※ 본 기사는 CMP MEDIA LLC와의 라이선스 계약에 의해 국문으로 제공됩니다

성공을 위한 실패 : 게임플레이를 간단한 플레이어 메트릭스로 전환하기

(*Hot Failure: Tuning Gameplay With Simple Player Metrics*)

크리스 프루엣(Chris Pruett)

가마수트라 등록일(2010. 12. 16)

http://www.gamasutra.com/view/feature/6155/hot_failure_tuning_gameplay_with_.php

[이 글은 2010 년 9 월 Game Developer magazine 에서 발췌한 것으로, 구글 게임 개발 지지자 Chris Pruett 이 안드로이드 게임, Replica Island 에 어떻게 빠르고 값싼 메트릭스를 구현했는가를 설명하고 있다.]

다른 사람이 당신의 게임을 지켜 보는 것 만큼 좋은 것은 없다. 개발 과정 중에, 당신은 매일 게임을 하고, 아마도 무의식적으로 특정한 플레이 스타일을 개발했을 것이다. 그러나 게임에 대한 경험이 많지 않은 사람들에게 당신의 게임을 플레이 하도록 하면, 당신의 디자인이 어떤 방식을 이해되는지를 알아볼 기회를 가지게 된다.

충돌이 발생하고, 애니메이션이 끊어지고, 튜노리얼 메시지가 혼란스러워지며 간헐적으로 발생하는 버그는 초보자가 게임을 플레이 할 때 더 증폭적으로 나타난다. 얼마나 많은 버그를 고치고, 얼마나 많이 게임을 수정하더라도, 당신의 플레이 스타일과 친근한 콘텐츠 유사성은 다른 이용자가 즉각적으로 마주치게 될 문제의 편견을 피할 수 없게 한다.

때문에 플레이 테스팅은 좋은 게임을 만들기 위한 필수적인 부분이다. 플레이 테스팅의 궁극의 결과를 가지기 위해서, 당신은 게임 플레이 메트릭스를 모으는 나의 경험을 통해서 데이터를 모아 보려고 할 것이다.

단순하게 시작하기

Game Boy Advance games 로 이야기를 시작하고자 한다. 그때, 플레이 테스팅에 대한 우리의 생각은 매우 직선적이었다 : 우리는 지역의 어린이를 초청하여 그들에게 GBA 를 준 다음, 얼마

동안 플레이를 하게 하고 영상을 촬영했다. 그리고 나서 촬영 영상을 리뷰하였다. 이 절차는 즉각적이고 극적인 버그를 찾을 수 있게 해 주었다.

개발 팀이 당연하다고 여겼던 영역이 테스터에게는 무한한 좌절을 안겨주는 원천이 되는 경우가 많았다. 타겟 오디언스가 특정 영역에서 지속적으로 실패하게 될 경우, 이것은 뭔가 고쳐져야 한다는 명확한 메시지를 전달한다. 이 생생한 테스트와 만들고 있는 게임에 대한 반복적인 플레이를 통해서 게임은 향상될 수 있다.

요즘, 나는 안드로이드 휴대폰 게임을 개발하고 있다. 나의 첫 번째 안드로이드 게임인 *Replica Island* 는 스크롤 게임인데, 10 년전 내가 만들었던 GBA 게임과 크게 다르지 않다. 그러나 일부 변한 것도 있다 : 나는 더 이상 게임 스튜디오에서 일하지 않는다 ; 나는 *Replica Island* 를 혼자 만들었는데, 한 명의 아티스트의 도움을 받았을 뿐, 대부분 여유 시간에 작업을 진행했다.

나는 또한 더 이상 어린 플레이 테스터를 동원하지 않는다. 만약 내가 예전에 그렇게 했다 하더라도, 이제 나의 타겟 오디언스는 조금 더 나이 들어 있는 층이다. 그리고 누군가가 플레이하고 있는 동안 핸드폰의 결과물을 기록한다는 것은 쉬운 일이 아니다. 핸드폰을 게임을 이용하고 있는 사람을 어깨 너머로 관찰하는 것은 너무 이상하기도 하고, 테스트 플레이에 영향을 미칠 수도 있기 때문에 테스트 자체가 어렵다.

인디 핸드폰 게임 개발자들은 어떻게 하고 있을까? *Replica Island* 의 완전성에 대하여, 나는 이 게임이 재미있다는 것을 증명할 어떤 방법도 없다는 것을 깨달았다. 게임은 거의 다른

사람의 도움 없이 만들어졌고, 내가 이 게임에 대한 자신감을 가지기 위해서는 출시하기 전에 더 많은 사람들의 견해가 필요했다.

내가 노력한 첫 번째 일은 이용자 조사였다. 나는 이메일을 보내 게임을 하고, 결과를 알려 달라고 하였다. 심지어 게임에 대한 몇 가지 질문을 형성하여 피드백 포럼을 구성하였다.

