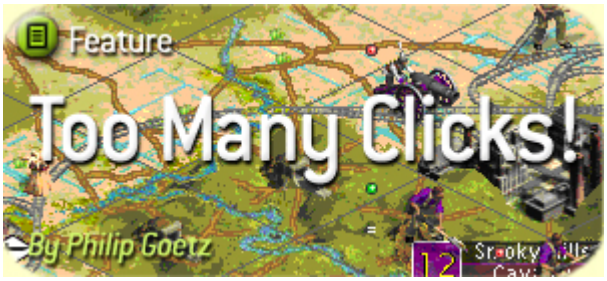
Gamasutra.com

클릭을 너무 많이하는 유닛 기반 인터페이스의 해로움

2006 년 8 월 23 일

http://gamasutra.com/features/20060823/goetz_01.shtml



소개

전통적으로 컴퓨터 게임들은 화면상에서 하나 또는 하나 이상의 유닛을 제어하는 플레이어를 가지고 있다. 초창기 게임에서 각 플레이어는 한 유닛을 제어했다. CPU 성능이 올라감에 따라 플레이어들은 하나 또는 하나 이상의 유닛을 제어하게 되었다. 오늘날 한 플레이어가 수 백 개의 유닛을 제어 할 수도 있지만 그것들 하나하나를 개별적으로 관리해야 한다. 유닛 기반 유저 인터페이스(UI)로는 더 이상 충분하지가 않다.

이 기사는 UI 에 대한 다양한 생각을 말하고 당신의 게임을 향상 시킬 수 있는지를 알아보기 위해 한 UI 와 다른 UI 를 비교하는 법 또는 한 UI 와 이론적으로 가장 효율적인 UI 와 비교하는 법에 대해서 상세히 설명 할 것이다. 나는 주로 전략 게임들을 가지고 예를 들어 설명하겠지만 이 방법은 모든 종류의 프로그램을 위한 UI 에서도 적용된다.

천 번의 클릭으로 한 턴 진입하기

내가 최고로 좋아하는 게임은 문명을 구체화 시킨 문명(Civilization)이라는 게임이다. 내가 문명이라는 이 게임을 친구들에게 소개시켜 주면 문명에서 약 천 턴이 천 년 또는 2 천년 동안 게임에 열중했다. 그때에는 도시들,유닛들과 기다리는 것이 너무나 많이 증가 되서 초심자에게 있어 그것은 게임 이상의 자질구레한 일이 되었다.

문명 중독자인 나 조차도 게임이 약 1800 년 후에는 그다지 재미있지가 않다. 너무나 많이 클릭 해야 한다. 나는 문명 III 에서 1848 년도에 한번의 움직임을 끝내는데 마우스 이동과 키스트로크 (자판을 치는 것),클릭수를 계산해보았다. 많은 시간 후에 그 턴이 끝났을 때 나는 422 번의 마우스 클릭과 352 번의 마우스 이동,290 번의 키 누름,23 번의

휠 스크롤, 화면을 스크롤 하기 위해 18 번의 화면 팬(pans)이 계산되었다. 이 수치는 문명의 모든 단축키, 자동화, 그룹 이동을 최대한 사용했을 때의 값이다. 이렇게 하지 않았다면 아마도 나는 2 배정도 많은 이동을 했을 것이다. 문명 4 가 나온 시점에서 왜 내가 왜 문명 3 에 대해서 이야기를 하고 있는 것에 대해 당신은 의아하게 생각할 것이다. 나는 문명 4 를 지 않았다. 나는 좀더 많은 플레이를 할 수 있는 기대와 희망 속에 문명 3 확장팩을 기다리고 있었다. 마침내 문명 3 확장팩이 도착했고 전과 마찬가지로 많은 클릭이 있었지만 새로운 애니메이션이 되는 3D UI 가 있었다.

오해하지 말자. 문명 4 는 새로운 게임플레이 측면에서 중요하다.
파이락시스(Firaxis)회사는 새로운 버전을 내놓을 때 단순히 새로운 유닛, 샵화, 컷씬(cut scenes)을 넣은 회사보다 훨씬 나았다. 하지만 나는 그런 게임들에 싫증이 났거나 아직 충분히 문명 3 를 하지 않았기 때문에 문명 3 를 플레이하는 것을 멈추지 않았다. 게임을 플레이 하는데 너무나 오랜 시간이 걸렸기 때문에 나는 중단했다. 문명 게임은 보다 멋진 인터페이스가 있을 필요가 없다. 단지 좀더 효율적인 어떤 것이 필요했다.

오버클릭(Overclick)은 문명이라는 게임에 한정되지 않는다. 실시간 전략 게임들은 손목관절(carpal tunnel)질환을 유저에게 남겨준다. 이것이 내가 워크래프트(Warcraft)나 이와 유사한 온라인 게임들을 하지 않는 이유이다. 클릭 스킬로 보면 나는 전성기를 지났다. 전략은 오늘날 실시간 전략게임에서 당신보다 두 배 빠르게 클릭할 수 있는 14 살짜리 아이들에 대항해서 플레이를 하고 있을 때는 무의미하다.

