하지 않았다. *Falling Sand* 를 개선하려고 노력 중인 애호가들에게 영감을 받아 모든 개선을 그리드에 바로 기입하여 메모리 부하와 프로세서 부담을 줄였다. *In Profundis'* 그리드의 조금 특별한 점들은 문제를 피하는데 도움이 되었다.

> ¡Üy루프는 아래에서 위로 반복된다. 이 방법으로 떨어지는 개체는 루프마다 한 번만 보여지게 되어 각 프레임마다 한꺼번에 여러 번 떨어지는 것을 막을 수 있었다.
> ¡Ü각 세포에 "계산 금지" 플래그가 있다. 현재 월드 프레임에서 변경된 것을 제외한 다른 셀들(개체가 이동하여 들어온 셀)은 계산 금지 깃발을 받게 된다. 이후 반복의 차례가 돌아왔을 때 깃발이 있으면 루프는 건너뛰게 된다. 이것은 처음에는 조잡한 인터페이스로 보여질 수 있지만 실제로는 그렇지 않다. 생각해 보면 두 위치를 교환할 때에 사실은 움직이는 개체와 빈 공간이 두 위치를 한번에 계산하는 것이기 때문이다. nocalc 플래그는 새로 지금 채워진 공간이 추가적으로 변형 되는 것을 효과적으로 방지하는 방법이다. 조잡하다고 볼 수도 있겠지만 실전에서는 유용하다.

위와 같은 고려사항에도 불구하고 가끔은 순차적인 루프의 특성상 컴퓨터의 공예품이 되어 버릴 때가 있다. 이들은 각 라인을 계산할 때 좌 우를 바꾼다거나 프레임을 모두 바꿈으로써, 어느 정도 다듬을 수 있다. 이 방법은 떨어지는 모래 공동체(Falling Sand community)가 떨어지는 모래(Falling Sand)라는 게임을 여러 확장된 버전으로 만들어낸 웹 작품에서 영감을 받았다. 세포 자동화를 이용하여 게임을 만들고자 하는 사람에게는 아주 흥미로운 곳이다.