**Terran War 졸업연구 최종보고서**

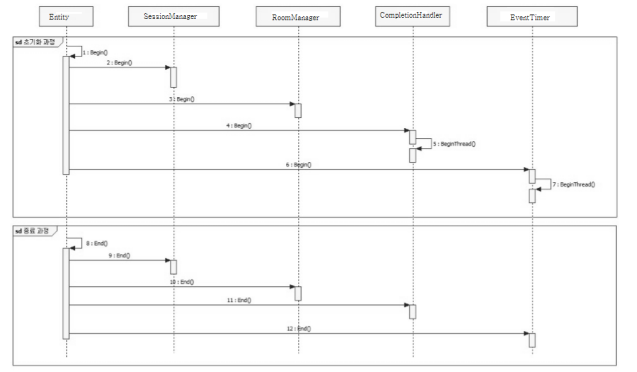
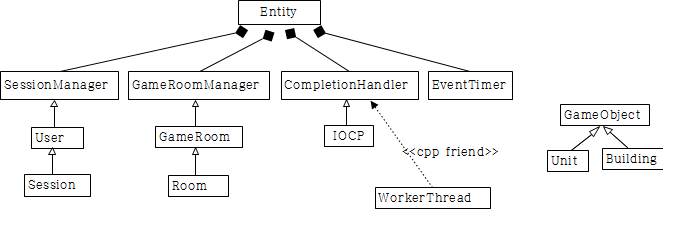
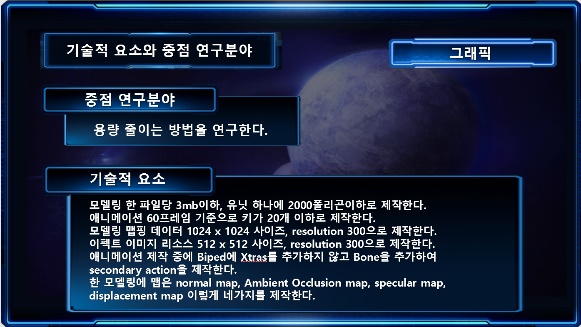
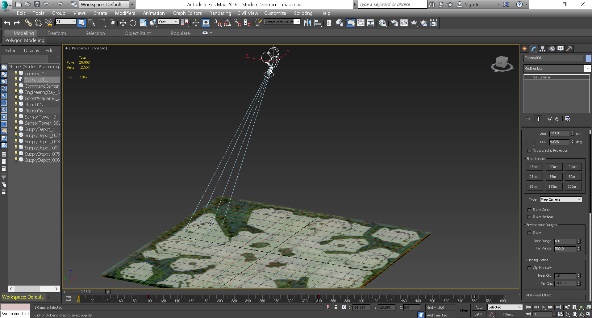
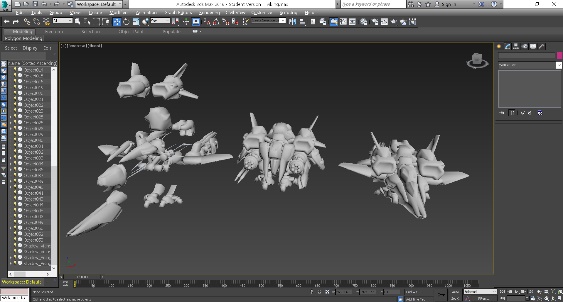
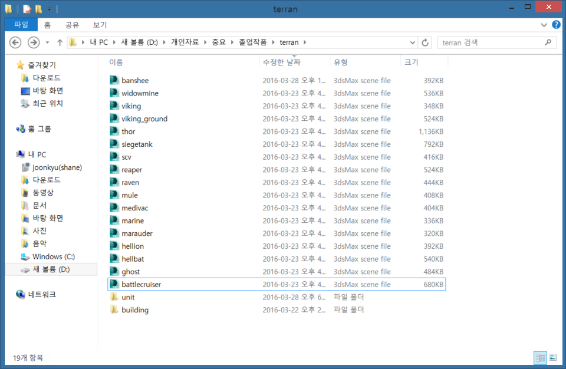
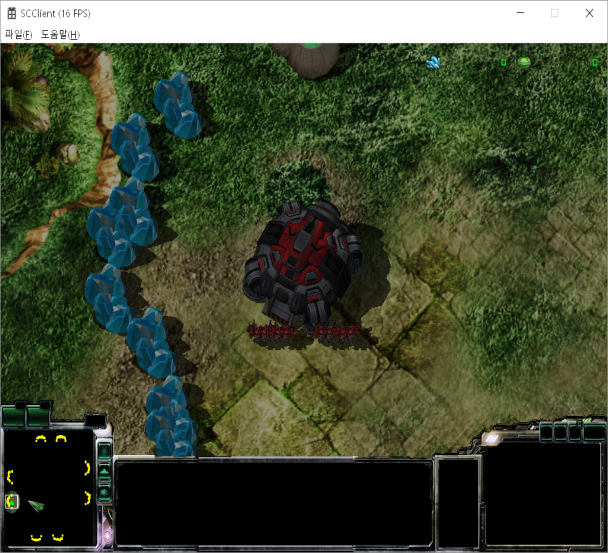
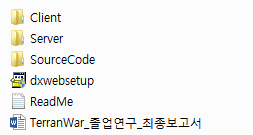
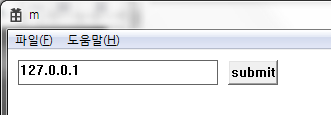
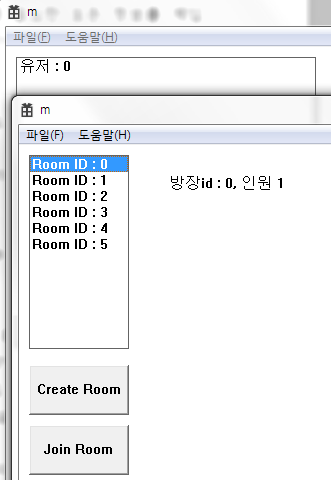
이준규, 전병걸, 한청근