RTS 사용자 인터페이스는 많은 현재 게임보다 좀더 유용한 유닛 자동화를 가졌던 토탈 어니힐레이션(*Total Annihilation)* (1997 년)이래로 개선이 없었다. 그 사이에, 컴퓨터가 제어하고 움직일 수 있는 오브젝트의 수는 증가했으며 계속해서 지수적으로 증가했다. 오래된 UI 모델은 전환점에 있지 않았다. 오래된 UI 모델은 부서져야 한다.

이 기사는 플레이어들이 보다 적은 클릭으로 자신이 원하는 것을 할수 있도록 UI 를 디자인하는 방법에 대한 것이다. 나는 UI 인간공학(사용 편의성) 또는 인지적 얼고나믹스(눈 긴장과 인간 메모리와 처리 여건과 같은 문제)로 다루지 않을 것이다. 활동적인 독자는 이것들을 UI 평가에 넣어야 한다고 하지만 이 단문 기사를 위해서 그것은 너무나 복잡하다.

7 의 법칙

전략 게임들은 전통적으로 대대장 같은 역할을 유저에게 부여한다. 그러나, 실제 대대장의 역할을 생각해 보면 그는 전장에서 지도를 보고 각 탱크들에게 명령을 내리지 않는다. 전투에서 대대장(battalion commander)은 직접 명령을 받는 소수의 사람들을 거느리고 있다.

미육군에서 분대장(squad leader)은 2 명의 소총팀 리더 (Fire Team Leader)의 도움으로 9 명의 병사들을 이끈다. 소대장(platoon leader)은 전형적으로 4 명의 분대장과 3-4 명의 다른 부관에게 명령한다. 중대장(company commander)은 4 명의 소대장과 부관에게 명령을 내린다. 중령(lieutenant colonel)은 4 명의 대대장과 부관에게 명령을 내린다. 즉, 지휘 계통의 체계 속에서 명령이 내려진다.

이것은 우연이 아니다. 지휘관이 효과적으로 제한된 수의 부하들을 관리하는 것은 군 조직 교리의 기본 원칙이다. 지휘관이 부하에게 직접 보고받는 수를 관리 범위(*span of control*)라고 한다. 제어할 수 있는 가장 효과적인 관리 범위는 극장이 관리하는 범위와 소대장이 관리하는 범위가 같다고 믿어진다. 19 세기 유럽 육군은 관리의 최대 범위를 7 로 정했고 그 수는 아직까지도 변하지 않고 있다.

오늘날 미 육군도 관리 범위수가 어림잡아 5 에서 7 까지 이다. FEMA 에 있는 관리자는 재해 구호 노력 동안에 7 명의 부하만을 볼 수 있다고 한다. 미국 백악관 보고서 104-631 에 따르면, 1997 년에 전체 미국 정부 관료주의의 관리 평균 범위는 7 이었다. 우연이 아니라 이 7 이란 숫자는 일반 사람이 동시에 품목을 기억할 수 있는 평균 수치이다. (밀러 1956 년). 이런 이유로 지역 번호 내의 전화 번호는 7 자리수가 된다.

마찬가지로 이 규칙을 전략 게임 디자인에 적용해야 한다. 플레이어가 7 개 이상의 엔티티들을 제어하는 플레이어는 그 중 어느 하나라도 효과적으로 관리할 수 없다. (그 결과 한 턴이 1 분의 시간이 걸리고 플레이어가 매 10 초마다 한번 이동한다면 그 플레이어는 아마도 집중을 못하고 전략 게임에 있어 즐거움의 원천인 전략적 깊이 있는 사고를 할 수 있는 기회를 가질 수 없게 된다. 초당 한번 클릭을 필요로 하는 게임은 복잡도에 상관없는 아케이드 게임이다.)

한 야전 사령관이 관리하는 7 명의 부하들은 전투에 실제로 가담하지 않지만 지휘자에게 정보만 제공하는 사람이 한두 명 포함된다. 이것은 다양한 정보 화면(문명에서 도시 화면과 기술 화면 같은)들이 7 개가 된다는 것을 의미한다.

이 규칙에는 곤란한 문제가 있다. 각 플레이어가 16 개의 체스 말들을 제어하는 체스에서는 16 개의 상대편 말에 대해 인식하고 있어야 한다. 이것이 체스가 초심자에게 좌절감을 주게 되는 이유가 된다. 체스는 7 의 규칙을 깨뜨렸는가?

글쎄, 맞을 수도 있고 아닐 수도 있다. 체스에서 전문적 기술을 습득하는 것은 여러분의 머리 속에 계속해서 체스 말들을 따라가는 것으로 달성되지 않는다. 관리할 수 있는 수인, 개념적인 수자 7 로 끌어내리기 위해서 보드 중앙에서 영향을 주는 세 개의 졸과 한 기사 뒤에 왕이 있는 우리에게 익숙한 구조인 폰 구조(Pawn Structure)로 보드 좌표를 개별적으로 분석하는 것으로 연관 지을 수 있다.