이 접근법은 꽤 실패적이었다. 많은 사람들이 게임을 다운로드 하였지만, 1% 미만의 사람들만 나의 5 문제 설문을 채워주었다. 그러나 이것 조차도 충분한 정보를 제공해 주지 않았다. "게임이 너무 어렵다"는 것이 플레이어 콘트롤의 문제인지, 레벨 디자인의 문제인지, 퍼즐 디자인의 문제인지, 아니면 튜토리얼 레벨의 문제인지를 판별해 내는 것은 너무 어려웠다.

메트릭스에 대하여 생각하기

이러한 실패 후에, 나는 Naughty Dog 이 개발한 오리지널 *Crash Bandicoot* 의 플레이어 메트릭스 시스템에 대한 기억이 났다. 이 시스템은 메모리 카드에 플레이에 대한 통계를 작성할 수 있게 해 주며, 플레이어의 실패 수와 플레이 기간 등을 기록하여 플레이어의 게임 경험이 극한에 이르는 지점을 찾을 수 있게 해 주었다.

이 문제의 지점은 수정되었고, 데이터는 다시 게임의 역동적인 난이도 조정을 조율하는데 활용되었다. Naughty Dog 은 게임 종료 화면은 어떤 비용을 들이더라도 피해야만 한다는 생각을 가지고 이 시스템을 개발하였고, 이 생각은 정말 흥미로운 원칙 중의 하나이다. 이들의 최종 목표는 플레이어가 게임에 갇혀서 계속해서 플레이를 할 수 없는 "선반 상태"가 일어나지 않도록 하는 것이었다.

나는 이것이 꽤 좋은 아이디어라고 생각했다. 그러나 나는 이것을 어떻게 휴대폰에 적용할 수 있을지 확신하지 못했다. 나는 예산을 많이 들인 게임의 메트릭스 리코딩 상태가 어떠한지를 알아보았고, 많은 회사들이 플레이어의 액션에 대한 통계를 기록하고 있다는 것을 알았다. 몇몇 사람들은 나에게 그들이 많은 정보를 수집하고 있으며, 특정한 디자인 변경을 제시할 결과를 데이터로부터 분석하는데 어려움을 겪고 있다고 말해주었다.

반면에, 일부 스튜디오는 레벨에 따라서 플레이어의 경로를 재창조할 수 있는 도구를 가지고 있었고, 이용자가 선호하는 무기에 대한 통계를 가지고 있었다. 어떤 적인 특히 강한지, 지도의 어느 부분이 특히 눈에 들어 오는지 등에 대한 선호체계도 알고 있었다. 플레이어 메트릭스의 콜렉션은 다양한 게임에 적용될 수 있을 듯 보였지만, 그들이 모은 데이터를 처리하는 도구를 만드는데 유의미한 시간을 보낸 스튜디오만을 위한 것이었다.

(이런 종류의 시스템이 극단으로 치닫는 예로써, Georg Zoeller 가 언급했던 BioWare 의 the crazy system 를 찾아 보라.) 데이터를 수집하는 것은 쉬운 부분이지만 디자이너가 활용할 수 있는 방식으로 렌더링 하는 것은 훨씬 어려움이 틀림없다.

낙담시키는 듯이 들리겠지만, 나의 목표는 나의 도구를 가능한한 단순하게 유지시키는 것이었다. 그러나 나는 어쨌든 메트릭스 레코딩을 실행하기로 결심했고, 몇 가지의 핵심 메트릭스를 가지고 시작하였다. 나의 안드로이드 핸드폰은 메모리 카드를 가지고 있지 않았지만, 지속적인 인터넷 접속이 가능하였다. 내 생각에, 내가 중요한 이벤트에 로그할 수 있고, 그것을 서버에 보낸다면, 이런 방식을 플레이어의 결과를 가질 수 있을 것 같았다. 나의 목표는 시스템을 가능한 한 단순하게 유지하면서 가능한 한 플레이어를 이해할 수 있는 방법을 찾는 것이었다.

기본 시스템

내가 작성한 로깅 시스템은 세 부분이다 : 플레이어 이벤트를 모으는 게임 런 타임 쓰레드와 이 정보를 서버에 보내는 부분, 서버 ; 그리고 서버에 기록된 데이터를 분석하는 툴.

"서버"는 두번째 구성 요소에서 중요한 단어이다. 내 서버는 PHP script 여서, 보내진 HTTP Get query 를 인증하고, MySQL 데이터 베이스에 결과를 작성했다. Query 자체는 완전히 단순하다 : 이벤트 이름, 레벨 이름, xy 좌표, 버전 코드, 세션 아이디 와 시간 스탬프. 이러한 필드는 데이터베이스에 글자 그대로 기록된다. 데이터의 실질적인 프로세싱은 스페셜 대시보드 페이지가 로드될 때, 요청에 의해서만 가동되는 PHP 에서 이루어진다.

