네트워크 게임 프로그래밍

Term Project 추진 계획서

게임공학과

2012180022 양기석

2012181042 황성섭

\ 목 차

1. 게임 소개……………………………………………………………………………………………………………………………………….

게임 제목

게임 장르

게임 설명

게임 조작법

게임 흐름

게임 스크린 샷 / 예상도

1. 요구 사항 분석………………………………………………………………………………………………………………………………

기능 요구사항

1. High-level 디자인…………………………………………………………………………………………………………………………..

클라이언트 흐름도

서버 흐름도

스레드 동기화 방식

1. Low-level 디자인……………………. …………………………………………………………………………………………………….

클라이언트 구조체 / 함수

서버 구조체 / 함수

1. 팀원 역할분담……………………….. . …………………………………………………………………………………………………….

양기석

황성섭

1. 개발환경…………………………………. ……………………………………………………………………………………………………..
2. 개발일정…………………………………. ……………………………………………………………………………………………………..
3. **게임 소개**

**\*게임 제목** - Try Again

**\*게임 장르** – RPG, 로그라이크

**\*게임 설명**

이 게임은 MMORPG 게임들의 주요 컨텐츠이기도 한 보스 레이드만을 다룬다.

소수의 인원이 여러 패턴이 존재하는 하나의 보스를 서로 부족한 점을 보안하며

협동하여 물리치는 게임이다.

**최소 1인 ~ 최대 4인**까지 플레이가 가능하고, 각각의 플레이어는 2가지 직업( **궁수**와 **사제** )을

선택하여 플레이 할 수 있다.

각 플레이어는 총 3가지의 스킬을 사용 할 수 있고, 스킬은 모두 논-타겟팅 발사 스킬이다.

각각의 직업은 서로 다른 3가지 스킬을 가지며 스킬별로 고유한 쿨타임을 가진다.

첫 번째 스킬은 마우스를 방향으로 날아가는 약한 공격or회복 스킬이며, 1초의 쿨타임을 가진다.

두 번째 스킬은 짧은시간 이동속도가 증가하는 스킬이며 15초의 쿨타임을 가진다.

세 번째 스킬은 마우스를 클릭한 방향으로 간단한 패턴을 그리며 날아가는 강한 공격or회복 스킬이며, 20초의 쿨타임을 가진다.

**궁수**는 공격을 전담하며 보스에게 스킬을 이용해 데미지를 줄 수 있다.

**사제**는 아군에게 스킬을 맞춤으로써, 아군의 HP를 올릴 수 있다.

**보스**와 플레이어는 적당한 체력을 보유하고 있으며, 체력이 0이하가 되면 사망한다.

**보스**는 총 2마리가 존재하고, 각각 여러 패턴의 탄막을 뿌리며 플레이어를 공격한다.

기존 모티브 게임이 탑 뷰인데 반해, 사이드 뷰 시점을 가지며, 2D로 제작하기로 한다.

또한, 맵 크기에 따라 종/횡 스크롤이 가능하게 한다.

**\*게임 조작법**

WSAD 키로 상하좌우 점프를 할 수 있으며, 마우스 클릭으로 스킬을 사용 할 수 있다.

스킬은 1~3번으로 선택할 수 있다.