미군에 있어 관리 범위는 지휘관이 1 단계 아래(하급부하)로 명령하고 감독하며 두 단계로 아래로(부하의 부하)일어나는 모든 것을 추적하도록 만들어졌다. 체스 플레이어는 관리 단계가 두 단계까지 책임져야 한다. 즉, 먼저 이러한 체스 구조(예. 관제 센터 또는 수비적인 말 구조를 깨트리는 것)각각을 위한 세부목표를 선택하는 것이다. 둘째로 각 세부목표를 실현하기 위해 한번 움직일 것을 고르는 것과 단지 이러한 움직임들의 하나를 선택하는 것이다. 따라서, 모든 체스 졸들을 추적하지는 않지만 매 움직일 것은 선택해야 한다. 이것은 7 법칙을 위반하는 것이다. 만약 당신이 체스보다 더 빠른 것을 플레이하는 게임을 원하고 여전히 그것이 전략게임이라면 7 의 규칙을 준수해야만 한다.

한번에 두 가지를 동시에 할 수 없다.

게임 디자이너는 플레이어가 모든 유닛을 관리할 수 있는 게임이 만들어 짐으로써 두 개의 월드 중에 가장 좋은 것을 가질 수 있다고 생각하지만 운 나쁘게도 그렇지 않다. 경제학에 그레샴 법칙(Gresham's Law)이 있다. 이것은 악화(惡貨)는 양화(良貨)를 구축(驅逐)한다는 것으로 한 나라에서 실제로 소재 가치가 있는 금화와 가치가 없는 지폐를 사용하려고 할 때 모든 사람이 왜 지폐만을 사용하는 것에 대해서 왜 지폐가 시장에서 돈을 몰아내는지에 대해서 그래샴은 설명했다.

게임에서, 나쁜 플레이어는 좋은 플레이어를 몰아낸다. 롤플레잉 게임에서 부와 힘을 축적하는 것을 강조하는 나쁜 롤플레이어는 보다 빠르게 앞으로 나아가기 때문에 결과적으로 좋은 플레이어를 몰아내게 된다. 게임에서 개개인의 유닛들을 관리할 수 있도록 허용한다면 플레어보다 유닛들을 더 잘 관리하는 컴퓨터가 나오려면 아직도 멀었기 때문에 1 초에 세 번 클릭을 할 수 있는 아드레랄린이 가득한 14 살짜리 애들은 자기가 계획한 것을 완성하기 위해 컴퓨터에게 맡기는 생각을 깊게 하는 플레이어들을 박살낼 것이다.

빠르게 누르고 세부적인 것까지 제어하기를 좋아하는 한 플레이어가 있으며 이 플레이어는 게임 산업 잡지, 게임 광고, 게임 유통 채널에 관심이 많은 14 살 남성일수도 있다. 인구 통계적으로 객관적 견지를 갖는 것은 언제나 위험하다. 국경에서 게임이 판매될 때까지 게임이 주류가 되지 않을 것이라고 믿는다. 제작자들은 이러한 게임 전체 시장을 잃고 싶어하지 않는다. 그래서 옵션으로 이용할 수 있는 개인 유닛 관리를 가지게 하는 것은 좋을 수 있다.

그러나, 내가 보기에는 오늘 날의 게임 시장 중 14 살 남성을 상대로 한 시장보다 더 큰 손목관절증상을 좋아하지 않는 플레이어들의 시장에서의 서비스는 아직 부족하다.

만약 게임이 개인 유닛을 관리할 수 있도록 허용 한다면 그것은 별도의 게임 옵션으로 하고 플레이어가 개인 유닛 컨트롤을 할 수 없는 멀티 플레이어 게임을 설정 할 수 있어야 한다.

지금쯤, 당신은 아마 가마스트라(Gamasutra) 편집자의 분별력과 나의 온건함에 대해서 문제를 제시할 것이다. 내가 전략 게임이 단지 플레이어가 7 개 단위를 건설한다고 것을 허용해야 한다고 말하고 있는가? 천만에. 나는 플레이어가 그들을 전부 직접적으로 통제하면 안된다고 말하고 있는 것이다. 우리는 관리 수준 개입을 개념화할 필요가 있다. 이것을 하는 것은 어려운 것이 아니지만 객체 지향 프로그램에 대한 흔한 오해에 의해서 방해를 받는다.

오브젝트들이 오브젝트가 될 필요는 없다.

스몰토크 유저들은 오브젝트들을 “객체”라고 말하고 또 다르게 오브젝트들은 메소들을“동사”라고 불렀다. 그 후로 많은 객체 지향 프로그래머들은 그 말을 “명사”와 같은 것으로 “객체”라고 해석한다. 나는 1980 년에 코드에서 게임에서 실제적인 대상이 OO 객체로 되어 있지 않으면 게임이 객체 지향이 아니라고 주장하는 어드벤쳐 프로그래머와 논쟁을 했었다.

