포스트모템: 인텔리전스 엔진 디자인 시스템즈의 <씨티 컨퀘스트>

(Postmortem: Intelligence Engine Design Systems' City Conquest)

작성자: 폴 토저(Paul Tozour)

작성일: 2013년 2월 6일

인텔리전스 엔진 디자인 시스템즈(Intelligence Engine Design Systems)[1]와 우리의 첫 번째 게임 <씨티 컨퀘스트(City Conquest)>[2]는 놀라운 개인적 여행의 결과로 탄생했다.

나는 1994 년 브로더번드(Brøderbund)[3]에서 실시간 전략게임 수석 디자이너로서 일을 시작했다. 그곳을 그만둔 후 나는 새로운 게임, 즉 내가 지금도 작업하고 있는 거대하고 서사시적이며 터무니없을 정도로 야심찬 꿈의 게임 디자인 컨셉트를 생각해 두었다. 나는 언젠가 그 꿈을 실현할 수 있으리라는 것을 알고 있었지만, 아직은 그럴 능력이 없었다는 것도 알았다. 내 꿈의 게임은 엄청난 양의 인공지능(AI)을 필요로 했고, 그 게임을 적절하게 기획할 수 있으려면 내가 인공지능에 대해 훨씬 더 깊이 이해해야 했다.

나는 <메트로이드 프라임(Metroid Prime)> 2 와 3, <메크워리어 4(MechWarrior 4)> 같은 게임의 인공지능 개발을 돕고, 게임 인공지능에 대한 기사를 쓰고, 길잡이용 항해망[4]의 사용을 독려하고, AIIDE 컨퍼런스에서 발표[5]를 하고, 꿈의 게임을

---

[1] 참조링크: http://www.intelligenceenginestudios.com/

[2] 참조링크: https://itunes.apple.com/us/app/city-conquest/id566901344?mt=8

[3] 참조링크: http://en.wikipedia.org/wiki/Broderbund

[4] 참조링크: http://www.ai-blog.net/archives/000152.html

디자인하기 위해 알아야 하는 모든 것을 배우는 동안 프로그래머로서 작업하는 등, 게임 인공지능에 대해 배울 수 있는 모든 것을 배우는 데 꼬박 10 년 이상을 투자했다. 이 과정에서 나는 게임 산업의 가장 숙련된 디자이너 몇몇에게 가르침을 얻을 수 있었다.

이 과정을 마치고 난 후 마침내 나는 꿈의 게임을 디자인하기 위해 알아야 하는 것들을 배웠다. 인텔리전스 엔진 디자인 시스템즈는 아직도 꿈의 게임을 디자인하고 있고, 그 게임을 개발할 수 있을 정도로 자금 수준과 회사의 크기가 성장하고 있다.

그런데 이 과정에서 예상하지 못했던 일이 발생했다. 나는 인공지능이 내가 게임 디자인에 대해 알고 있다고 생각했던 모든 것을 바꾸리라고는 생각하지 못했다.

인공지능과 게임 디자인에 대해 많은 것을 알게 될수록 나는 이 두 가지가 동전의 양면 같은 것이라는 사실을 알게 되었다. 게임 디자인은 플레이어들이 활동할 수 있는 가능성의 공간을 창조하는 것이고, 게임의 인공지능은 그 가능성의 공간 안에서 탐험하고 활동하기 위한 시스템(보통은 게임의 특징이나 본질에 대한 것이지만 때로는 별개의 도구에 대한 것이기도 하다)을 개발하는 일과 관련된 것이다. 인공지능을 현명하게 사용하면 기획을 개선하고 기획 결정의 분기를 더 빨리 탐색하는 데 도움이 되는 도구와 기술을 엄청나게 많이 얻을 수 있다.

게임을 기획하면서 직면하게 되는 문제들을 고찰하게 될수록 나는 많은 문제들이 근본적으로 인공지능에서 해결하는 문제들과 매우 비슷한 결정 최적화(decision optimization)의 문제이거나, 이미 다른 엔지니어링 분야들에서 많이 쓰였던 결정 모델링 과정들로 최적화할 수 있는 조합 최적화(combinatorial optimization)의 문제라는 사실을 알게 되었다. 결국 우리가 디자인 과정을 최적화하기 위해 다른 종류의 도구들을 만들어낼 수 있는 매우 주목할 만한 미지의 몇 가지 가능성들이 있다.

게임 디자인은 학습과 탐색의 과정이다. 모든 게임 디자인은 발전한다. 그리고 아무리 집중되고 잘 계획된 프로젝트들도 올바르게 진행하기 위해 디자인 과정을 수없이 반복한다. 잘 진행되는 프로젝트들은 디자인 과정의 반복과 실험에

---

[5] 참조링크: http://intrinsicalgorithm.com/IAonAI/tag/paul-tozour/

들어가는 비용에 맞서, 디자인 과정의 발전과 그것이 게임의 품질에 미치는 영향 사이의 균형을 맞추는 방식으로 기획 영역을 탐색할 수 있다. 잘 진행되지 않는 프로젝트들은 창조력의 낭비, 집중적이지 않은 디자인 탐색, 또는 끝없는 선(先)디자인 속에서 갈피를 잡지 못한다

게임 디자인은 어둠 속에서 싸우는 과정이다. 우리는 게임을 더 재미있고 매력적이며 몰입할 수 있게 또는 중독성 있게 만들 수 있는 디자인 과정의 변화를 불러오기 위한 선택지와 아이디어를 수없이 많다. 그러나 우리에게는 게임 디자인 과정에서 변화하고 추가되는 것들의 모든 세부를 정확하게 예측할 수 있는 능력이나, 우리가 고려하고 있는 변화들을 실제로 구현하지 않고 게임의 특성이 어떻게 변할 것인지를 진정으로 이해할 수 있는 능력이 거의 없다.