나는 두 가지의 이벤트로 시작했다 : 플레이어 죽음과 레벨 완료. 플레이어가 죽거나 레벨을 완료할 때 마다, 게임은 서버에 이벤트에 대하여 리포트 한다. 이 데이터로부터, 나는 꽤 세부적인 게임 흐름에 대한 오버뷰를 구성할 수 있었다. 나는 어떤 레벨이 어느 정도의 시간을 요하는 지를 알 수 있었고, 어느 부분에서 플레이어가 가장 많이 어려움을 겪는지, 어디에서 가장 쉽게 해결하는 지 등을 알 수 있었다.

독특한 플레이어 수로 나누어서, 나는 특정 지점에서 플레이어가 죽는 퍼센트를 알 수 있었고, 각 플레이어의 평균 죽음의 수를 알아냈다.

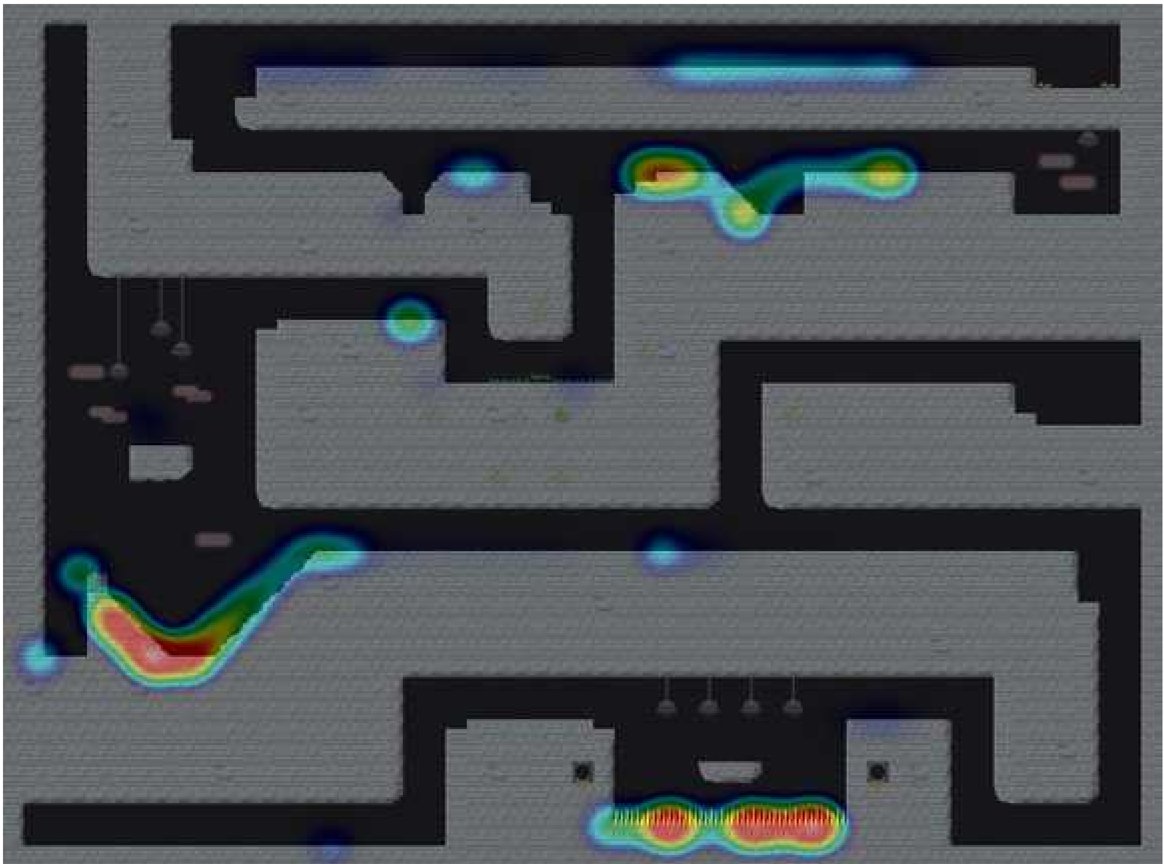
이벤트의 공간 위치를 봄으로써, 적과의 싸움에서 이기지 못해 죽음을 맞이하는지, 뜻밖의 상황에서 죽음을 맞이하는지를 구별할 수 있었다. 이 단순한 메트릭스 시스템은 꽤 세부적인 것을 증명해 주었다.

밝은 붉은색으로 실패를 강조하기

기본적인 시스템을 만들어 운영한 다음, 나는 테스트에게 업데이트를 거친 버전으로 플레이를 하게 하였다. 그리고 습득한 데이터를 보았다. 매우 빠르게 패턴이 나타났다. 최소한 하 지점에서 거의 플레이어가 전부 다 죽는 레벨이 있었으며, 또 다른 지점에서는 몇 시간 동안 빠져 나오지 못하기도 하였다 (5 분 정도 걸리는 레벨은 좋은 디자인은 아니다). 숫자를 보는 것 만으로도, 나는 어떤 레벨이 가장 작업을 필요로 하는 지에 대한 명확한 그림을 그릴 수 있었다.