내가 게임상에서 실제 오브젝트가 코드에서 동사를 가지도록 코드를 구조화 해야 한다고 제안했고 이것은 게임상에 일관된 물리를 강화시켰다. “오브젝트는 객체이며 동사는 동사이다” 이것을 하기 위해 게임상에 실제 오브젝로 게임 코드, 유저 인터페이스를 체계화 했다.

이렇게 할 필요는 없었다. 객체 지향관점에서 오브젝트는 당신이 선택하는 어떤 것이던지 추상적인 것이 될 수 있다. 문명 3 의 경우에, 객체는 군사 행동 또는 엔지니어링 프로젝트가 될 수 있다. 그림 1 을 살펴보자. 이 그림에서 나의 문명은 최근까지 철도를 위한 기술을 개발해 왔다. 나는 내 문명의 남쪽 끝과 북쪽 끝을 연결하는 선로를 건설하고 있었다.

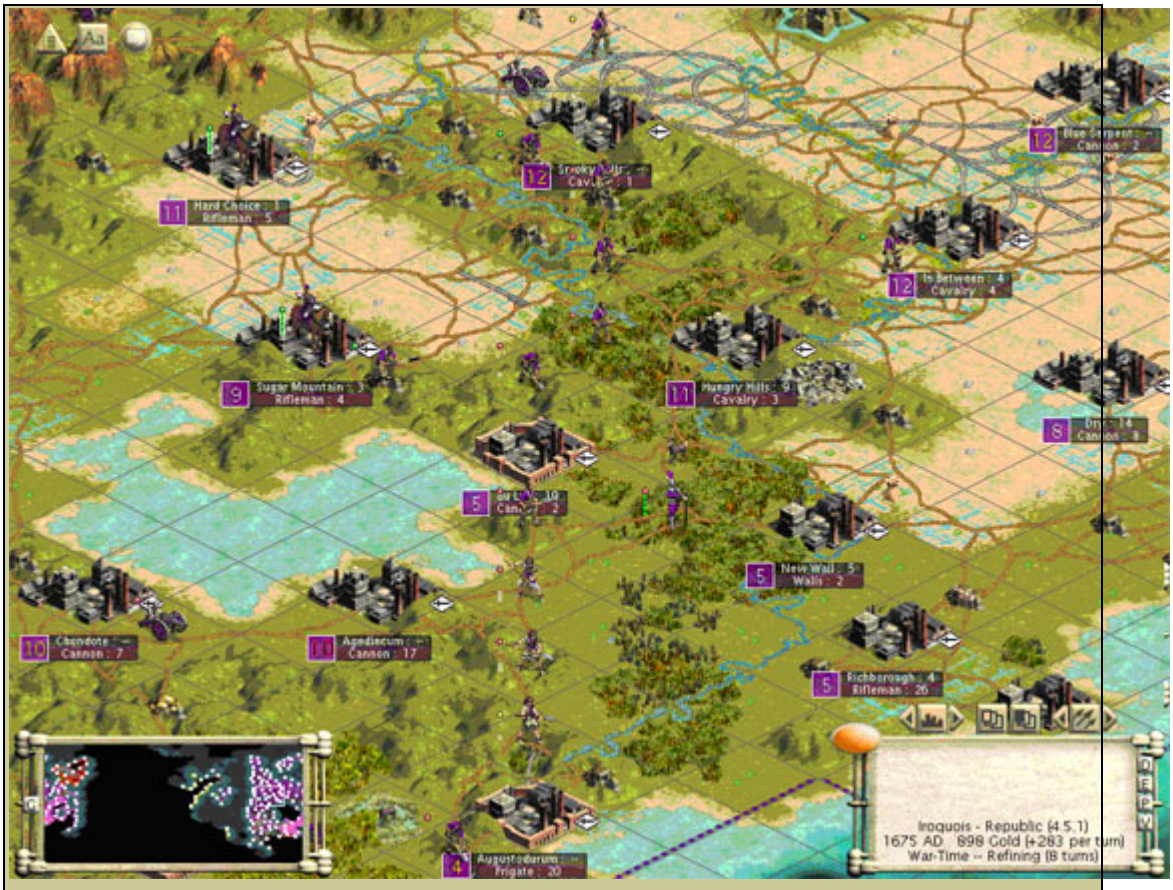


Figure 1

*이것을 하기 위해 문명 3*의 유닛 중심 인터페이스로 개인 작업자 유닛을 선택하고 그들을 선로의 개별 섹션에 할당하는 것이 필요했다. 한 유닛을 북쪽에서 남쪽을 향하는 한 유닛으로 시작하고 다른 유닛은 남쪽에서 북쪽으로 향한다면 작업자 유닛들은 경로를 계산하고 작업의 시초를 따를 것이고 다른 유닛에 의하여 그사이에 되어 진 작업을 책임지기 위하여 경로를 맞추지 않는다. 다수의 겹쳐지지 않은 평행한 선로로 이르게 함으로써 로마군대가 당신을 침략하기 전에 군대들이 국경에 도달할 수 없다.