우리에게는 우리의 상상력과 실제로 일을 진행하고 실험을 하고 원형을 만들고 우리의 아이디어를 테스트할 수 있는 능력 이상으로 기획 영역을 탐색하는 도구들이 거의 없다.

전문적인 비행기 디자이너들에게는 비행기의 성능을 측정할 수 있는 "가상의 바람 동굴"을 제대로 실험하기 위한 강력한 엔지니어링 도구들이 있다. 비행기 디자이너들은 실제로 원형을 만들어 물리적인 바람 동굴에 띄워야 했던 훨씬 전부터 CAD 같은 툴을 사용해 빠르고 쉽게 비행기의 모형을 만들 수 있었고, 디자인 과정이 비행기의 성능에 어떻게 영향을 주는지 확인할 수 있었다.

게임 디자인에는 이런 것이 없다. 우리에게는 핵심 디자인 결정들의 모든 세부를 보여주는 도구들이 없다. 우리는 게임 디자인에 대한 현재의 매우 인간 중심적인 접근법을 넘어서 성장해야 하고, 다른 산업 분야에서 많이 그랬던 것처럼 인공지능의 도움을 받는 과정의 필요성을 인정해야 한다. 게임 산업에 "인공지능 포토샵"[6]은 필요가 없다. 게임 디자인을 위한 가상의 바람 동굴이 필요하다.

그래서 나는 그것을 만들기로 했다.

---

[6] 참조링크: http://games.soe.ucsc.edu/events/event/125

내가 와튼스쿨(Wharton School)의 공동 후원을 받는 펜실베이니아 대학 기술경영 과정에서 MSE 학위를 받고 있던 2011 년 8 월에 <씨티 컨퀘스트> 기획을시작했다. 와튼스쿨의 마케팅 수업에서 했던 시장 분석을 통해 나는 <스타크래프트(StarCraft)> [7] 같은 실시간 전략게임의 깊이와 <킹덤러쉬(Kingdom Rush)> [8] 같은 타워 디펜스 게임(tower defense games)의 요소들을 혼합한 게임 시장에 기회가 있다는 사실을 알게 되었다.

나는 <씨티 컨퀘스트>를 단순성과 용이성, 중독성을 혼합하고, 실시간 전략게임의 깊이와 전략적인 요소들과 타워 디펜스 게임의 느낌을 혼합한, 타워 디펜스 게임과 실시간 전략게임의 혼합체로 기획했다. 이 게임은 각각의 플레이어가 주어진 수도 건물을 방어하고 매 턴마다 특별한 "드랍쉽 패드(dropship pads)"에서 새로운 유닛을 생산하면서 전면전을 벌이는 타워 디펜스 게임으로 느껴질 필요가 있었다.

---

[7] 참조링크: http://www.starcraft2.com/

[8] 참조링크: http://www.kingdomrush.com/

목표는 그것이 "연습용"으로 급조된 게임을 만드는 것이 아니라 프랜차이즈로 성장할 가능성을 가진 진지한 게임을 개발하려는 노력이라는 점을 확실히 함과 동시에, 개발 주기를 1 년으로 제한하는 것이었다. 포화상태에 이른 최근의 모바일게임 시장에서는 야심차고 세련되며 독특해지려는 최소한의 시도도 하지 않는 개발 노력에 시간을 낭비할 여유가 없다.

<씨티 컨퀘스트>에서 우리의 목표는 "비용 대 품질의 비율"을 최적화하는 것, 즉 최소의 비용으로 최고의 게임을 만드는 것이었다. 레트로 스튜디오(Retro Studios)와 닌텐도에서 일했던 경험은 우리에게 게임의 품질과 세련도에 대한 강박적인 집중과 이런 근면성이 장기적인 이익으로 이어지리라는 확신을 심어주었다.

동시에 우리는 작업 비용을 최소화함으로써 위험 요소들을 줄이기로 했다. 즉 팀을 필요 이상으로 키우는 것을 피했고, 게임의 디자인과 제작, 엔지니어링의 대부분은 우리가 맡고 모든 예술 작업과 음향 작업은 외부 회사에 맡겼다. 위험 요소를 줄이는 데 중점에 두는 이러한 태도 덕분에 인텔리전스 엔진 디자인 시스템즈는 <씨티 컨퀘스트> 개발 기간 내내 매우 소규모의 팀을 유지했다.

우리는 한달 만에 플레이할 수 있는 기초적인 원형을 만들었다. 게임의 컨셉트는 우리가 예상했던 것보다 잘 작동했다. 어떤 특정한 이야기 구조나 "하이 컨셉트" 대신 게임의 전개를 이끌어가는 근본적인 게임플레이를 사용하기로 한 것은 게임을 재미있고 플레이할 만한 것으로 만드는 데 매우 큰 도움이 되었다.

제작 과정은 우리가 작업했던 가장 잘 진행된 트리플 A 급의 프로젝트들보다도 놀라울 정도로 순조롭게 진행되었다. 도중에 몇 가지 실수와 명백한 실사(實査)의 실패들이 있었지만, 게임의 최종 품질에 나쁜 영향을 주는 것은 없었다. 게임의 재미는 개발 초기 몇 개월만에 빛나기 시작했고, 나는 내가 이 게임에 완전히 빠져 있어서 게임을 플레이하느라 실제 작업 과정에서 주의가 산만해졌다는 것을 알게 되었다.