그러나 문제가 있는 레벨을 구별하는 것만으로는 충분하지 않았다. 때로 나는 특정한 레벨이 왜 문제인지를 구별할 수 없었다.

그래서 나는 한 단계 더 들어갔다. 같은 데이터를 활용하여, 나는 탑 레벨에서 죽음 위치를 알아낼 수 있는 도구를 만들었고, 그 결과 나는 이용자가 어디서 죽어가고 있는지를 정확하게 볼 수 있었다(그리고 그들이 어디서 활발히 살아 움직이는지를 볼 수 있었다). 한명의 플레이어가 죽을 때마다 레벨에 작은 점을 그리게 되는데, 플레이어의 수가 많아지면, 죽음의 위치가 많은 지도를 렌더링하여 레벨 수준을 바꾸었다.



***Replica Island* 플레이어의 죽음 통계로 만들어진 히트 맵**

오브젝트 레슨으로써 게임 디자인 실패

고레벨 플레이 통계와 죽음 위치의 조합은 이해에 도움이 되었다. 예를들어, 나는 수 많은 플레이어가 첫 번째 적을 만났을 때 죽어가고 있다는 것을 알게 되었다. 이는 적이 특별히 강했기 때문이 아니라 적이 낮은 천정 때문에 특정 행동을 구현하기 어려운 장소에서 공격을 해 왔기 때문이었다.

나는 또한 나의 단순한 동적인 난이도 조정 시스템이 그 자체로 조정을 필요로 하고 있다는 것을 알게 되었다. 이 시스템은 플레이어가 특정 수의 연이은 죽음을 맞이한 이후에 힘을 더 얻거나 목숨을 증가시키도록 한 것이었는데, 데이터를 보고서 나는 이것이 훨씬 더 빨리 발생해야 한다는 것을 알 수 있었다.

그리고 나의 레벨 기하학도 바뀌어야 한다는 것을 알게 되었다. 한 단계를 완료하는데 시간은 매우 오래 걸리지만, 죽을 가능성은 매우 적은 몇 단계를 만들었는데, 플레이어는 단순히 길을 잘 잃어버렸다. 나는 이 단계를 좀더 분명한 길을 만들어 주는 것으로 수정하였다. 이러한 한 두 가지 경우로, 나는 전체 레벨을 폐기하고 새로운 것을 만들었다.

그러나 내가 규명한 가장 큰 문제는 구덩이에 있었다. *Replica Island* 는 플랫포머이고, 당신이 추측할 수 있듯이, 구덩이를 넘어 가는 많은 점프를 수분한다. 그러나 돌아다니는 동물과 파이프 배관공과 달리, 나의 캐릭터의 이동에 대한 메인 모드는 날아가는 것이다.

나는 D-pad 를 필요로 하지 않는 콘트롤 시스템이 필요했다. 그래서 *Replica Island* 에서 주인공인 녹색 안드로이드 로봇은 그의 발에 반동 추진 엔진 로켓을 이용하여 날아 다닌다. 기본적인 움직임 모델은 점프하기 전에 땅에서 가속도를 올린 다음 그 가속도를 활용하여 엔진으로 멀리 날아가도록 하는 것이다. 반동 추진 엔진은 연료를 빨리 소모하지만 착륙할 때 다시 채워진다. 그래서 플레이어는 멀리 있는 바위나 정확한 지점에 도달하기 위해 점프를 하고 연료를 소비할 것이다.

이 모든 것이 잘 되어 있고, 좋지만 나는 플레이 테스트로부터 돌아온 죽음의 데이터를 보았을 때 플레이어가 바닥이 없는 구덩이에서 집단으로 죽어가고 있다는 것을 알게 되었다. 플레이어 무리는 심지어는 아작 작은 구멍에도 떨어지고 있었다. 게다가 구덩이에 의한 죽음은 게임 코스를 넘어가면서도 감소하지 않았다 ; 플레이어는 시간이 지나도 점프를 더 잘하지 못하고 있었다.

이 정보를 가지고, 나는 핵심 게임과 레벨 디자인을 재 정비 했다. 그리고 수 많은 이론을 떠올렸다. 기본적인 문제는 플레이어가 그들이 점프하고 있는 구덩이를 볼 수 없다는 것이라고

생각했다. 우선, 죽음의 구덩이가 진짜 죽음의 구정이라는 시각적인 지시물이 없었다 ; 내가 설정한 레벨이 좀 높은 편이기 때문에 어떤 구덩이가 지하의 레벨 부분과 이어지는지, 어떤 것이 소름끼치는 사망에 이르게 하는지를 구분하는 것이 어렵다.

둘째, 가장 중요한 나의 카메라가 플레이어가 점프할 때 바닥을 보여줄 수 있도록 역할을 잘 하고 있지 않았다. 거의 플레이어가 도약하자 마자 땅이 스크린의 바닥으로 스크롤되어서, 어디에 착지 해야 할지 판단하기 어렵게 만들었다.