한국산업기술대학교 게임공학부

{mercer11, qudrjf , [han15369}@kpu.ac.kr](mailto:han15369%7d@kpu.ac.kr)

**요 약**

학과내에서 졸업연구 주제로써 잘 선택되지 않는 RTS(Real Time Strategy)장르에 대하여 도전을 해보고 싶었다. 대표적으로 높은 평가를 받고있는 ‘블리자드’의 '스타크래프트 시리즈를 연구하는 것이 충분히 가치 있다 판단하여 본 보고서에서는 연구방향을 스타크래프트를 개발 하는 것으로 정했다.  
 학과에서 배운 내용을 기반으로 원작의 구현된 기능들을 살펴보고 이를 게임서버 및 클라이언트로 구현하였으며 개발에 필요한 리소스를 직접 제작하였다.  
 개발을 마치고 원작과 근접한 표현 및 기능을 얻어냈다.

1. **서 론**여태까지의 교내 졸업연구의 주제로 RTS장르를 잘 선택하지 않다는 것을 알게 되었다. 이유를 알아보니 구현 난이도가 높고, 만들어진 틀이 존재하지 않아서 개발환경이 열악하다고 판단하였다. 이러한 환경에도 불구하고 RTS장르의 게임을 개발하여 자기계발을 하고 어떠한 어려운 과제를 수행하더라도 극복하기 위해 RTS장르 게임 개발을 졸업연구의 주제로 삼게 되었다.  
   RTS게임을 분석해보기 위해 대표적이고 높은 평가를 받고있는 스타크래프트 시리즈를 졸업연구 대상으로 정하였다.  
   개발환경은 PC플랫폼으로 설정하였으며 3D RTS개발을 목표로 하였다.  
   3D RTS개발에 맞는 스타크래프트 시리즈인 스타크래프트2 : 자유의 날개를 모티브로 초보자들이 스타크래프트의 게임플레이를 쉽게 익힐 수 있고 실제 튜토리얼로 쓰이는 종족인 테란을 구현하기로 결정하였다.  
   게임 모드 중 ‘사용자 지정 1:1 대전’ 으로 전략적인 플레이를 통하여 상대방의 진영을 파괴하여야 한다.  
   본 보고서의 구성은 다음과 같다.  
   2장에서는 개발내용을 클라이언트, 서버, 그래픽 순으로 개발내용을 기술한다.  
   3장에서는 개발내용을 토대로 결과를 도출하여서 비교를 하여 분석하고 향후 계획을 밝힌다.  
     