우리는 이것을 고무적인 신호로 받아들였고, 이것은 그 어떤 것보다도 <씨티 컨퀘스트>의 디자인을 처음부터 끝까지 지켜보도록 우리를 밀어붙였다.

나는 게임을 개발하는 동안 "게임의 재미를 찾으려고" 노력하는 데 수개월에서 1 년을 투자했던 몇몇 팀들과 작업했다. 이것은 혼란스러운 일이다. 아직도

재미있는 요소를 찾지 못했다면 도대체 왜 개발에 참여하는가? 프로젝트는 처음에 어떻게 청신호가 켜졌는가? 여전히 사전제작(preproduction) 과정에 있으면서 왜 그것을 개발이라고 하는가?

<씨티 컨퀘스트>는 예상보다 몇 개월 늦게 완성되었지만, 지체의 대부분은 우리가 품질을 최적화하기로 하고 게임을 더 좋게 만드는 데 시간을 투자했기 때문이라는 정당한 이유가 있었다. 우리는 꼭 필요할 때가 아니라면 새로운 기능을 추가하지 않음으로써 프로젝트의 범위를 제한하려고 주의를 기울였지만, 이미 존재하는 기능들을 다듬고 플레이테스터들의 피드백을 처리하는 데 엄청난 시간과 노력을 투자했다. 우리는 끊임없이 게임의 품질을 향상시킬 변화들을 놓치는 것과 일정에 차질이 생기는 것을 원치 않았다.

잘된 점

1. 인공지능의 보조를 받은 디자인 과정

인공지능에 기초한 우리의 디자인 접근법은 확실히 성공적이었다. 이 접근법이 우리의 기대와 향상된 품질, 제품이 시장에 나오는 시간을 앞질렀다는 데에는 의문이 없다. 이것은 어떤 고답적이고 학문적인 허울이 아니라 현실적이고 경쟁적인 이득이었다.

우리의 디자인 과정을 뒷받침했던 전체 이론은 포스트모템의 범위 안에서 평가하기에는 너무 복잡하다. 시간이 허락한다면 나중에 이것을 설명할 수 있게 되기를 바란다. 그러나 <씨티 컨퀘스트>에 영향을 주었던 디자인 접근법의 두 가지 주요 요소들에 대해서는 논할 수 있다.

첫 번째는 핵심 디자인 과정에 대한 최적화 기반의 접근법이었다. <씨티 컨퀘스트>의 디펜시브 타워와 유닛들은 무작위로 결정된 것들이 아니다. 그것들의 역할을 둘러싼 거의 모든 결정이 뚜렷한 결정 모델링 과정에 의해 진행되었다.

우리는 모든 디펜시브 타워와 유닛의 디자인 목표와 제약들을 명확하게 구체화한 후, 모든 유닛과 디펜시브 타워의 특성들을 전체적으로 함께 최적화하기 위한 결정 모델을 만들었다.

그런 다음 우리가 가진 디자인 제약들 안에서 우리의 디자인 목표를 만족시킴과 동시에 가장 잘 작동하는 아홉 가지 디펜시브 타워와 유닛의 최상의 조합을 선택하기 위한 진화 최적화 장치를 사용했다.

이 접근법은 훨씬 단순하고 게임 디자인 결정에 최적화된 것이긴 하지만 <Decision-Based Design>[9]라는 책에서 언급된 접근법과 대체로 비슷하다.

우리는 이런 최적화 기반의 접근법이 게임 개발자들에게 큰 도움이 될 수 있다고 굳게 믿는다. 이 접근법은 디자인의 복잡한 문제들을 감소시키고, 최선의 결정을 더 빨리 내릴 수 있게 해주고, 어떤 경우에는 해결할 수 없는 문제들을 해결할 수 있게 해줌으로써 구매자들에게 더 높은 가치를 제공하는 데 도움이 된다.

두 번째 이점은 예전에 AIGameDev.com 와의 인터뷰[10] 에서 논한 바 있는 에볼버(Evolver)였다. 에볼버는 공동 진화의 유전적 알고리즘을 기반으로 한 자동화된 균형 조절 도구다. 매일 밤 에볼버는 적색 군대와 청색 군대 각각이 스크립트의 "개체수(population)"(각각의 스크립트는 본질적으로 게임 내 건물들의 고정된 건설 순서이다)를 진화시키면서 그들 사이에서 벌이는 모의 전투를 수없이 실행했다.

에볼버는 무작위의 스크립트를 생산해 상대편에 맞서 실행하고, 얼마나 획득하느냐에 따라 플레이어의 승패가 갈리는 적당한 점수를 스크립트에 배정한다. 그런 다음에는 개체수 각각을 유전적으로 최적화하는 크로스오버와 돌연변이 같은 표준적인 진화 실행장치를 사용한다.

그래서 우리는 매일 아침 일어나 에볼버를 실행하고, 어떤 스크립트를 적색 플레이어와 청색 플레이어가 가장 많이 획득했는지를 측정한 다음, 그것들을 게임에 입력하고 상대편에 맞서 실행되는 것을 지켜볼 수 있었다. 이것은 게임의 밸런스 컨트롤이 어떻게 작동하고 있는지를 즉시 알려주었다. 즉 플레이어가 로켓 발사대를 너무 많이 건설했는가, 스카이스크래퍼(Skyscrapers)는 충분히 건설했는가,

---

[9] 참조링크: http://www.amazon.com/Decision-Based-Design-ebook/dp/B00A9YH096/ref=sr_1_1?ie=UTF8&qid=1354819595&sr=8-1&keywords=decision-based+design

[10] 참조링크: http://aigamedev.com/open/interview/evolution-in-cityconquest/

특공대는 충분히 유용했는가 아니면 시종일관 건쉽보다 크루세이더를 선호했는가 같은 것들 말이다.