Super Mario Bros 와 같은 플랫포머는 거의 절대 수직으로 스크롤하지 않는다 ; Mario 는 카메라가 상하로 움직이도록 허락하는 특정한 환경에 대하여 정확하게 지시해 주는 완결된 법칙을 가지고 있다. 그러나 *Replica Island* 에서 비행 메캐닉은 일반적인 경우에도 수직의 스크롤링을 허락하고 있다. 수 많은 변경 후에, 나는 플레이어가 알아볼 수 있는 공간을 떠나는데 근접하지 않는 한 수직으로 스크롤하지 않는 좀더 똑똑한 카메라를 고안했다.

이러한 변화를 만든 후에, 나는 베타 테스터에게 새로운 업데이트 게임을 주었고, 이전 버전의 결과와 비교했다. 그 다음 버전은 매우 안정적이었다 ; 죽음은 전반적으로 줄어들었고, 레벨 완료 시간도 거의 대부분 정상적인 범주로 돌아왔다. 그리고 죽음 구덩이는 크게 줄어들었다. 나는 게임을 출시할 준비가 될 때까지 이 테스트를 여러 번 수행하였다. 그러나 제자리에 메트릭 보고 시스템을 구비하면서, 내가 만든 변화가 얼마나 테스터가 플레이하는데 영향을 주고 있는지를 알아 볼 수 있는데 도움이 되었다.

출시

테스트 그룹을 활용한 몇 번의 반복 끝에, 내가 찾고 있었던 벨 커브에서 일직선으로 변하는 그래프를 맞이하게 되었다. 게임을 출시해야 할 때가 되었고, 나는 메트릭스 시스템을 떠나기로 결정하였다. 테스트 그룹에서 나온 데이터와 실제 이용자로부터 얻게 되는 데이터가 다른지 궁금했다.

물론 서버에 돌아가는 앱 데이터를 받을 때 마다, 이용자가 어떻게 이용하고 있는지를 알려주는 것이 최선이다. *Replica Island* 가 출시되었을 때, 많은 게임 향상에 대하여 환영의 인사가 나타났다. 이 메시지는 또한 게임을 향상시키기 위하여 서버에 업로드 될 익명이 플레이 데이터로 이용자에 대한 정보를 제공할 것이다.

이 접근이 최고의 해결법인 것 같았다 : 코드가 오픈소스이고 어떤 사람도 데이터의 내용물을 패킷 그 자체로 볼 수 있더라도 (그리고 어떤 메트릭스 데이터도 어떤 특정한 이용자나 디바이스에 국한될 수 있다고 확신함에도 불구하고), 이용자가 아웃 풋에 "아니요 괜찮습니다"라고 말할 기회를 준다는데 의미가 있다.

안드로이드 마켓 인스톨에 대하여 독특한 레포팅을 해 준 이용자를 비교해 본 결과, 20% 이하의 이용자가 메트릭스 결과에 대한 아웃 풋을 선택한 것 같았다.

결과적으로, 나는 이제 어마어마한 데이터를 가지게 되었다 – 백 4 십만개의 데이터 포인트를 넘어서 유저 베이스에 의해 형성된 기가 바이트의 정보에 가까워 졌다. (이 글을 쓰고 있는 지금 1 백 2 십만명의 플레이 정도의 규모이다.)

사실, 데이터의 양은 데이터 처리 도구가 감당할 수준을 넘어섰다 ; 나는 만 3 천명의 플레이어로부터 스냅샷 통계치를 구했다. 그러나 그 이후에, 많은 도구가 실패했다. 희소식은 처음 만 3 천명의 플레이어를 통해 집적한 데이터가 더 작은 테스트 그룹과 유사하였다는 것인데, 이는 테스트 그룹 결과가 더 많은 플레이어 그룹에 적용될 수 있다는 것을 의미한다.