     
   **2. 개발 내용  
   2.1 서버**기존 스타크래프트 서버는 슈퍼클라이언트 구조를 갖추고 있다. 하지만 이는 보안에 취약하고 핫 스팟에서의 과부하를 견뎌낼 수 없다. 때문에 MMO(Massively Multiplayer Online)게임서버에 적절한 CS(Client/Server)구조로 개발하기로 하였다.2.2.1 서버 모듈  
   [그림1]는 서버의 시퀀스 다이어그램을 나타낸다. Entity클래스는 동시에 많은 클라이언트들의 요청을 처리하기 위해 Listen소켓을 생성한 뒤, 접속 요청을 기다린다. SessionManager 와 RoomManager클래스의 초기화가 끝나면 CompletionHandler클래스는 IOCP객체를 만든다. 그리고 프로세서 개수만갯 Worker Thread를 생성한다. 이때부터 Worker Thread는 IOCP에 들어오는 정보를 기다린다. 마지막으로 EventTimer 클래스는 타이머 Thread를 만들고 그 스레드에서 우선순위큐를 검사하며 작업을 기다린다. 또한 서버의 종료를 확인할 수 있는 Event객체를 생성한다. 모든 과정이 성공하면 서버의 초기화 과정은 끝난다.  
   서버 종료 과정은 우선 EventTimer클래스가 서버의 종료를 알리는 Event객체를 꺼버리는 것으로 시작된다. 그 뒤 타이머 Thread를 종료시킴으로써 큐에 있던 작업들을 비운 뒤 타이머 Thread를 소멸시킨다. CompletionHandler 클래스는 Worker Thread를 먼저 소멸시킴으로써 클라이언트들로부터의 요청을 받지 않는다. 그 후 IOCP객체를 소멸한다. RoomManager와 SessionManger클래스는 할당되어 있는 메모리를 모두 해제한 뒤 Entity클래스는 Listen소켓을 소멸시키고 서버는 안전하게 종료된다.  
   [그림 1 ] 서버의 시퀀스다이어그램  
   [그림 2]에서는 게임 서버의 클래스 다이어그램을 요약해서 나타낸 것이다  
   다음은 서버에서 중점적으로 연구한 충돌, 이동 및 공격의 알고리즘에 대해 기술한다. [그림2 ] 서버의 클래스다이어그램  
   [그림3]은 충돌과 이동을 위해 맵을 흑백으로 저장한 png파일이다. 검정색으로 칠한 부분은 장애물로 이동 불가 지역이며 충돌처리 된다. 각각의 점들은 node로 구성된다. 총 42,025개의 node와 333,744개의 edge로 구성된다. GameRoom객체는 이러한 그래프정보를 게임이 시작하면 메모리를 할당한 뒤 정보를 png파일에서 읽어 채운다. 이러한 초기화 과정은 클라이언트 프로그램의 초기화 과정보다 빠르기 때문에 걸리는 시간은 무시하였다. 하지만 개선의 여지는 남아있다. 장애물에 포함되는 노드는 실제 사용되지 않는 노드이다. 그럼에도 불구하고 번호를 매김으로써 이동과 충돌뿐만 아니라 모든 부분에서 성능이 향상되었다. 그 이유는 좌측 상단부터 우측하단까지 번호를 매김으로써 한 지점에서 가장 가까운 노드를 찾는 시간이 상수가 되었기 때문이다. 첫 번째로 충돌알고리즘은 충돌되었을 때 검사하는 것이 아니라 충돌 가능성을 없애는 방식으로 구현하였다. 구체적으로 자신이 점유한 노드를 기준으로 사방으로 상대방이 있는지를 검사한 후 움직인다. 상대방이 점유하고 있다면 그 방향으로 움직이지 않고 다시 길을 찾는다. 이 방법은 여러 스레드가 동시에 점유된 노드들의 리스트를 검사할 때 문제가 될 수 있다. 따라서 점유된 노드들을 관리하는 자료구조는 뮤텍스 락으로 동기화 시켰다. 이렇게 기본적인 충돌과 이동을 처리하였다. 하지만 문제점이 있다. IOCP모델을 사용하기 때문에 Worker 스레드가 여러 개 있다. 때문에 클라이언트가 오른쪽 버튼을 여러 번 누르면 서버에게 길을 찾아 달라는 요청이 그만큼 들어오게 된다. 서버에서 이 요청들을 모두 받아서 처리한다면 과부하가 걸린다. 때문에 bool형 변수를 하나 만들어 타이머 큐에서 가장 최근에 실행되어야 하는 길 찾기를 무효화 시켰다. 그 다음에 타이머 큐에서 나오는 작업은 이전에 무효화 시키며 bool형 변수를 다시 바꿔줬기 때문에 실행되므로 단 한번만 길 찾기를 진행하도록 동기화 하였다. 이는 스타크래프트의 특성인 스스로 이동하는 경우가 많아 클라이언트의 길 찾기 요청 뿐만 아니라 스스로 적을 쫓아갈 경우나 도망갈 경우 등 모두 동기화 시킬 수 있으므로 이러한 동기화 방식은 유용했다. 공격 알고리즘은 클라이언트에서 1차적으로 사정거리를 통해 검사한 후 서버에게 공격을 요청한다. 서버는 요청을 받은 후 사정거리를 다시 확인한 뒤 공격 애니메이션을 진행하라는 패킷을 보낸다. 애니메이션의 일정 시간이 지나면 실제 체력이 깎고 공격에 성공했다는 패킷을 보냄으로써 공격알고리즘은 끝난다.  
   ****[그림3] 서버에서 사용한 충돌체크 맵  
     