그런 다음 우리는 매일 이 결과물에 유닛과 건물의 자원값, 체력, 속도, 데미지, 화력, 그리고 그 밖의 변수들에 변화를 주면서 몇몇 유닛과 건물들을 조율하고 다듬는 데 썼다. 이것은 밤새도록 게임을 플레이하는 외부의 테스트 팀을 운영하는 것이나 마찬가지였다. 다만 사람이 테스트하는 것보다 완전한 객관성을 얻을 수 있었고, 비용이 적게 들어가고 테스트에 난조가 적게 발생했다.

우리는 렌더링을 불가능하게 하고, 업무 기반 병렬 처리(task-based parallelism)[11]를 추가하고, 게임플레이의 논리를 수작업으로 최적화함으로써 에볼버를 최적화했다. 이것은 12 시간당 약 100 만 번의 게임 실험을 할 수 있게 해주었다. 나중에 우리는 섬 모델(island model) [12] 을 사용하기 위해 유전적 알고리즘을 업그레이드했고, 특정한 결과(스크립트 개체수가 장기적인 최선의 결과 수준을 달성하기 위해 재빨리 스카이스크래퍼를 업그레이드하도록 돕는 것 같은 것들)를 얻기 위해서 적당한 기능을 수작업으로 조율했다.

이것은 번거로워 보이지만 그렇지 않았다. 에볼버를 만들고 조율하고 최적화하는 작업은 전체 개발 기간 중 약 2 주 정도가 걸렸다. 에볼버가 우리에게 주었던 귀중한 모든 피드백, 수작업으로 했을 때 보다 더 좋은 결과를 내주었다는 사실, 그리고 수작업으로 했던 초기의 조율 작업이 에볼버로 작업했을 때보다 2 주 이상 소요되었다는 사실을 고려한다면 이것은 명백한 최종 승리였다고 생각한다.

또한 에볼버는 디자인 과정에서 발생하는 모든 변화의 세부 사항을 확인하기 위한, 게임플레이의 변수에 발생할 수 있는 모든 변화들을 테스트하기 위해 재빨리 실행할 수 있는 시스템을 남겨주었다. 한 예로 우리는 롤러의 값을 크리스탈 두 개에서 한 개로 줄일 때(동시에 롤러 유닛을 스탯도 줄였다) 발생할 수 있는 문제들을 재빨리 확인했는데, 에볼버는 즉시 문제를 발견하고 이런 생각이 게임플레이에 문제를 야기하기 전에 그것을 포기하게 해주었다.

그리고 출시일까지 8 개월 동안 우리에게 에볼버가 줄 수 없었던 귀중한 미적 요소와 유용성, 그리고 그 밖의 주관적인 피드백을 제공했던

---

[11] 참조링크: http://threadingbuildingblocks.org/

[12] 참조링크: http://www.gustafsonresearch.com/thesis_html/node105.html

테스트플라이트(TestFlight)[13] 팀의 수많은 플레이테스터들도 큰 도움이 되었다. 우리는 결국 우리의 모든 킥스타터(Kickstarter) 후원자들을 플레이테스터로 초청했다.

이 모든 요소들이 작용한 결과 게임은 거의 첫날부터 재미있었다. 모든 디자인 컨셉트들이 효과를 발휘한 것이다.

2. 버그를 즉시 고쳐라

나는 예전에 버그가 5,000 개 이상에 달하는 몇몇 프로젝트에 참여한 적이 있다. 팀이 최종적인 "버그 치료" 단계로 접어들면 불가피하게 끔찍한 버그들을 수없이 많이 발견하게 된다. "이런, 버그가 있다는 것을 알았더라면 시간을 훨씬 절약할 수

[13] 참조링크: https://testflightapp.com/

있었을 것을!" "그 버그 하나가 다른 버그 10 개로 이어졌어!" "그 버그를 6 개월 전에 치료했었더라면 레벨 디자인하는 방법을 완전히 바꿨을 텐데!"

기억할 만한 어떤 프로젝트에서 프로듀서가 자신의 팀에게 다음과 같이 말했다. "버그를 치료하지 마. 지금 당장은 그럴 시간이 없어! 마지막까지 남겨 두라고!" 몇 개월 후 프로젝트는 비참하게 실패했고, 스튜디오가 파괴됨과 동시에 전체 프랜차이즈를 철수하는 지경에 이르렀다. 적지 않은 버그가 디자인 시작 단계에서 발생한 엄청난 양의 버그에서 비롯되었다.

이런 일이 발생해서는 안 된다. 소프트웨어의 결함을 남겨두는 것은 일정과 코드베이스의 완전성에 위험요소가 된다.

우리의 접근법은 버그를 즉시 치료하는 것이었다. 우리는 발견된 버그들이 모두 치료될 때까지는 새로운 특성을 추가하거나 다른 개발 작업을 진행하지 않았다. 디자인 과정의 결함과 실행상의 문제들은 "버그"로 취급되었다. 플레이터스터들의 제보도, 특히 한 사람 이상이 같은 문제를 보고하거나 같은 변화를 제보할 경우 "버그"로 취급되었다.

개발 과정 내내 문제가 되는 버그가 10 개 이하로 유지되었다.