**\*게임 흐름**

서버의 IP, 포트 번호 입력

**대기룸**

캐릭터 선택, 게임 시작

**게임진행**

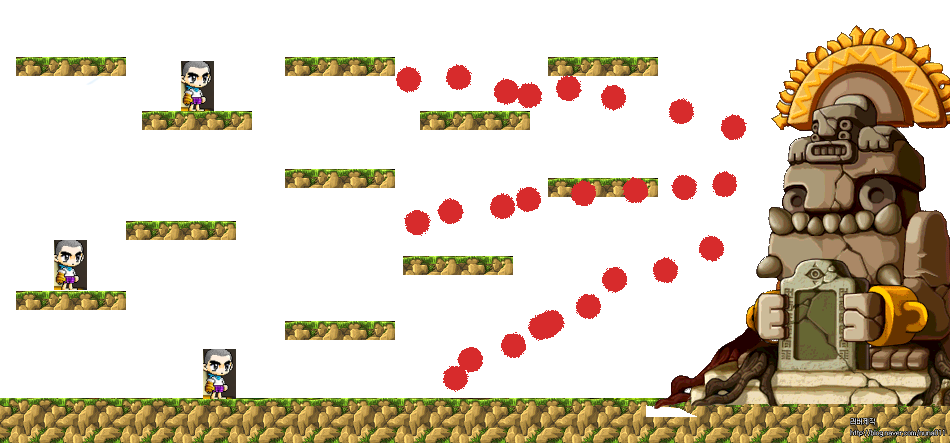
패배

승리

**게임종료**

**\*게임 스크린 샷**

보스

****

탄막

유저1

유저2

유저3

그림1. 게임 예상도

(출처: http://blog.daum.net/\_blog/ArticleCateList.do?blogid=0aMmK&CATEGORYID=37&dispkind=B2201)

1. **요구 사항 분석**

**\*기능 요구 사항**

-네트워크 방식은 TCP로 한다.

-게임 시작을 누르면 서버의 IP, 포트번호를 입력 할 수 있다.

-서버는 최대 4개의 클라이언트를 수용할 수 있다.

-서버에 접속을 성공한 클라이언트는 대기 씬으로 이동한다.

-대기 씬에서는 캐릭터를 변경 할 수 있으며, 캐릭터 이동과 점프가 가능하다.

-대기 씬에서 서버에 접속한 모든 플레이어의 행동이 서로에게 실시간으로 전달된다.

-대기 씬에서 처음 접속한 클라이언트가 방장이 되며, 방장이 게임을 시작할 수 있다.

-방장이 게임을 시작하면 서버에 접속한 클라이언트들은 같은 게임 씬으로 이동한다.

-각 클라이언트는 맵 스크롤시 자신의 캐릭터를 중심으로 보는 (카메라)뷰를 가지며,

고정된 충돌체인 맵 과의 충돌처리는 맵 데이터를 서버에서 한번만 로딩한 뒤,

클라이언트에서 처리한다.

-클라이언트에 공통적으로 사용 될 정보(보스 탄막, 보스, 맵 등)은 서버에서 관리하며,

변경 될 여지가 있는 정보의 충돌처리(캐릭터 - 탄막 등)는 서버에서 처리하고 각 클라이언트에게 알려준다.

- 멀티스레드를 이용한 이벤트 처리로 동기화 과정을 거친다.

- 스레드 동기화시의 문제점을 해결해야 한다.

클라이언트 오류로 데이터를 보내지 못하면 무한 대기 상태에 빠질 수 있다.

따라서, 한 스레드가 일정 시간 동안 데이터를 받아오지 못하면, 다음 스레드로

넘어가는 방법을 생각 할 수 있다.

- 서버의 갱신 속도가 일정하지 않으면 문제점이 생길 수 있다.

Ex)네트워크로 연결 된 타 클라이언트 캐릭터의 움직임이 부자연스러운 움직임을 보인다.

-> 데이터를 처리 할 때 최대 처리의 시간을 생각하여 처리할 데이터가

적을 때에도 평균 처리 시간만큼을 지연시킨다면 조금 더 자연스러운

연출을 기대할 수 있다.

1. **High Level 디자인**

**\*클라이언트 흐름도**

시작

IPPortInput()

False

ChangeScene()

True

False

대기 씬

StartGameMessageToServer

IsChanged()

True

게임시작시

True

GameScene

recvClientCharacterInfo ()

False

IsGameOver()

유저입력

True

False

캐릭터 변경

IsChanged()

sendCharacterChange ()

엔딩 씬

True

recvClientCharacterInfo ()

sendToServer()

**\*서버 흐름도**

시작

True

IsClientPull(4)

False

g\_Client==1

True

sendChiefClient()

False

True

WaitRoom

GameScene

recvFromClient()

recvFromClient()

SyncThread()

SyncThread()

StartGameMessageFromClient()

False

IsGameOver()

ProcessClientData()

False

True

sendToClient()

엔딩 씬

ProcessClientData()

sendToClient()

**\* 스레드 동기화 방식**

**- g\_nTreated ( 처리한 쓰레드 수 ) = 0;**

**-Thread Function-**

쓰레드 2

TimeOut

쓰레드 1

//TimeOut

쓰레드 3

TimeOut

쓰레드 4

TimeOut

recvFromClient**(** ,

**clientInfo[g\_nTreated]**  ,…**)**

**g\_nTreated++;**

**//서버에서의 데이터 처리**

**TreatedByServerInfo**

**( clientInfo )**

TimeOut 시간 = 최대 4~ ms

총 최대 16~ ms 이하

- 클라이언트의 60프레임을 기준으로 60/ 1000 / g\_Client(int) sec 의 최대 스레드 대기시간을 가지며, TIMEOUT시 바로 다음 스레드가 실행 되게 한다.