어쨌든, 효과 있는 계획

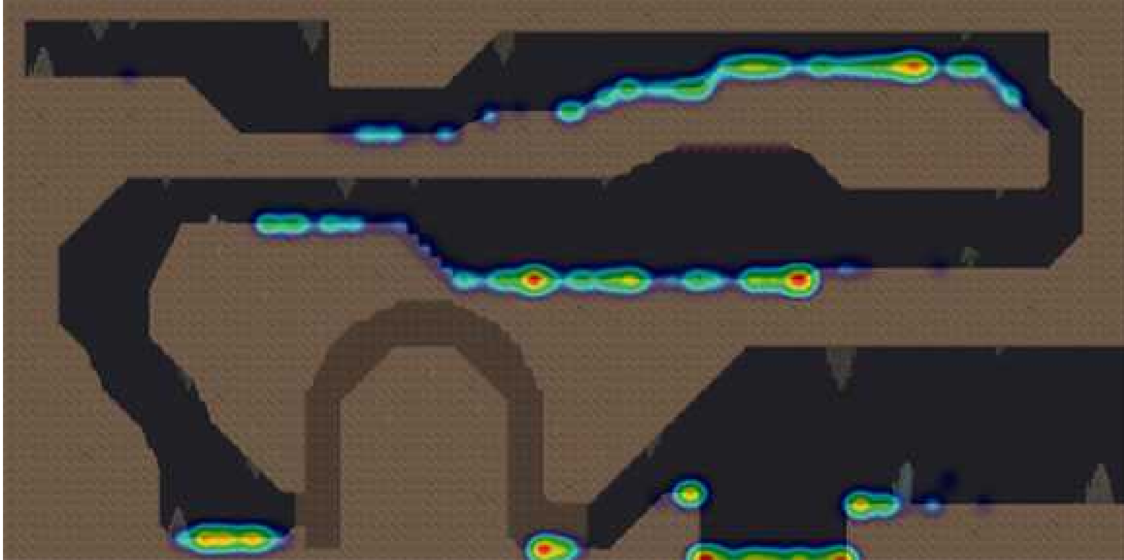
나는 *Replica Island* 이벤트 레포팅 시스템에 만족하였다. 적은 노력과 적은 비용 및 오직 두 형태의 이벤트 만으로, 나는 플레이어가 문제를 가지고 있는 지점을 효과적이고 빠르게 알아 낼 수 있었다. 게다가, 이 데이터를 모으기 시작하자, 나는 메트릭스 버전 간의 결과를 비교할 수 있었고, 이는 디자인의 변화가 효과적인지 아닌지를 더 쉽게 알 수 있게 해 주었다.

백 엔드 서버 언어로 PHP 와 MySQL 를 이용한 것은 좋은 선택이었다 ; 실제로 이벤트 기록은 매우 사소해서 어떤 언어도 가능하다고 확신한다. 그러나 PHP 를 이용하면, 전체 서버 데이터를 30 분 내로 통합할 수 있다.

각각의 쓰레드를 이용하는 것 또한 좋은 움직임이었다. 나는 어떤 종류의 UI 가 HTTP 요청을 막는 것을 원하지 않았다. 또한 각각의 쓰레드가 웹 커뮤니케이션이 움직이는 것을 이치에 합당하도록 하고 싶었다. 그러나 처음에 나는 오버 헤드의 문제에 대하여 걱정하였다. 그러나 오버헤드가 너무 작아서, 프로필에도 쓰지 못할 정도로 나타났다.

시스템을 가능한한 간단하게 유지하는 것은 정말 긍정적인 결과를 초래한 결정이었다. 나는 많은 잠정적인 이벤트 후보에 대하여 고민하였지만, 내 게임에서는 플레이어 죽음과 단계 완료를 따라가는 것 만으로 충분히 많은 정보를 제공하였다. 더 많은 통계는 데이터 처리를 복잡하게 하고, 명확한 시각을 가질 수 있도록 피드백을 줄이는데 어려움을 줄 것이다. 나는

이제 자동 메트릭스 레포팅에 대한 경험을 가지게 되었다. 앞으로는 좀더 많은 데이터를 받을 수 있게 할 지도 모르겠다. 그러나 간단하게 시작하는 것은 분명 좋은 움직임이었다.



시행착오

이벤트 레포팅 시스템에 관하여 모든 것을 잘 하지는 못했다. 완전히 형편없고, 시간 낭비만 하는 결정을 하기도 했다.

레포팅 서버에 PHP 를 이용한 결정은 좋은 선택이었다. 그러나 PHP 를 이용하여 데이터를 처리하고자 한 것은 실수였다. 나의 아이디어는 웹 대시보드를 이용하여 모든 것을 하려는 것이었다(심지어 나는 레벨 에디터를 PHP 와 자바스크립트로 작성했다). 그러나 PHP 는 내가 필요로 하는 데이터의 양이 너무 많을 때 급격히 추락했다. PHP 는 엄격한 메모리를 활용했고, 일정한 속도를 요구했는데, 나는 이러한 제안적 상황에 즉각적으로 대응하지 못했다. 2 만명의 이용자를 넘어서자, 거의 모든 PHP 베이스의 처리 도구가 멈추었다.

비트맵 처리는 특히 PHP 에 취약했다. 나는 모든 히트 맵 제너레이션을 PHP 로 했지만, 웹 서버 대신에 지역적으로 운영될 수 있는 어떤 것을 작성했어야만 했다. 나는 PHP GD 인터페이스에서 수 많은 버그를 접했고(알파로 작성된 비트맵은 약하다), 처리를 위해서 레벨 아트 이미지의 사이즈를 줄였다.

나는 파이톤과 이미지 매직을 활용하는 도구를 새로 만들었고, 결과는 훨씬 좋았다. 나는 이 실행을 위한 코드를 제공했고, 이것은 공식 Game Developer 메거진 웹사이트에서 볼 수 있다.