나는 침략 받기 전에 약 백 명의 작업자들을 선로를 건설하는 작업에 배치할 필요가 있었다. 각 작업자들에 대해서 유닛을 클릭해서 매번 초점 안으로 작업자 유닛을 가져와야 했다. 이것은 움직임을 시작하기 위해‘g’키를 누르고 선로의 시작 지점을 위해

스크롤 했고 다시 클릭을 했다. 뒤이어 철도를 건설하기 위해 “ctrl-r”을 누르고 철로의 유닛 부분의 끝을 스크롤 하기 위해 다시 클릭했다. 한 유닛당 세 번의 마우스 이동,세 번의 키 누름,세 번의 마우스 클릭을 한다. 나는 정상시간에서 약 30 분정도에서 이것을 할 수 있었음에도 불구하고 세 개의 그룹내의 작업자들에게 이것을 맡겼다. 그래서, 철도를 건설하기 위해 600 번의 클릭, 키 누름, 스크롤을 했다.

철도를 건설한다고 하고 철도 시작지점에 마우스를 클릭하고 끝 지점에 마우스를 클릭한다고 상상해보자. 철도 섹션에서 작업하는 작업자들을 이용할 수 있게 됨에 따라 컴퓨터는 작업자들을 지휘할 수 있게 된다. 전체 철도는 내가 한 작업자에게 지시한 것과 같은 양으로 건설될 수 있다.

철도는 로마에 방어하기 위해 내 군대를 국경선으로 이동하기 위해서 필요했다. 또한 건설된 각 새로운 유닛은 방어를 위해 국경선을 따라 어떤 지점으로 수송되어야만 했다. 간단하게 국경선, 도시, 방어지점을 정의해 주면 컴퓨터가 군대를 직접 지휘해서 나의 손목에 고통이 덜 갈수 있을 것이다. 그러나, 이것을 깔끔하게 구현하기 위해서 프로그래머는 첫 번째 클래스 객체로 철도와 국경선을 생각해야 한다.

온라인 대 오프라인

지휘관이 7 명의 부하들에게만 명령함으로써 얻을 수 있는 부분은 준비보다 더 중요한 것이다. 지휘관은 어떤 행위에 대한 시나리오를 미리 그려보고 한 시나리오에서 다른 시나리오로 변경하는 명령을 내림으로써 전투 상황에서의 재빠르고 극적인 변화를 할수 있게 한다.

지휘관은 교전 중에 그의 생각을 부하에게 설명하기 위해서 사용할 수 있는 분대 수준에서 전술의 표준 라이브러리를 가지고 있다. 부하들은 그들이 다양한 적군의 변화와 간섭 없이 비 전투원의 다양한 변화에 대처하는데 도움을 주는 교전 규칙을 가지고 있다.

지휘관은 이러한 규칙을 전투가 들어가기 전에 추가할 수 있다. 지휘관은 많은 연습을 했기 때문에 부하들에게 무엇이 옳고 그른 것인지를 말한다. 그리고 그의 상사는 그에게 무엇이 옳고 그른지를 말해준다. 이것은 전투에서 필요한 직접적인 관리의 양을 줄여준다.

당신이 룰-기반 시스템이나 유한-상태 오토마타와 같은 획일화된 형식을 사용해서 게임 AI 를 디자인 한다면, 당신의 플레어들이 유닛에 지시하는 다른 방법으로 유닛을 제어할 것이다. 반 자동 유닛을 위한 교전 규칙을 생성하는 것은 JASF 나 ONESAF 같은 실제 군 전쟁 게임 시뮬레이터에서 반드시 고려되는 유닛이다. 실제로, SAF 는 “반자동형 포스 (semi-automated forces)”을 나타내는 것으로 오퍼레이터가 가능한 적게 개입될 수 있도록 복잡한 임무와 교전 규칙을 가지고 있는 유닛이다. 반면에, 여전히 각각의 반대 유닛을 관리하려고 하는 인간에 대해서는 실제 훈련 환경을 제공해야 한다.

퀘이크(*Quak)*와 유사한 몇몇 게임들은 적군을 프로그램 하기 위해 AI에 접근할 수 있도록 하고 있다. 로보워(Robotwar)로부터 계승받은 다른 것들은 완전히 프로그램 되어야만 하는 플레이어 유닛을 제공하고 게임 플레이 중에는 그것은 관리될 수 없다.

반자동 힘에 대한 인터페이스를 위해 어떠한 것도 제공하지 않는다. 교전수칙을 제공하기 위해 오프라인에서 사용하는 것과 온라인에서 사용하는 두 개의 유저 인터페이스를 제공하면 개개의 유닛을 다룰 수 있는 스트레스를 줄일 수 있다.