   **2.2 클라이언트**수많은 우수 개발자들이 여러 해에 걸쳐 만들어진 것이 스타크래프트 시리즈인 만큼, 본인이 어느 부분까지 개발을 할 것인지를 정하는 것은 중요한 문제가 되었다. 더불어 스타크래프트 내의 세부적인 게임 기능들(예시로 랠리 포인트가 있다.)과 같이 서버 프로그래머와 구현 방식에 대해 논의해야 하는 부분들도 있었기에, 한 게임이 가능하도록 하는 최소한의 기능을 개발범위로 산정하였다. 이에 해당하여 개발을 마친 내용을 실제로 개발한 순서에 따라 하나하나 짚어가며 설명하도록 하겠다.프레임워크 구조를 만들 때에 크게 3가지의 항목을 고려하여 진행해야 했다. 첫째로는 렌더링 구조, 둘째로는 게임 기능구조, 마지막으로 패킷 통신구조이다. 렌더링 구조에서 신경 써야 할 점은 게임 내 많은 객체들이 그려지므로 한 번에 다량의 객체를 그려내는 인스턴싱 렌더링을 가능하게 하는 것과, 프레임레이트 확보를 위한 렌더링 최적화였다. 그래픽카드가 많은 시간을 소모하는 두 작업이 렌더링 옵션 세팅, 드로우콜 이므로 이 두 작업을 줄이는 식의 구조를 짜도록 했다. 배치 처리를 활용, 같은 렌더링 옵션에 해당하는 것들은 한번 세팅한 시점에 모두 그려내도록 하고 카메라 절두체 컬링 기법을 사용하여 그려내야 할 객체들을 구분짓도록 하였다. 게임 기능구조에서는 게임세계에 생성된 객체들의 관리를 신경 써야 했다. 끊임없이 생성, 파괴, 유지가 되는 스타크래프트의 건물과 유닛들을 효율적으로 관리하기 위해서 여러 방법을 고려하던 중 STL의 컨테이너를 활용하는 방법을 채택했다. 컨테이너에 미리 일정량의 객체들을 만들어 놓고 이를 관리하여 게임 진행중에 메모리 할당으로 인한 프레임저하가 발생하지 않도록 하였다. 패킷 통신구조에서는 우선 통신모델 선정이 중요했다. 본인은 사용경험이 있어 활용하기 편한 '윈도우 비동기 셀렉트 모델'을 선택했다. 패킷 수신에 경우 윈도우 메시지로 오는 패킷을 해석, 적절한 처리함수를 통해 다음 갱신 때 이를 반영할 수 있도록 하였으며 발신의 경우는 사용자 입력의 경우는 이 역시 윈도우 메시지이므로 같은 방식으로 처리하였으며, 그 외에 갱신에 의해 발생하는 경우는 갱신 관련 함수에서 패킷을 생성하여 발송하도록 하였다. 처음 기획 당시 사용자의 입력 외에 유닛들이 생성하는 패킷을 통합해서 전송할지, 객체 별로 전송할지에 대해 논의를 했다. 스타크래프트가 어떠한 방식을 사용한지는 모르기에 논의 끝에 'Terran War'는 객체 별로 패킷이 생성되면 바로 보내는 식으로 개발을 진행하였다.지형을 그려낼 때 우선은 그래픽 디자이너와 게임세계 단위 설정 및 모델 맵 사용여부를 논의해야 했다. 지형에 경우는 실제 게임에서 적혀 있는 수치가 어떠한 기준으로 설정되어 있는지 확인할 수 없어 직접 게임을 실행하여 가장 작은 유닛인 '해병'을 기준으로 잡고, 직접 몇 칸인지 세어 보았다. 모델 맵 사용여부에 경우는 당시에 본인이 지형 렌더링이 3D모델 렌더링보다 우선이었으며 모델을 게임세계로 불러오는 '임포팅'이 준비되어 있지 않아 모델 맵 대신 다이렉트X 11의 기술인 테셀레이션을 활용하기로 하였다. 이에 그래픽 디자이너는 이를 위한 지형 표면 이미지와 높이 표현을 위한 이미지를 직업하였다.PNG파일을 읽어 필요한 높이값 1바이트만을 사용하는 RAW형식으로 추출하여 저장한 뒤, 이를 불러들여 사용한다.  
     
   마우스의 활용이 많은 스타크래프트인 만큼 2D의 모니터 내의 마우스로 3D 가상세계를 클릭하는 '피킹' 개발을 진행했다. 개발 전 기획단계에서 게임 카메라에 대해 이야기를 마쳤지만 지형을 그려내고 간단한 객체를 띄워보며 그래픽 디자이너와 함께 실 수치를 조절해야 했다. 피킹은 사용자에게 보여지는 화면에서부터 시작되므로 이 부분에 대해 논의할 때 실제 게임을 작동해 3DMAX에서 가상세계를 구현해보고 그것으로부터 카메라 정보를 얻어 설정함으로써 스타크래프트와 유사한 카메라 뷰를 얻게 되었다. 이 카메라를 기반으로 게임세계에 있는 객체를 선택하고 다른 곳으로 이동시키는 프로그램을 작성할 수 있었다. 피킹의 방식은 다양한데 본인의 게임에서는 심층적인 피킹은 필요하지 않을 것이라 판단하여 객체의 피킹은 '바운딩 박스'를 이용했고 지형 역시 같은 방법이지만 좀 더 심층적으로 지형을 구성하는 삼각형의 어느 위치인지 얻어내는 식으로 구현하였다. 또한 이를 위해 객체의 피킹을 먼저 검사하고 이후 객체의 피킹이 아닌 경우 지형의 피킹을 검사하는 식의 순서를 신경 써서 작업했다. 그 뒤 단일 객체가 아닌 다중 객체를 피킹하기 위한 드래그 컨트롤을 개발했다. 방식은 클릭이 시작된 지점과 끝나는 지점으로부터 사각형을 만들어내어 그 범위로부터 해당되는 객체들을 얻는 것이다. 키보드 이벤트는 추후 서술할 사용자 인터페이스 부분에서 설명하도록 하겠다.  
   3DMAX에서 작업한 모델을 프로그램에 불러오기 위해 어떠한 형식을 사용할 것인지에 대해 생각해야했다. 또한 다이렉트X 11부터는 툴 내에서 모델 및 애니메이션을 위한 컴포넌트가 존재하지 않기에 따로 엔진을 쓰거나 직접 만들어야 했다. 형식은 가장 대중화된 AutoDesk의 FBX 확장자 모델을 선택했으며 엔진 사용은 하지 않으므로 파일로부터 필요한 정보를 추출하기 위한 '익스포터'와 프로그램 내에서 동작시키기 위한 '임포트' 관련 작업을 진행하였다. AutoDesk사에서 제공하는 익스포팅 샘플을 통해 스키닝 애니메이션을 위해 필요한 정보들만 추출하여 프로그램에 사용할 수 있도록 파일을 따로 만들고, 애니메이션 컨트롤 컴포넌트를 제작한 뒤, 애니메이션 정보 저장을 위해 프레임워크의 게임 기능구조를 확장했다. 원래의 객체들을 인스턴싱으로 그려낼 때에는 필요한 인스턴싱 정보를 그래픽카드가 활용하는 '상수 버퍼' 리소스에 담아서 사용하면 문제가 없었으나, 애니메이션 정보가 추가되면서 하나의 상수버퍼로는 그 양을 다 담아낼 수 없었다. 방법을 강구하니 두 가지 해결안이 떠올랐는데 첫째는 상수버퍼를 여러 개 사용하는 것이고 다른 것으론 '구조화 버퍼' 사용이라는 셰이더 상위모델부터 지원하는 기능으로 개발하는 것이었다. 첫번째 방법은 렌더링 구조에 변화를 주는 부분이 생겨 가능한 바꾸지 않는 방향인 두번째 방법을 사용하였다.이 구조화 버퍼에 현재 그려낼 유닛들의 정보를 담고 인스턴싱으로 한번에 그려내는것이다.  
     