이 데이터가 나에게 플레이어가 어디에서 죽고, 레벨을 완료하는데 얼마나 많은 시간이 걸리는지를 말해 준다고 하더라도, 죽음과 관련있는 선반 상태를 규명하는데 도움을 주지 못한다. 나는 메트릭스가 절대로 잡아 주지 못하는 몇 가지 핵심 레벨 디자인의 실패를 가지게 되었다. 가장 지독한 경우에, 플레이어는 퍼즐에 잡혀서 어떻게 진행해야 하는지 이해하지 못하게 되었고, 결국 레벨을 완료하기 전에 포기했다.

이 같은 경우는 메트릭스에 나타나지 않는데, 이벤트 상태가 절대 도달하지 못하기 때문이다 ; 나는 이용자가 같은 지점에 갇혀 있다는 것을 불평할 때가 되어서야 이것을 알게 되었다. 자동 메트릭스는 매우 유용하지만, 이것은 게임에 대한 완벽한 시각을 주지 못할 수 있다. 내 경우에, 메트릭스는 문자가 있는 레벨 레이아웃을 찾는데는 좋았지만 룰 커뮤니케이션과 관련된 디자인 실패를 규명하기 위한 특정한 비효율성을 찾는데는 도움이 되지 않았다.

향후

다음 게임에, 나는 분명히 자동 메트릭스 레포팅을 다시 이용할 것이다. 죽은 위치 뿐 아니라. 적음의 형태에 기반한 이벤트를 더 할 것이다 ; 플레이어가 어디서 죽는지가 아니라 정확하게 어떻게 죽는지를 아는데 도움이 될 것이다. 그리고 게임에 의존하여, 죽음 전의 위치 변화에 대한 추적을 제공하는 것에 유용할 것이다. 이로써 개인 플레이어의 경로가 추적될 수 있다.

그러나 이러한 종류의 시스템의 핵심은 단순성이다 ; 데이터를 수집하는 것은 이후에 이것을 처리하는 도구가 신뢰성을 가지지 못한다면 유용하지 않다. 다음 게임을 위해서, 나는 기본적인 레포팅을 넘어서서 메커니즘만 저장하고 수 많은 문제를 해결하기 위한 더 좋은 도구를 만드는데 시간을 쓰고자 한다.

나는 또한 플레이 메트릭스로의 형태로부터 집적된 아웃풋이 런 타임 다이나믹 시스템에 정보를 제공해 줄 수 있는지 의문스럽다.

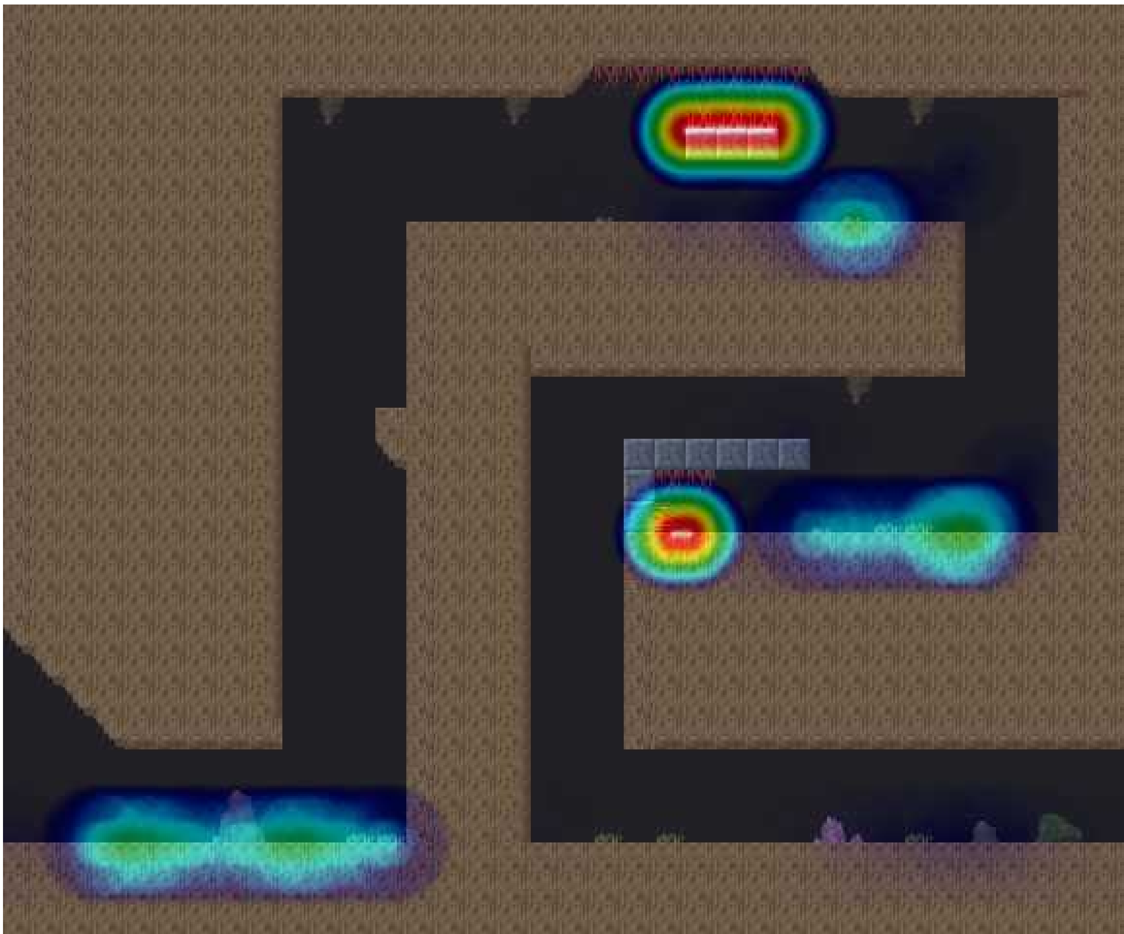
게임이 서버로부터 집적된 아웃을 읽는 것이 가능하다면, 싱글 플레이어의 플레이에 기반할 뿐만 아니라 수 백만의 평균적인 습관에 기반하여서도 변화를 만들 수 있다. 이 데이터의 가용성은 모든 종류의 가능성을 열어 준다.