유저 인터페이스 프로파일링

유저 인터페이스가 효과가 없는 지역을 탐지하기 위해 유저 인터페이스 프로파일러와 함께 게임을 테스트할 수 있다. UI 프로 파일러는 코드 프로 파일러와 같지만 CPU 사이클을 기록하는 대신에 유저 인터페이스 이벤트를 기록한다. 코드 각 부분에서 플레이어가 얼마나 많은 누르기, 키 누름, 마우스 움직임을 하는지를 정확히 보여준다. 유저 인터페이스 프로파일러는 유저가 취한 행동의 시퀀스를 그래프로 보여주는 것 같은 더 많은 정보를 보여줄수 있다 (Ivory & Hearst 2001 년).

I/O 이벤트나 함수 호출을 계산하기 위해서 IDE 프로파일러를 사용할 수도 있지만 이것은 일반적으로 플레이어가 대부분의 시간을 무엇을 하면서 보내는지는 알려주지 않을 것이다. 만약 여러분의 코드에 UI 이벤트를 기록하기 위해서 함수 호출을 코드화 함으로서 UI 프로파일을 짰다면 좀더 많은 정보를 얻을 수 있을 것이다.

*만약 Civ III*의 개발자가 UI 프로 파일러를 사용했다면 그들은 게임의 협상 부분에 사용되는 비정상적은 클릭 회수와 키 누름을 알아냈을 것이다. 이것은 유저가 그것을 발견하기 위해서 이진탐색을 하도록 만드는 것보다 무역에서 제공되는 수긍할만한 금화수를 추천하기 위해서 게임에 필요할 뿐만 아니라 도시 선택 메뉴에서 스크롤바를 빠트리는 게임에서의 버그를 드러내게 한다. 이 2 개의 업무는 천 번 이상의 UI이벤트가 일어났다.

다양한 UI를 비교하기 위해서, 점수를 기록하는 방법이 필요하다. 나의 *Civ III* UI 셈으로 돌아가보면 UI 액션의 각 타입에 가치를 1 점으로 할당해서 총 UI 점수를 구했다. 점수를 할당하는 방식은 측정하는 것이 무엇인가에 따라 달라진다. 아케이드 게임의 경우는 속도가 1 차 범주가 될 것이며 키를 누르는 것 만큼 빠르게 마우스 클릭을 셈하게 된다.

싱글게임에 많은 시간이 걸리는 문명 같은 게임에서 여러분은 플레이어가 지치게 되는 것에 더 신경을 써야 한다. 마우스 버튼을 클릭하는 것은 키를 누르는 것보다 더 많은 힘을 필요로 하며 유저는 매번 같은 손가락을 사용하게 된다. 따라서 이것은 손목에 심한 부하를 주게 된다.(타이핑할 때의 사람들의 고통은 마우스 사용에서 비롯된다.)

그래서 나는 마우스 움직임과 키 누름을 1 UI 점으로 계산 했으며 마우스 클릭은 3 UI 점으로 했고 휠 스크롤은 6 UI 점으로 했으며 마우스 패닝(맵을 스크롤하는것)은 9 UI 점으로 해서 계산 했다. 이 값은 턴당 총점수가 2208 UI 가 되었다. 다른 UI 들도 점수에 의해 비교될 수 있다.

UI 프로파일러는 이미 만들어진 UI 를 다듬거나 평가하기 위해서 사용할 수 있다. 약간의 수학으로 여러분은 코드를 작성하기 전에 UI 를 평가할 수 있다. 나는 간단하게 다음부분에 이것을 다룰 것이다.

작업계산

유저 인터페이스를 사용해서 하나의 움직임을 지정하기 위해서 플레이어가 해야 하는 작업 W 를 계산할 수 있다.

게임플레이 영역크기와 플레이어 유닛수 같은 게임 변수로 W 를 측정한다면 당신은 심지어 W 를 추정만 했을지라도 다양한 유저인터페이스를 비교할 수 있을 것이다.

예를 들어, 가상 점토 조소(sculpting virtual clay)용 UI를 생각해보자. 당신은 *n′n′n 복셀 배열 크기를 가지며* 각각 복셀은 on/off를 할 수 있다. 우리는 네 가지 가능한 유저 인터페이스를 생각해 볼 수 있다.

첫 번째 인터페이스에서 유저는 켜기를 원하는 각 복셀의 좌표 내에서 친다. 1 부터 n까지 수를 지정하기 위해서 취해야 하는 키 누름의 수는 $\log_2(n)$에 비례해서 각 복셀로 나타내기 위해 작업은 $3\log_2(n)$ 비례하게 된다. (이제부터 나는 비례한다 라고 하는 말을 W=O(f(x))의 의미를 가지는 것으로 W=f(x)라고 쓰겠다.) 아티스트들이 효율적으로 한 표만을 나타낸다고 가정하면 조소 내부 전체를 채우지 않을 것이다. 따라서, 전체 작업은 $W=3n^2\log(n)$이 될 것이다. (표면의 크기는 n^2에 비례한다.)