   아군의 객체가 가지는 시야의 범위를 표현하기 위한 방법으로 먼저 떠오른 것은 객체별로 조명을 갖는 것이었다. 허나 1대1 대전을 상정하고 있다 해도 유닛 및 건물의 수는 수백 개가 넘으므로 이는 적절한 방안이 아니었다. 더불어 이미 정찰한 지역에 대한 처리 등을 구현하기 위해선 그 지역에 대한 정찰여부의 정보가 필요해졌기에, 맵을 적절한 크기의 2차원 격자로 나누어 시야와 안개를 처리하도록 했다. 아군 객체의 시야 내 들어오는 격자의 점을 갱신하는 방식이다.이를 통해 미 정찰 지역, 현재 시야를 확보한 지역, 정찰한 적은 있으나 현재 시야는 아닌 지점, 그리고 언덕처리를 표현할 수 있었다.  
     
   그림자 표현 기법에는 여러 종류가 있으며 그에 따른 차이가 확연하다. 본인은 제작하려는 게임에서 가장 적절한 그림자 기법이 무엇일지를 고민했다. 판단으로는 간단하게 원형 그림자를 표현해도 크게 문제가 없을 것으로 보았지만 연구의 과제이기도 하고 당시에는 그림자 기법에 대한 어떠한 테스트 프로그램도 만들지 않은, 구현해본 적이 없었기에 이왕 개발하는 김에 표현력이 더 좋은 그림자 기법을 채택하기로 했다. 그래서 선택한 것은 '투영 텍스쳐 그림자 기법'으로, 그림자 정보를 먼저 만들어내고 이후에 이 정보로 그림자를 표현하는 멀티 패스 렌더링 기법이다. 이를 통해 객체의 모양과 거의 흡사한 그림자를 표현할 수 있었다.게임 화면에 보여지는 미니맵이나, 명령어 창, 유닛 정보, 현재 피킹된 유닛들에 대한 표시, 자원 등을 표현했다. 이러한 기능을 개발할 때 본인은 블렌딩 여부를 고민해야했다. 이유는 블렌딩 상태변화를 자주 호출하게 될 것이라 생각했기 때문이다. 인터페이스의 양도 많았고 그 레이어 역시 여러 층이었기 때문인데, 본인은 렌더링의 순서를 잘 지킨다면 블렌딩을 사용하지 않고도 가능할 것이라 판단하여 진행하였다. 결과적으로는 성공이었다. UI를 그릴 때에는 렌더링 파이프라인의 깊이 검사를 꺼 놓은 뒤 필요한 순서대로 이미지들을 그려 나갔다. 이 부분에서 자주 사용되었던 렌더링 기법은 기하셰이더이다. 이를 통해 UI가 그려질 위치의 정점 하나로부터 사각형을 만들어낼 수 있었다. 다음으로 그 이미지들에 대한 사용자 입력 매핑은 많은 양의 수작업이었다. 가령 건설로봇을 선택했을 때와 전투순양함을 선택했을 때 표현되어야 하는 이미지와 키 입력에 대한 매핑이 다르듯, 유닛 및 건물 그리고 중립자원 객체까지 다양한 경우마다 그에 따른 적절한 이미지와 키 입력 처리를 해야 했기 때문이다. 난이도 있는 부분은 아니었지만 시간이 많이 들고 프레임워크 구조를 계속해서 확장하게 되었다. 또한 입력에 따른 서버와의 통신, 그리고 그 결과에 대한 처리와 후에 서술할 사운드 처리까지 모든 것을 통합적으로 진행해야 했다. 여담이지만 객체를 부대지정하고 그 부대번호를 연속 두 번 누를 시 그 객체의 위치로 카메라가 이동하는 기능이 있는데, 이와 같이 연속 두 번 입력 사이에 시간을 검사하는 것을 구현하는게 본인에겐 새롭고 흥미로웠다.  
     