플레이어 메트릭스는 유저 테스트를 대체할 완벽한 것이 아니지만 이것은 꽤 유용하다. 많은 유저 그룹을 테스트 하는 것이 개인 테스터 몇 명을 대상으로 하는 것보다 나을 것이며, 이것을 가능하게 해 주기 때문에, 결국에 메트릭스는 당신의 게임에 관하여 더 많은 것을 당신에게 알려줄 수 있다.

이익에 대한 비용은 *Replica Island* 의 경우 매우 긍정적이었다. 런 타임과 서버 데드를 간단하게 유지하면서, 나는 레벨 디자인과 플레이어의 습관에 대하여 많이 배웠고, 게임은 더 좋은 결과를 가지게 되었다. 유일한 후회는 내가 이 같은 종류의 시스템을 좀더 일찍 도입하지 못했다는 것이다.

히트 맵을 만드는 방법

히트 맵을 만드는 것은 어렵지 않다. 그러나 정확한 절차로 정보를 찾는 것은 어렵다. 나는 아래에 설명된 것과 유사한 방법을 활용했다.



기본적인 절차는 다음과 같다 :

방사형의 경사도로 가장자리를 투명하게 하고, 중심부의 검은색에서 회색 스케일로 변하게 되는 원을 준비한다. 이것이 이벤트 스팟 이미지 이다.

컬러 경사 이미지를 준비한다. 바닥은 흰색, 빨간색 또는 "가장 강렬한" 것을 나타낼 수 있는 당신이 원하는 색을 사용해야만 한다. 이미지의 상부는 검은색이고, 그 사이에 몇 가지 다른 색을 사용해야 한다. 이 이미지는 나중에 결과를 색깔로 색인할 수 있도록 활용될 것이다.

이벤트 위치 리스트를 만든다.

최대치의 오버래핑 데이터 포인트를 계산한다 (가령, 가장 일반적인 xy 포지션으로 발생하는 이벤트의 수를 계산한다). 이것은 최대 히트 값이다.

이벤트 리스트의 각각의 위치는 스팟 이미지로 캔버스에 그려진다. 이미지를 ((이 위치의 이벤트 수) / (최대 히트) * (100%)) 불투명함으로 그려라. 증가 이송 모드 (src * dest)를 이용하여 각각의 지점을 캔버스에 섞어라.

완료되었을 때, 당신은 다양한 그림자가 나타난 곳에 검은색 지점이 있는 이미지를 가지게 되어야만 한다. 이것은 중간 아웃풋 이미지이다.

아웃풋 이미지를 가지고 색 테이블을 이용하여 이미지를 다시 만든다. 각각의 픽셀의 알파 값을 가지고 Y 오프셋을 찾는데 이용하라.

결과 이미지를 가지고 레벨 아트에 겹쳐서 섞어라. 이벤트 핫 스팟은 색깔이 있는 부분으로 나타나게 될 것이다. 색깔의 강도에 따라서 더 진한 쪽이 더 많은 이벤트가 발생한 지역이다.

이 작업을 할 때, 색깔 공간을 유지하도록 확실히 해라 (특히 불투명도의 계산을 5 단계로 나누어라). 채널 마다 8 비트의 범주 이내로 정하고 또는 프로팅 포인트 픽셀을 지원하는 포맷을 활용하는 방법도 고려해 보아라. 싱글 이벤트가 1%로 떨어지는 너무 많은 데이터 이벤트가 있을 때에만 나타나는 정확한 버그를 만드는 것이 쉽다. ImageMagick 과 같은 도구가 이것을 하는데 도움을 줄 수 있다.