두 번째 유저 인터페이스에서, 유저는 3 차원 3Dconnexion 사의 Spaceball 같은 마우스로 3 차원 공간상에 한 점을 선택하고 마우스를 클릭해서 on/off 상태를 토글 시킨다.

3D 공간에서 특정 한 점으로 가기 위해 필요한 작업은 3 번의 움직임의 조합이 된다. 각 움직임은 축이 된다. 여기는 축은 1 부터 n까지 범위를 가지게 된다. 각 움직임은 $n/2$ 평균을 가지면서 1 부터 n의 범위를 가지는 한 축에서의 움직임이 된다.

이것은 마치 조각품을 생성하기 위해 작업 $W=n^5/8$ 이 되는 것으로 보인다. 그러나, 한 복셀을 켠 후에 유저가 이웃하는 26 개의 복셀 중에 하나를 움직인다고 생각해보자. 우리는 이것이 $\log_2(26)$인 26 중에서 1 개의 선택을 지정하기 위해 필요한 정보에 작업이 비례 한다고 말할 것이다. 우리는 $\log_2(3^3=27)$로 그것을 근사화 할 것이다. 왜냐하면, W를 위해 이것과 전 값에서의 상수 3 은 3 차원 조각품에서 온 것이기 때문이다. 그때 전체 작업은 $W=3n^2\log_2(3)$ 이 된다. 이 인터페이스는 첫 번째 것보다 더 개선된 것처럼 보인다.

세 번째 인터페이스에서 유저는 원하는 조각품으로 같은 볼륨을 가진 점토 구체부터 시작해서 유저는 2D마우스를 점토 표면 주위로 움직이고 커서 아래지점(표면과 수직지점)에 복셀을 누르기 위해 왼쪽 버튼을 선택하고 위로 나오게 위해서 오른쪽 버튼을 이용할 것이다. 점토가 마지막 클릭에서 같은 방향으로 움직여질 때 푸쉬하거나 풀이 2 배가 되고 반대 방향으로 움직일 때는 푸쉬/풀이 반이 되는 복셀의 수를 나타내는 푸쉬/풀 인터페이스를 사용할 것이다. 그래서 적절한 위치가 이진 탐색으로 찾아지게 되며 시간은 $\log_2(n)$에 비례하기 된다. 다시 유저가 표면 위에 각 점에 작업할 필요가 있다고 생각해보자. 다음 복셀로 이동하기 위해 필요한 작업은 2 차원 움직임이기 때문에 $2\log_2(3)$이 된다. 그때 전체 작업은 $W = n^2 \cdot 2\log_2(3) \cdot \log_2(n) = 2n^2\log_2(n)\log_2(3)$ 이 된다. 이것은 가성 점토를 푸쉬와 풀을 할 때 취하는 클릭회수 때문에 표면에 움직임을 구속함에도 불구하고 앞에 있는 UI보다 더 안 좋다.

네 번째 인터페이스에서 유저는 2D 마우스를 가지고 표면 주위로 움직이며 앞에서처럼 들쭉날쭉한 점들을 푸쉬하고 풀할 것이다.하지만 점토 표면은 인장을 가지고 있어서 들쭉날쭉한 한 점을 푸쉬하는 것은 이웃에 있는 복셀 모두를 끌어당기게 될 것이다. 그 결과 조종점과 스플라인을 사용해서 곡선을 정의하는 것과 같은 방법으로 표면이 조각될 수 있다. 그때, $W = 2c^2\log_2(n)\log_2(3)$이 되며 이때, c는 조종점의 수이다. W는 복셀수가 아닌 조각품의 불규칙 함수가 된다. 고해상도 모델링에 대해서는 $n>>c$이며 $W = O(\log_2(n))$가 된다. 이것은 대단히 뛰어난 유저 인터페이스이다.

인지공학과 W 를 결합하기 위해 플레이어가 클릭 가능한 아이콘, 기타 등등 각 키보드 단축키의 의미를 기억하는데 필요한 메모리의 양을 세고 보여지는 정보를 적절하게 사용할 수 있는 정보(이것은 어려운 부분이다.)로 변환하기 위해서 작업 측정값을 섞을 수 있다. 메모리 추출 시간을 위해 하나의 단어를 추가할 수 있다. 정보검색시간은 주어진 메모리의 다른 형태에 의하여 추정된다. (*(앤더슨 1974 년),* 옵션들의 한 리스트중의 한가지를 기억하기 위하여 (다시 말해, 한 유닛에 대한 가능한 명령어들) 그 시간은 K+an 이다. 여기에서 n 옵션의 수이고 k 와 a 는 상수이다. 만약 당신이 비 인식의 항들로서 비교하기 위하여 인식의 항들을 헤아릴 수 없다면 인식의 항들은 아마도 각각 더해진다.