   다음으론 객체들의 공격 및 피격, 각종 표현되는 스프라이트 이미지에 대한 표현이다. 파티클 시스템의 경우 CPU에서 파티클의 생성, 파괴, 유지를 관리하는 방법과 GPU에서 관리하는 방법이 존재한다. 역시 이번에도 무엇이 가장 본인의 프로그램에 효율적일지 판단한 결과 CPU 파티클을 선택하였다. 이유는 파티클의 양이 많기는 하지만 추가적인 파티클 생성은 그래픽 디자이너가 적절한 이미지로 대체해주기로 했으며(클라이언트 개발시간이 많이 부족하여 협의한 결과이다.) 관리가 훨씬 편리했고 무엇보다 서버에 공격메시지를 보낼 때와 파티클의 생성의 시점을 맞추는 것이 용이했다. 본인은 게임 내 파티클의 종류로 바로 발생하는 즉발성 파티클과, 목표를 향해 날아가는 투사체성 파티클 두 종류로 구분하여 관리했다. 그리고 이를 구분짓기 위해 스타크래프트 내 그려지는 표현을 보고 본인의 게임에서 유닛 별로 구분하여 설정했다. 렌더링 표현은 사용자 인터페이스에서 사용했던 기하셰이더 기법을 활용했다. 파티클 정보를 최대한 간소화하고 적절한 관리 구조를 구현함으로써 CPU에서 처리해도 프레임레이트에 큰 영향을 끼치지 않음을 확인했다.마지막으로 사운드 재생의 경우는, 스타크래프트 내에도 워낙 다양한 사운드가 존재하므로, 사운드 풀과 같은 적절한 관리구조를 지원하는 컴포넌트를 사용할 필요성을 느꼈다. 정보를 모아보니 FMOD라는 컴포넌트를 알게 되었고, 사용법을 익힌 뒤 바로 프로그램에 투입했다. 객체들의 생성, 파괴, 그리고 선택 시 재생되는 상태음 등을 다양하고 상황에 맞게 재생할 수 있도록 개발했다. 부족한 점은 개발한 이후에 깨달은 부분인데 객체를 선택하고 상태음이 재생되는 중 화면을 그 객체로부터 벗어나면 상태음이 정지되어야 하는데 그런 부분은 신경 쓰지 못했다. 여담으로 만족스러운 부분은 객체 선택 시 상태음을 다양하게 추출한 뒤 적용하는 부분을 신경 써서 했다.  
    **2.3 그래픽**졸업연구를 시작하기에 앞서서 이번 연구 목적을 생각해 보았으며 목표를 설정하게 되었다. 그래픽 리소스의 최적화를 통해 유저가 원활한 플레이를 하도록 하는 것이 이번 졸업연구의 목표가 되었다.  
   [그림1]졸업연구 목표 및 기술요소졸업연구 1년6개월 동안 스타크래프트2의 그래픽 리소스 조사 및 분석이후에 제작하기에 부족하다고 판단하였다. 조사를 통하여 그래픽 리소스의 항목을 정리하였으며 팀원들간 기획을 하면서 리소스의 비중을 조정하였다.  
   리소스를 제작하기 이전에 클라이언트 프로그래머와 회의를 하여 게임플레이의 카메라 셋팅, 게임월드에서의 측정단위를 설정하였다. 카메라는 3Ds max 툴을 사용하여 카메라를 배치하고 FOV와 카메라 셋팅 수치를 클라이언트가 다이렉트에 적용할 수 있도록 기재하여 카메라 셋팅을 완료하였다.[그림2] 카메라 셋팅  
   다이렉트 내에서 스타크래프트2의 실제 단위를 구현하는 것은 매우 어려운 과제 였고 이를 해결하기 위하여 자체적으로 최소단위를 설정하고 최소단위에 비례하여 맵의 크기를 클라이언트 프로그래머와 회의를 통하여 설정완료 하였다.  
   이때 사용된 최소단위는 게임내에서 가장 작은 유닛인 테란 종족의 마린을 게임에서의 최소단위로 설정하였다.  
   제작해야 할 그래픽 리소스의 항목은 UI, 3D유닛, 3D건물 플레이 맵 텍스처, 그리고 이펙트 스프라이트 이렇게 크게 다섯가지로 분류를 하였다. 회의를 통해 3D유닛의 개수나 종류, 컨셉을 정하고 맵의 종류와 크기를 설정하였다.  
   게임플레이의 순서도를 확인하고 UI의 전체적인 작업내용물을 확인하였다.  
   테란 종족을 선택하여서 3D유닛의 개수, 건물의 개수도 수치로 정확하게 구분하였으며 작업순서는 유닛을 먼저 모델링하고 텍스처를 제작하고 3D건물도 같은 방식으로 제작하였다. 모델링은 3Ds max를 이용하여 제작을 하였으며 엔진내의 추가적인 플러그인은 사용하지 않았다. 테란종족의 유닛은 14종 건물은 15개로 측정하였으며 애니메이션 및 이펙트는 전부 종족 모델링 구현이 완료된 후에 작업을 들어갈 계획을 하였다.  
   3D 모델링 리소스 제작을 위해 연구목표를 다시 확인하였다. ‘그래픽 리소스 용량 줄이기’를 목표로 설정하고 버텍스와 폴리곤의 양과 데이터 용량이 어떤 연관관계가 있는지 확인하였다. 처음엔 폴리곤의 양이 절대적으로 많아질수록 용량이 늘어날 것이라 생각했는데 폴리곤이 늘어나는 것 보다 중요한 것은 버텍스의 총 개수라고 알게 되었다.   
   이번 주 연구목표 이외에도 스스로 발전을 하기 위하여 소목표를 정하게 되었다. 그래픽 리소스 작업에서 가장 중요시 하게되는 세가지 항목 중 시간이라는 주제를 가지고 어떻게 하면 똑 같은 작업도 더 빠르고 일관성 있게 작업을 할 수 있는가 고민하였다. 여러가지 제작 방법을 시도해 보았고 가장 시간이 적게 걸리는 방법을 채택하였다. 완성할 모델을 파츠별로 구분하여서 미리 원화 작업을 거치고 부품을 만들고 조립을 하는 제작 과정을 나만의 연구 방법으로 설정하였다. 이 방법은 평소 교내 모델링 수업 시간때 만들던 모델링의 완성시간보다 훨씬 줄어들었으며 불필요한 버텍스와 폴리곤을 빠르게 찾아내어 없애는 작업까지 할 수 있어서 연구 목표를 충족하며 개인 소목표도 충족하게 되는 결과를 얻게 되었다. [그림3]모델링 제작방법  