개발을 마친 지금으로서는 이 방법 이외에는 다른 어떤 개발 방법도 상상할 수 없다. 단지 "일정"을 위해 버그 치료를 미루는 일에 무슨 의미가 있는가? 그것은 본질적으로 기만 행위이다. 즉 장부 한쪽에서는 일이 잘 돌아가고 있는 것처럼 보이기 위해 문제를 장부 다른 쪽에 제쳐두는 것이다. 이것은 게임의 완전성을 희생함으로써 보이지 않는 대가를 치르는 것이다.

플레이테스터들은 버그가 비교적 적다는 사실을 자주 언급했다. 그리고 이런 코드베이스의 완전성은 테스터들이 기술적인 문제보다 게임플레이에 집중할 수 있게 해주었다.

우리의 경험에 비춰보았을 때, 버그를 데이터베이스에 보관해두었다가 나중에 치료하는 관행은 생산 지연, 개발자의 스트레스, 일과 생활의 불균형, 그리고 스튜디오 실패의 주요 원인이다. 이것은 원시적인 관행이므로 사라져야 한다.

3. 발견이 이끄는 계획

펜실베이니아 대학 와튼스쿨에서 MSE 과정에 다니고 있을 때, 나는 운 좋게도 <Unlocking Opportunities for Growth>[14]의 공동저자 이안 맥밀란(Ian MacMillan) 교수의 강의를 듣고 "발견이 이끄는 계획(discovery-driven planning)"[15]에 대해 배우게 되었다. 그의 강의는 매우 놀라운 경험이었다.

발견이 이끄는 계획은 분명하게 혁신을 지향하는 프로젝트에 맞춰 만들어진 과정이며, 불확실성을 줄이는 데 초점을 맞추고 있다. 우리는 디자인 과정에서 발생한 모든 변화들이 불가피하다는 것을 알고 있기 때문에, 개발자들은 대체로 계획을 세울 때 학습결과를 반영하거나, 불확실성을 확인하고 줄이는 것을 목표로 하는 방법론을 사용하지 않는다.

우리는 스크럼(Scrum)과 그 밖의 "애자일(agile)" 방법론을 도입하고 상황이 변함에 따라 계획을 조정했다. 그러나 대체로 디자인 시작 단계부터 불확실성을 측정하는 일이나 가장 중요한 위험요소에 집중할 수 있게 해줄 방식으로 계획하는 일에는 별 노력을 쏟지 않았다.

<씨티 컨퀘스트>에서 발견이 이끄는 기획 시스템은 우리가 모든 가설들을 분명하게 밝히고 측정하고, 반대의 손익 계산서를 작성하고, 각각의 가설이 이윤에 가져올 영향을 세밀하게 분석하게 해주었다. 이 방법론은 프로젝트의 비용과 수입에 영향을 주는 불확실성의 가능성이 가장 큰 영역에 개발 노력을 쏟을 수 있게 해주었다.

그러나 발견이 이끄는 계획을 분명하게 사용하지 않는다고 해도, 어떤 주어진 순간에 ((불확실성) x (재정적 영향))의 값이 가장 높은 특성들이나 영역들에 노력을 집중시키는 일의 일반적인 원칙은 위험요소를 사전에 관리하면서 증가[16]를 오랫동안 보장할 수 있다.

---

[14] 참조링크: http://www.amazon.com/Unlocking-Opportunities-Growth-Uncertainty-Limiting/dp/0132237903/ref=sr_1_1?ie=UTF8&qid=1354892393&sr=8-1&keywords=unlocking+opportunities+for+growth

[15] 참조링크: http://en.wikipedia.org/wiki/Discovery-driven_planning

[16] 참조링크: http://discoverydrivengrowth.com/

4. 기술 선택

기술을 선택하는 것은 위험요소로 가득찬 일이다. 최근에는 그 사례가 많은데, <글리치(Glitch)>[17]의 경우 개발자들이 플래시를 선택한 것이 자신들의 프로젝트 목표를 달성하지 못하게 했다는 사실을 분명히 했다. 또 다른 명확한 사례에서는 개발자가 기술적인 지체와 결점들을 이유로 엔진 판매자를 고소했다.

우리의 초기 기술 선택은 게임의 성공에서 결정적이었다. 내가 가진 경험은 내가 인증받은 기술과 관련된 위험요소들 그리고 스크립트 언어의 실행과 생산성 영향에 극도로 집착하게 했다.

<씨티 컨퀘스트>는 4 세대 아이팟 터치 같이 비교적 평범한 몇몇 디바이스들을 비롯한 모바일 디바이스에서의 폭넓은 실행 요구를 담고 있다. 이 게임은 모든 플랫폼에서 실행되기 위한 광범위의 최적화와 세심한 주의를 요구했다.

우리는 마멀레이드(Marmalade)[18]를 이용해 게임을 제작하기로 했는데, 왜냐하면 마멀레이드는 엔진이 아니라 고성능의 가벼운 크로스플랫폼 API 레이어이기 때문이다. 마멀레이드는 우리 팀의 모든 프로그래머들이 C++를 이용해 윈도우 비주얼 스튜디오(Visual Studio)에서 코딩과 디버깅 작업의 95%를 해내고, 마멀레이드의 PC 에뮬레이터를 이용해 게임을 테스트하고, iOS 와 안드로이드에 동일한 코드를 배치할 수 있게 해주었다(마멀레이드가 매킨토시와 윈도우를 지원하게 됨에 따라 앞으로는 매킨토시와 윈도우에서도 가능할 것이다). 플랫폼이 정해진 멀티플레이 게임 코드의 일부 말고도 최초의 "데모 버전(first-playable)" 안드로이드 포트는 그야말로 한 시간 이하가 걸렸다.