이론적인 인터페이스 효율성

일반적으로 어려움에도 불구하고 이론적으로 최적인것과 비교해서 효율적인 유저 인터페이스를 계산할 수 있다. 정보 이론은 현재 상황, 역사, 심볼과 같은 지식이 주어지면 일련의 수나 다른 심볼들에서 보여지는 많은 정보들을 계산하는 방법을 담고 있다. 정보이론에 따르면 한 움직임에서 보여지는 정보 I 를 추정할 수 있다. 가능한 좋은 UI 는 $O(W) = O(I)$인 의미로 W 와 I 를 계산 복잡성의 동일 순서로 같게 놓는 것이다..

I/W 는 인터페이스 효율성을 측정한다. 이것은 최대 $O(1)$이 될 수 있다. I 로 표현되는데 있어 매 변수는 W 와 같거나 보다 큰 지수가 된다. 그래서, W/I 로 생각하는 것이 쉽다. 이 값은 플레이어에게 나쁘게 측정되는 값이다. I 를 추정하는 것은 W 를 추정하는 것 보다 더 어렵다. 그래서, 종종 I 에 대한 상한 경계 값 만 얻을 수 있다. 의미 있는 W/I 를 위해서 이 값이 하한경계 값을 가져야만 한다.

많은 유닛을 가지는 게임에서 당신이 찾고 있는 유닛 기반 유저 인터페이스에 대한 것이다. 상한 경계값 I 보다 하한 경계값 W 가 유닛수가 더 빠르게 증가한다. 이것은 대부분의 시간에 다수의 유닛들이 같은 일을 하고 플레이어의 유닛의 적은 수가 하는 것을 아는 것은 숙력 된 플레이어가 좋은 정확도를 가지고 그들의 유닛의 나머지가 하는 일을 예측하도록 가능하게 하기 때문이다. 이것은 정확히 점토 조각품을 위한 유저 인터페이스와 유사하다. 조각품 표면 위의 대부분의 점들은 그것들 근처에 점들이 서로 같은 표면 탄젠트를 가진다. 매 유닛 마다 움직이는 것이 필요한 유저 인터페이스는 매 복셀을 움직이도록 만드는 조각품 인터페이스와 유사하다. 정보이론의 사용은 단순히 게임 중에 유닛 사이의 관계를 가시화하는 방법을 가지지 않더라도 I 를 계산할 수 있도록 해주며 필요한 UI 실제 차원을 알 수 있게 해준다.

일반적으로 W 와 I 를 계산 하는 방법에 대해서 설명한 부분은 다른 문헌에서도 찾을 수 있다. 여러분은 확률과 정보 이론에 관한 책으로부터 이 부분을 찾을 수 있다. 좋은

참고문헌은 일리노이 대학 출판사에서 1998 년에 증쇄된 *The Mathematical Theory of Communication* 이다. 어려운 경우에 조합이론에 의해 I 를 추정할 수 있었고 플레이테스팅(playtesting) 동안에 모인 통계의 조합을 추정할 수 있었다.

요약

컴퓨터는 플레이어가 합리적으로 제어할 수 있는 것보다 더 많은 유닛들을 움직일 수 있으며 그 수는 계속해서 지수로 증가할 것이다. 이것은 플레이어를 재미보다 훨씬 더 좌절하게 만들 수 있을 것이다. 좋은 유저 인터페이스 디자인에서 플레이어는 7 개선에서 게임엔티티를 관리해야 한다. 이것을 하기 위해서 UI 는 화면 유닛보다 더 많은 추상적인 것으로 플레이어를 컨트롤하도록 해야 한다. 객체 지향 개발자는 코드 오브젝트를 화면상에 보이는 유닛의 추상화로 생각할 수 잇다. UI 는 또한 온라인에서 감독하는 양을 줄이기 위해서 오프라인에서의 행동을 지정하는 기회를 제공해야 한다.

게임 개발자는 유저 인터페이스 프로파일링 도구를 사용해서 유저 인터페이스를 평가할수 있으며 다른 인터페이스에 포함된 작업을 계산할 수 있다. 이것들은 더 향상될 여지가 있는지를 알기 위해 이론적인 효율성을 추정할 수도 있다. 좋은 게임 디자인의 궁극적인 목적은 게임의 FPS(초당 재미)를 증가시키는 것이다. 이것을 하기 위해서 동일 행위를 몇 초 정도 더 적게 만드는 것이다. 이것은 유저 인터페이스를 향상시키는 가장 쉬운 방법이다.

추가 문헌

- John R. Anderson (1974). Retrieval of prepositional information from long-term memory. *Cognitive Psychology* 6: 451-474.
- Brig. General Huba Wass de Czege & Major Jacob Biever (1998). Optimizing future battle command technologies. *Military Review*, Mar/Apr.
- Melody Ivory & Marti Hearst (2001). The state of the art in automating usability evaluation of user interfaces. *ACM Computing Surveys* 33(4): 470-516.
- Kenneth Meier & John Bohte (2000). Ode to Luther Gulick: Span of control and organizational performance. *Administration & Society*, 32(2): 115-137.
- G. A. Miller (1956). The magic number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63.
- Claude Shannon & Warren Weaver (1949). *The Mathematical Theory of Communication*. Reprinted in 1998 by the University of Illinois Press.