   [그림4]각 유닛당1mb이내의 용량  
   건물 모델링 작업 역시 유닛 모델링 작업방법과 같은 방법을 사용하였다. 이로써 모델링 하는 시간을 대폭 줄여서 텍스처 작업이나 시간이 많이 걸리는 애니메이션 작업에 조금 더 시간분배를 하게 되었다.  
   모델링이 끝난 이후에 처음 클라이언트 프로그래머와 협의를 통하여 정해진 게임내 유닛 단위를 이용하여 유닛들의 크기를 설정하였고 이에 맞게 건물들의 크기 또한 조절하였다. 그리고 마지막으로 모델들을 실제로 카메라 셋팅을 마친 3Ds max 공간 상에서 띄워 보았고 실제 스타크래프트와의 차이를 줄이기 위해 모델의 크기를 전체적으로 수정 작업을 하였다.  
   이때까지의 작업은 게임 플레이를 하면서 원작과의 이질감을 줄이고 연구목표를 이루기 위해 노력하였다.  
   모델링 작업을 마치며 텍스처 제작 작업을 시작하였다. 모델링에 씌워질 텍스처의 종류들과 그리고 렌더링하는 방법을 알아보기 위하여 직접 게임내에서 환경설정에 들어가 그래픽옵션을 조정하는 설명을 보면서 구현해야 할 내용을 분석하였다. 필요로 하는 텍스처 맵은 디퓨즈 컬러맵, 아웃라인을 살리기 위한 노말맵, 조명 반사를 위한 스펙큘러맵 이렇게 세가지 맵 텍스처가 필요하다고 분석하였다.  
   쿨라이언트 프로그래머가 아직 다중 텍스처 사용이나 모델링을 월드에 띄워볼 수 있는 상황이 아니었기에 디퓨즈 컬러맵 텍스처를 제작할 때 최대한 높은 퀄리티가 나올 수 있도록 손맵을 제작하였다. 주요한 디테일은 레퍼런스를 참고하며 간단한 무늬 및 재질 표현은 손맵 제작 방식으로 작업을 진행하였다.  
   [그림5]1024x1024 텍스처맵  
   텍스처 맵을 3Ds max 상에서 확인해보면서 실제 스타크래프트2의 인게임 스크린샷과 단순히 비교를 했을 때에는 많은 이질감이 있었고 이를 해결하기 위해서 채색톤을 확인하고 모니터 색상환 및 출력에 관해서도 연구해보았다. 어떠한 절대적인 기준을 잡는 것보다 스타크래프트2와 얼마나 동화될 수 있는 가를 판단하기 위하여 인겜내의 스크린샷에 직접 제작한 모델링과 텍스처를 적용하여 이질감을 줄이기 위하여 노력하였다. 유투브 영상에 공개된 스타크래프트2 시기별 개발 영상을 보면서 그래픽 적인 퀄리티를 진행하기위해 어떠한 방식을 사용했고, 순서는 어떻게 했는지 이해하고 연구하였다.  
   [그림6]텍스처수정작업이후 렌더링  
   텍스처까지 구현이 완료 되었고 다음으로는 애니메이션 작업 일정을 확인하였다. 주로 모든 유닛들이 idle, move, attack, die 이렇게 네가지 애니메이션 클립이 공통적이었으며 die는 non loop. 형식의 클립이였고 idle은 다른 클립의 이어짐과 평상시 명령이 없을때의 상태를 맡은 클립이다. 따라서 idle에서부터 move와 attack, die 클립이 생성되었음을 알게 되었고 축의 이동이나 각 클립별 키 프레임 수를 설정하는 것을 클라이언트 프로그래머와 회의를 통해 작업을 진행하였다.  
   애니메이션 제작 이후 게임월드상에서 애니메이션이 포함된 fbx 파일을 로드 했을 때 마다 수많은 시행착오를 겪었다. 아예 모델링데이터가 꼬여서 나와 제대로 작동하지 않는 경우, 애니메이션 키 값이 반절만 표현되는 경우 등등 팀원들과 회의를 통해 여러가지 방법으로 데이터를 제작해보고 수정하며 결과적으로 애니메이션을 게임월드상에 띄울 수 있게 하였다.  
   모든 모델링 리소스가 준비되었고 그 이후로 맵 제작을 고민하게 되었다. 타일맵 형식으로 제작할 것인지 아니면 메쉬 데이터인 통맵 형식으로 제작할 것인지 고민하였고 용량을 줄이면서 게임플레이의 최적화를 위해선 한 장의 플레인을 깔고 하이트맵을 제작하여서 맵을 구현하였다.  
   [그림7]게임맵 제작  
   전반적인 게임 플레이가 가능할 때 UI작업을 하였다. 압축률이 가장 좋고 alpha값을 가지고 있는 dds 포맷형식으로 UI리소스를 제작하려고 했으나 포멧을 변환하면서 불필요한 데이터까지 들어갔음을 알게 되고 png 형식으로 리소스를 제작하였다.  
   게임플레이를 지장없이 할 수 있게 되었을 때 이펙트 작업을 시작하였다. 원작인 스타크래프트2 에서의 이펙트는 3D파티클을 이용하여 그래픽 리소스를 이용하여 3D이펙트를 만들 생각이었으나 클라이언트 프로그래머와 협의가 잘 이뤄지지않아 2D스프라이트 이미지로 대체하였다. 여러 번의 프레임 수정과 스크립트를 이용하여 직접 이펙트 투사체의 속도나 프레임 변환 시간 등등을 조절하면서 가장 적절한 이펙트를 만들었다.  
   [그림8]이펙트 스프라이트  
   마지막 게임 전시 전까지는 이펙트 리소스를 다듬고 텍스처 맵의 색상을 조절하고 팀원들과 함께 버그 및 수정사항을 함께 찾았다.  
     