1. **Low Level 디자인**

**\*** **클라이언트 구조체 / 함수**

- **클라이언트 구조체**

서버로부터 처음에 받는 맵의 정보를 담는 구조체.

**Struct mapInfo {**

**Int \*\*first\_mapInfo;**

**Int \*\*second\_mapInfo;**

**}**

서버에 보낼 데이터의 정보를 담는 구조체.

**Struct ClientInfo {**

**Player player; //플레이어의 정보**

**queue <Skill> skill; //내 스킬들의 정보**

**}**

서버에서 처리한 데이터의 정보를 담는 구조체.

**Struct treatedByServerInfo {**

**Player \*player; //다른 플레이어의 정보**

**Boss \*boss; //보스의 정보**

**queue<Projectiles> projectiles; //스킬들의 정보**

**queue<Effect> effect; //이펙트의 정보**

**}**

- **클라이언트 함수**

bool **recvFromServer( SOCKET s, treatedByServerInfo tbServerInfo, int len, int flags )**

return value : 에러 처리용 값

-서버로부터 데이터를 받는다.

bool **sentToServer( SOCKET s, int len, ClientInfo clientInfo , int flag )**

return value : 에러 처리용 값

- 고정-가변 길이의 데이터 전송방식으로 데이터를 보낸다.

void **IPPortInput**() - 게임시작시 IP/포트번호 입력을 한다.

bool **ChangeScene( int SceneNum )**

return value : 씬 변경 성공여부

- 씬을 변경하는 함수

void **sendCharacterChange(SOCKET s, BOOL IsChangedCharactor, int len, int flags)**

- 캐릭터 변경시 서버에 데이터를 보낸다.

void **recvClientCharacterInfo(SOCKET s, treatedByServerInfo tbServerInfo**

**, int len, int flags)**

- 캐릭터들의 정보가 바뀌면 서버로부터 캐릭터들의 정보를 얻어온다.

bool **IsChange() -** 클라이언트의 바뀐 정보가 있는지 여부 확인

bool **IsGameOver() -** 게임 종료 조건 체크

bool **StartGameMessageToServer( int sceneNum)**

자신이 방장 속성(bool IsChief==true)일 때 게임시작 버튼을 누를 경우

해당 정보를 변경.

**\*** **서버 구조체 / 함수**

**- 서버 구조체**

클라이언트에 처음에 보낼 맵의 정보를 담는 구조체.

**Struct mapInfo {**

**Int \*\*first\_mapInfo;**

**Int \*\*second\_mapInfo;**

**}**

클라이언트로부터 받을 데이터의 정보를 담는 구조체.

**Struct ClientInfo {**

**Player player; //플레이어의 정보**

**queue <Skill> skill; //내 스킬들의 정보**

**}**

서버에서 처리한 데이터의 정보를 담는 구조체.

**Struct TreatedByServerInfo {**

**Player \*player; //다른 플레이어의 정보**

**Boss \*boss; //보스의 정보**

**queue<Projectiles> projectiles; //스킬들의 정보**

**queue<Effect> effect; //이펙트의 정보**

**}**

- 서버 함수

클라이언트로부터 받은 데이터로 처리해야 할 데이터를 처리한다.

**TreatedByServerInfo** **processClientData( ClientInfo \*p\_clientInfo )**

return value : 처리된 정보

TreatedByServerInfo \*treated;

내부 의사 코드

{

// 모든 오브젝트의 위치 값 갱신

RenewObjects();

// 각 플레이어의 충돌에 따른 정보 변경

For ( i = 0; i < g\_Client; i++){

Coll\_Function( player[i], nowStage->boss , treated );

For( auto j = projectiles.begin(); j != projectiles.end(); j++){

Coll\_Function( player[i], \*j , treated );

Coll\_Function( nowStage->boss, \*j , treated);

}

}

return treated

}

bool **recvFromClient ( SOCKET s, ClientInfo** **clientInfo, int len, int flags )**

return value : 에러 처리용 값

- 클라이언트로부터 데이터를 받는다

bool **sentToClient( SOCKET s, int len, treatedByServerInfo tbServerInfo, int flag )**

return value : 에러 처리용 값

- 고정-가변 길이의 데이터 전송방식으로 데이터를 보낸다.

bool **IsClientPull(int Client)**

return value : true시 클라이언트를 더 이상 받지 못한다.