마멀레이드에 결함이 없는 것은 아니다. 다른 도구들과 마찬가지로 마멀레이드에는 고유의 모순들이 있다. 온디바이스 디버깅(on-device debugging)의 부재는 때때로 문제가 되었다(그러나 실제로 필요했던 일은 극히 드물었는데, 왜냐하면 버그의 99%가 우리 때문에 발생한 것이었고 윈도우의 마멀레이드 에뮬레이터를 사용하는 비주얼 스튜디오의 디버깅 툴에서 복제가 가능했기 때문이다). 우리는 또한 자원 관리도 광범위하게 최적화해야 했다.

---

[17] 참조링크: http://www.glitch.com/closing/

[18] 참조링크: http://www.madewithmarmalade.com/

마멀레이드와 C++를 사용했던 것은 우리가 광범위한 모바일 디바이스에서 적절한 프레임레이트로 작동하는 그래픽이 매우 뛰어난 게임을 만드는 데 꼭 필요한 결정적인 최적화 작업을 할 수 있게 해주었다.

5. 세심한 비용 관리와 아웃소싱

우리는 너무 많은 스튜디오들이 너무 빨리 성장하고 공격적인 확장을 감당하지 못해 내부에서부터 파괴되는 것을 보았다. 이런 고용은 회사의 비용 구조와 수익성을 위해 필요한 손익 분기 수준을 극적으로 팽창시키고, 일반적으로 회사의 상태를 악화시킨다. 한 회사의 팀워크는 보통 팀의 규모보다 훨씬 느리게 성장하며, 스튜디오들은 모든 신규 채용에서 성공하게 해주는 문화와 원칙들을 확장하는 일에서 실패하는 일이 자주 있다.

우리의 작은 스튜디오 규모와 세심한 비용 관리는 작업 비용을 최소화하고 손익 분기점을 가능한 한 낮게 유지하게 해주었다. 우리는 천천히 성장하고, 적절한 사람을 고용하고, 위험요소들을 관리하고, <씨티 컨퀘스트>와 그 밖의 게임들을 안정적이고 끝없이 발전하는 장기적인 프랜차이즈로 만드는 일에 훨씬 관심이 많았다. 우리는 예술과 음향, 멀티플레이 게임 엔지니어링을 맡았던 하청업자들이 적정한 비용으로 놀랍도록 높은 수준의 품질을 생산해냈다는 것을 알게 되었고, 그들과 함께 작업할 수 있었다는 사실에 매우 기쁘다.

잘못된 점

1. 미션 설계

우리는 <씨티 컨퀘스트>의 개발을 시작하기 오래 전부터 이 게임이 플레이어들에게 게임플레이어의 컨셉트를 쉽게 이해할 수 있는 부분들로 나누어 가르쳐주고, 각각의 부분들을 그것들이 게임을 재미있게 해주는 데 도움이 되도록 사용하는 최고의 게임이라는 사실을 알고 있었다. 이는 게임 디자인의 기초(Game Design 101)로, 노련한 닌텐도 개발자로서 이것은 거의 제 2 의 자연이 되어서 그 이외의 생각은 거의 못하게 되었다는 것이 명백해졌다.

그러나 어쨌든 이상한 일이지만 우리는 15 분 동안 텍스트 팝업창을 무수히 띄워 게임플레이의 모든 주요 컨셉트(건물 건설, 업그레이드, 영토 관리, 드랍쉽 패드 교환, 게임 효과, 위장막, 골드와 크리스탈 자원 관리)를 가르쳐주는 튜토리얼 미션을 제작함으로써 시작했다.

내가 매우 견실한 튜토리얼 미션이라고 느꼈던 것을 개발하는 모든 작업이 끝난 후, 나는 외부의 테스터 한 사람에게서 잔인할 정도로 솔직한 피드백을 받았다. 아무도 재미를 보거나 실제로 게임을 하기도 전에 팝업창이 15 분 내내 뜨는 것을 보고 싶어하지는 않았다. 그리고 그것은 한꺼번에 너무 많은 정보를 주는 것이었다. 나에게 분명한 것으로 보였던 게임플레이의 컨셉트들이 플레이테스터들에게는 물론 분명하지 않았다.

그리고 마침내 튜토리얼 미션을 마치고 나면 플레이어는 바로 다음 미션에서 갑자기 쏟아지는 옵션들에 압도당했다. 플레이어는 곧바로 20 채의 모든 건물을 건설하고 *적절한* 사용법을 배우기도 전에 네 종류의 모든 모선(Mothership)들의 효과를 사용해야 했다.

결국 한숨을 돌리고 미션 설계의 요소들을 완전히 수정하는 것 말고는 달리 선택지가 없었다. 우리는 게임을 다섯 가지의 컨셉트 요소들(즉 공격하기, 방어하기, 스카이스크래퍼를 건설해 영토 넓히기, 게임 효과 사용하기, 드랍쉽 패드 교환하기)로 분리하고, 그 다섯 가지의 핵심 게임플레이 요소들 각각에 대해 가르쳐주고 그것들을 재미를 주는 방식으로 사용하는 적어도 하나의 특화된 미션을 만들었다. 또한 플레이어의 건물 선택을 미션 스크립트 안에서 제한된 부분집합으로 제한할 수 있게 해주고, 플레이어들이 너무 빨리 옵션들에 압도당하지 않게 하는 것을 도와줄 수 있도록 사용자 인터페이스를 점검했다.