   **3. 결 론**본 논문에서는 대표적인 높은 평가를 받고 있는 실시간 전략 시뮬레이션 게임 중 ‘블리자드’의 ‘스타크래프트’를 살펴보고 이를 학과에서 배운 내용을 기반으로 개발 기법을 연구하고 게임서버 및 클라이언트를 구현하고 필요한 모든 리소스를 직접 제작하였다.  
    [그림 1] 시작화면  
    [그림 2] 자원캐는 장면  
    [그림 3] 전투장면  
    [그림 4] 전투장면   
   [그림1~4]는 개발 결과물이다. 차례로 게임을 시작한 직후 화면, 자원을 캐고 있는 모습, 전투장면1, 전투장면2이다. 클라이언트와 서버간의 동기화가 잘 맞고 그림자와 지형을 똑같이 구현하였다. 하지만 실제 게임의 테크트리를 구현하지 못하였다.  
     
   **4. 부 록  
     
   4.1 게임 실행 환경**  
     
   셰이더 모델 5.0을 지원하는 그래픽 카드  
   윈도우 8.1 이상 혹은 DirectX SDK가 설치된 PC  
   **4.2 게임 설치**  
   압축 해제 시 다음과 같은 구성을 볼 수 있다.  
     
   ‘ReadMe’ 텍스트 문서를 실행하면 실행방법에 대한 안내를 해 놓았다.  
     
   **4.3 게임 진행**  
   1) 서버 실행 후 클라이언트 연결  
     
   2) 방 생성 및 참가 후 시작  
   ****  
     
     
     
   3) 게임 플레이  
     
   ① 유닛 및 명령어 창  
   ② 미니맵 표시  
   ③ 자원 표시  
   ④ 유닛 상태 표시  
     
   자원관리 및 부대운영으로 상대의 모든 건물을 파괴하면 승리한다.  
     
   **4.4 문제 해결**  
   1) 유닛들의 충돌이 불완전해 겹치는 현상이 발생할 수 있다.  
   2) 같은 번호의 방에 중복으로 생성하거나, 이미 진행된 방에 새로이 방을 생성하는 경우 정상적인 연결이 되지 않는다.  
     
   문제 해결 연락처 : 010 5502 3484