- 인자 - 총 받을 수 있는 클라이언트 개수 ( 관련 속성 : g\_Client )

bool **sendChiefClient(SOCKET s, BOOL** **isChief, int len, int flags )**

- 클라이언트에게 방장 권한을 준다.

bool **StartGameMessageFromClient(int sceneNum)**

- 게임 씬으로 넘어간 정보를 변경하고 성공여부를 리턴.

bool **SyncThread( int nThread )**

- 멀티스레드 작업을 할 때, 각 쓰레드를 모두 처리했는지 단순 체크 함수

인자 – 스레드 개수, if( g\_Client == 처리한 스레드 개수 ) return true

1. **팀원 역할분담**

\*양기석(PM) - 클라이언트 제작 ( 충돌 처리와 캐릭터 관련 개발 )

- 서버에서의 충돌 처리

- 주로 클라이언트의 네트워크 기능 구현.

\*황성섭 - 클라이언트 제작 ( 스킬과 보스 관련 개발 )

- 캐릭터와 보스의 탄막 패턴 정보 데이터 관리 및 보내기

- 주로 서버에서의 네트워크 기능 구현.

1. **개발환경**

Visual Studio 2013

Github (프로젝트 관리)

1. **개발일정**

|  |  |  |  |
| --- | --- | --- | --- |
| 날짜 | 양기석 | 황성섭 | |
| 11.6 | 개발 환경 통합 , Github 개설 | | |
| 11.7 | 게임 프레임 워크 제작 | | |
| 11.8 | 게임 프레임 워크 제작 | | |
| 11.9 | 게임 맵 구성 (2가지 맵) | 관련 리소스 수집, 제작 | |
| 11.10 | 게임 캐릭터 클래스 제작 | 플레이어 스킬 제작 | |
| 11.11 | 맵 스크롤링 구현, 맵 충돌 처리 구현 | 플레이어 스킬 제작 | |
| 11.12 | 보스 몬스터 제작 | 플레이어 스킬 제작 | |
| 11.13 | 게임내 충돌 처리 구현 | 보스 탄막 패턴 제작 | |
| 11.14 | 캐릭터 애니메이션 | 보스 탄막 패턴 제작 | |
| 11.15 | 코드 병합 및 클라이언트 완성 | | |
| 11.16 | 코드 병합 및 클라이언트 완성 | | |
| 11.17 | **sentToServer()**  **recvFromServer()** | **sentToClient ()**  **recvFromClient()** | |
| 11.18 | **IPPortInput()** | **IsClientPull()** | |
| 11.19 | 코드 병합, 서버와 클라이언트간의 통신 확인, 다중 접속 확인 | | |
| 11.20 | **ChangeScene()** | **SetChiefClient()** | |
| 11.21 | **recvClientCharacterInfo()** | **IsChanged()** | |
| 11.22 | **sendCharacterChange()** | **StartGameMessageFromClient()** | |
| 11.23 | 코드 병합, 대기 씬 및 게임 씬 데이터 확인 | | |
| 11.24 | **StartGameMessageToServer()** | | **IsGameOver()** |
| 11.25 | **SyncThread()**  **processClientData() 기본 처리** | | **sendToClient() 확장** |
| 11.26 | **processClientData() , 멀티 스레드 동기화 작업** | | |
| 11.27 | **processClientData() , 멀티 스레드 동기화 작업** | | |
| 11.28 | **processClientData() , 멀티 스레드 동기화 작업** | | |
| 11.29 | 멀티 스레드 동기화, 공동작업 마무리 | | |
| 11.30 | 프로토 타입 테스트 및 버그 픽스 | | |
| 12.1 | 코드 병합 , 각각 미흡한 부분 보완 | | |
| 12.2 | 코드 병합 , 각각 미흡한 부분 보완 | | |
| 12.3 | 오류 수정 및 버그 픽스 | | |
| 12.4 | 오류 수정 및 버그 픽스 | | |
| 12.5 | 오류 수정 및 버그 픽스 | | |
| 12.6 | 제출 | | |