2. 과금 제도

우리는 <씨티 컨퀘스트>를 iOS 에서 다운받을 수 있는 무료 버전과 앱 내에서 구입할 수 있는 4.99 달러짜리 풀 버전을 출시했다. 무료 버전은 전체 멀티플레이 게임 경험과 14 개의 싱글플레이 캠페인 중 초반 다섯 개의 미션, 그리고 여섯 개의 보너스 "도전" 미션 중 첫 번째 미션을 포함하고 있다.

처음에 우리는 무료 다운로드가 제한된 "라이트 버전"과 "풀 버전"으로 분리해 출시하기로 했었다. 그러나 마멀레이드 팀과 논의한 결과, 우리는 결국 두 버전을 하나의 무료 다운로드로 통합하고 게임 내에서의 결제를 통해 데모 버전과 풀 버전을 분리하는 편이 나으리라는 것을 확신했다.

이렇게 한 의도는 좋았다. 이 접근법은 유저가 게임을 한번만 다운받아도 되게 해주었고, "라이트 버전"과 "풀 버전" 사이에서 혼동할 가능성을 최소화해 주었고, 플레이어들이 무료 버전에서 풀 버전으로 전환하는 것을 쉽게 해주었고, 모든 플레이어들이 무료로 멀티플레이 게임을 할 수 있게 해주었고, 성과와 캠페인 완수가 라이트 버전과 풀 버전 모두에서 공유되게 해주었다.

돌이켜 보면 우리가 전환 비율에 만족하기는 하지만 이 접근법에 심각한 한계들이 있고, 충분히 발전한 과금 제도에 기반을 두지 않은 그 밖의 문제들이 있다는 사실은 분명하다. 모바일 시장의 특성은 사용자들의 전체 스펙트럼에 걸쳐 적절한 가격 차별을 두는 것을 매우 중요하게 만들고 있다. 과금 제도에 대한 아티클은 가마수트라에 많이 있으므로 참고하기를 바란다.

현재 우리는 <씨티 컨퀘스트> 안드로이드 버전을 거의 다 완성했다. 이것은 광고로 수익을 충당하는 무료 게임으로 출시될 것으로 예상된다.

3. 킥스타터

<씨티 컨퀘스트>에는 두 개의 킥스타터 광고가 있었다. 우리는 최초 목표[19]를 약 12,000 달러로 책정했었다. 이 목표가 너무 느리게 진행되자 우리는 그것을 취소하고 게임을 광범위하게 세련화하는 작업을 했다. 몇 개월 후, 우리는 더 좋은 피치(pitch)와 더 좋은 상품, 약 10,000 달러 이상의 광고를 가지고 돌아왔다[20].

좋은 소식은 우리가 재정적 목표를 달성했다는 것과 실제로 킥스타터가 약속했던 것보다 훨씬 좋은 상품을 생산했다는 사실이다.

나쁜 소식은 킥스타터에 그럴 가치가 없을 수도 있었다는 사실이다. 우리에게는 후원자가 146 명 뿐이었고 재정적 목표를 겨우 앞질렀을 뿐이다. 두 개의 킥스타터 광고와 프로모션 영상들을 만들고 관리하는 데 들어갈 시간에 게임을 제작하는 작업에 투자하는 편이 나았을 것이다.

---

[19] 참조링크: http://www.킥스타터.com/projects/paultozour/city-conquest

[20] 참조링크: http://www.킥스타터.com/projects/paultozour/city-conquest-tower-defense-evolved

우리에게 가장 문제가 되었던 것은 우리의 후원자들에게 풀 버전의 게임을 무료로 제공하겠다는 약속이었다. 그때 우리는 우리가 이 약속을 지키는 데 애플의 프로모 코드를 사용할 수 있으리라고 믿었지만, 가격 모델 계획을 "라이트 버전"과 "풀 버전"으로 분리하는 것에서 하나의 게임 안에서 앱 내 결제 시스템(IAP)으로 두 버전을 분리하는 것으로 바꾸면서 우리도 모르게 그 약속을 지킬 수 없게 되었다.

우리는 애플의 프로모 코드가 IAP 로 작동했던 몇 가지 소스들에 대해 확신했었으므로, 이 접근법에 대해 매우 자신만만했었다. 또한 이것이 실패한 경우를 위한 차선책들도 많이 있었는데, 여기에는 수령인들의 디바이스 UDID 를 하드코딩(hard-coding)하는 것 또는 게임 내에 별도의 쿠폰입력 다이얼로그를 제공하는 것이 포함되어 있다. 불행하게도 프로모 코드는 IAP 에서는 작동하지 않는 것으로 밝혀졌고, 우리는 애플이 별도의 쿠폰입력 다이얼로그, UDID 의 사용, 그리고 그 밖에 우리가 염두에 두었던 꽤 많은 차선책 각각에 대해 얼마나 심하게 못마땅해 했는지를 알게 되었다.

이것은 명백하고 불행스러운 실사의 실패였다. 우리는 즉시 후원자들에게 사과하고 후원금을 돌려주었다. 금전적인 영향은 수십 건 밖에 안 되는 후원금 반환을 1 인당 6 달러씩 돌려주는 것으로 약속함으로써 최소화되었지만, 이런 일은 여전히 일어나서는 안 되는 일이었다.

우리에게 있어 킥스타터의 진정한 가치는 많은 후원자들을 귀중한 피드백을 제공했던 플레이테스터로 바꿀 수 있었다는 점이었다. 만약 킥스타터를 다시 사용하게 된다면 우리는 그것을 플레이테스터를 구하는 데 쓰는 일에 초점을 맞출 것이며, 우리의 후원자들을 자금 조달 수단으로 쓰는 대신 그들과 소통할 것이다.

4. 엔지니어링의 도움을 너무 늦게 받은 것

2012 년 5 월 우리는 우리 팀이 멀티플레이 게임을 적절하게 구현하고 테스트하기에는 너무 폭 좁게 확장했다는 사실을 알게 되었다. 우리는 비교적 견실한 퍼스트 패스 멀티플레이 게임을 적절하게 구현했지만, 우리의 발견이 이끄는 계획은 멀티플레이 게임이 큰 위험요소이며 훨씬 많은 주의를 요구했다는 사실을 알려주고 있었다.

동시에 멀티플레이 게임은 협상의 대상이 아니었다. 즉 그것은 게임의 비전에서 본질적인 부분이었고, 그만두는 것을 고려하기에는 프로젝트에 너무 많은 가치를 부여했다.

우리는 우리의 개념 증명(proof-of-concept) 넷코드(netcode)를 진지한 멀티플레이 게임의 구현으로 교체하기 위해 풀 사이클 엔지니어링(Full Cycle Engineering) 팀을 운영했다. 지금 멀티플레이 게임이 그들의 도움을 받지 않고 구현했을 때 나왔을 결과보다 훨씬 나은 경험이라는 사실은 의심의 여지가 없다. 그러나 우리는 문제를 발견하고 훨씬 빨리 풀 사이클 엔지니어링 팀을 운영했어야 했고, 휴대폰으로 실험할 것이 아니라 개발 시작 단계부터 와이파이에 초점을 맞췄어야 했다. 관련 업무는 매우 많았고, 이런 개발 노력은 우리에게 도움이 필요하다는 것을 더 빨리 인정하려 했던 아주 초기 단계부터 폭넓은 도움을 받았어야 했다.

5. 실사의 실패

우리의 가장 큰 실수는 실사에서 저질렀던 몇 가지 기초적인 실수들이었다. 이 실수들 중 어떤 것도 결과적으로는 게임의 품질에 영향을 주지 않았지만, 우리는 이런 종류의 실수들을 저지르는 스튜디오가 되기보다 더 높은 기준에 우리 자신을 맞춘다.

앞서 언급한 킥스타터에서의 실수를 제외한 최악의 실사 실패는 아이팟 터치 지원이었다. 여기서 설명하기에는 복잡한 이유들 때문에 4 세대 아이팟 터치는 우리의 기준 iOS 디바이스가 되었고, 조력자들과의 논의와 우리가 직접 진행한 조사에서 4 세대 아이팟 터치가 아이폰 4 와 기본적으로 동일한 것으로 나타났다.

이것은 부정확했다. 4 세대 아이팟 터치는 확실히 열악한 디바이스로 메모리의 제한이 매우 심각하다. 이런 이유 때문에 우리는 꾹 참고 사소한 메모리 최적화 작업을 엄청나게 많이 해야 했다. 시간이 오래 걸리는 텍스쳐와 사운드의 메모리 사용을 축소하는 작업과, 재원 부담이 큰 우리의 시스템과 자원 관리 코드를 개선시키는 주요 작업을 하는 데 나흘에서 닷새가 걸렸다. 이런 수많은 메모리 최적화 작업들이 전체 게임에는 도움이 되었지만, 이런 문제들은 더 일찍 확인되고 해결되었어야 했다.

또한 우리가 멀티플레이 게임 로비를 위해 선택했던 외주(third party) 기술에도 문제가 있었다. 개발 후기 단계에서 우리는 이 기술을 둘러싼 해결하기 어려운 기술적인 문제들에 직면해야 했고, 이 기술을 아마존 웹 서비스(Amazon Web Services)에 기초한 해결책으로 대체해야 했다. 이 기술을 우리가 선택할 것이 아니라, 이것을 둘러싼 실사를 풀 사이클 엔지니어링 팀에게 맡기고 이 작업에 적합한 최선의 도구를 그들이 선택하게 하는 편이 나았을 것이다.

세번째 문제는 업적에 대한 것이었다. 개발 초기에 동료 개발자들 몇몇이 게임당 주어진 애플 게임센터(GameCenter) 업적의 숫자에는 제한이 없다는 사실을 알려줬었다. 실제로는 게임센터의 업적이 100 개로 제한되어 있다는 사실을 알게 되었을 때, 우리는 업적을 100 개 이하로 돌려놓기 위해 수많은 업적들을 제거해야 했다. 결국 이것은 좋은 결과로 이어졌다. 우리는 그저 며칠 정도만 낭비했을 뿐이며, 필요없는 것들 사이에서 중요한 것을 골라냄으로써 업적의 퀄리티를 전체적으로 향상시킬 수 있었다. 시간 낭비와 실사의 실패에 대해서는 후회하고 있지만, 결국 우리는 우리가 느끼기에 가장 가능성이 높은 100 가지 업적을 만들었다.

결론

인텔리전스 엔진 디자인 시스템즈는 색다른 종류의 인디 게임 개발회사이며, 우리는 게임 제작에 대한 완전히 다른 접근법을 탐색하는 일에 관심이 있다. 우리에게 게임 개발에 대한 어떤 궁극적인 해결책이 있다고 주장하는 것은 아니지만, 우리는 지금까지 우리가 우리의 접근법이 가져온 결과들과 게임의 성공에서 살펴보았던 것에 대해 매우 고무되어 있다. 우리는 <씨티 컨퀘스트> 최종 버전이 제공하는 게임 경험에 매우 자랑스러워 하고 있으며, 이것을 발전시키고 게임의 성공을 발판으로 삼아 속편을 개발할 것으로 기대하고 있다.