네트워크 게임 프로그래밍02반

Term 프로젝트 추진 계획서

-Fall Guys-

게임공학과 2019180030 이양희

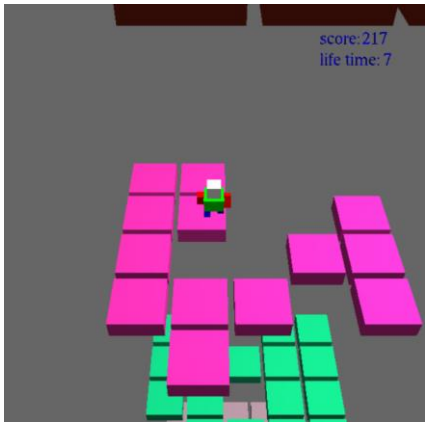
게임공학과 2019182019 변승은

1. 애플리케이션 기획

- 게임 이름: Fall Guys
- 장르: 플랫폼 게임
- 플레이어 수: 2인
- 시점: 3인칭 카메라
- 플레이 시간: 1분

게임 원작 화면(Fall Guys):

게임 화면(1인 플레이 기준):

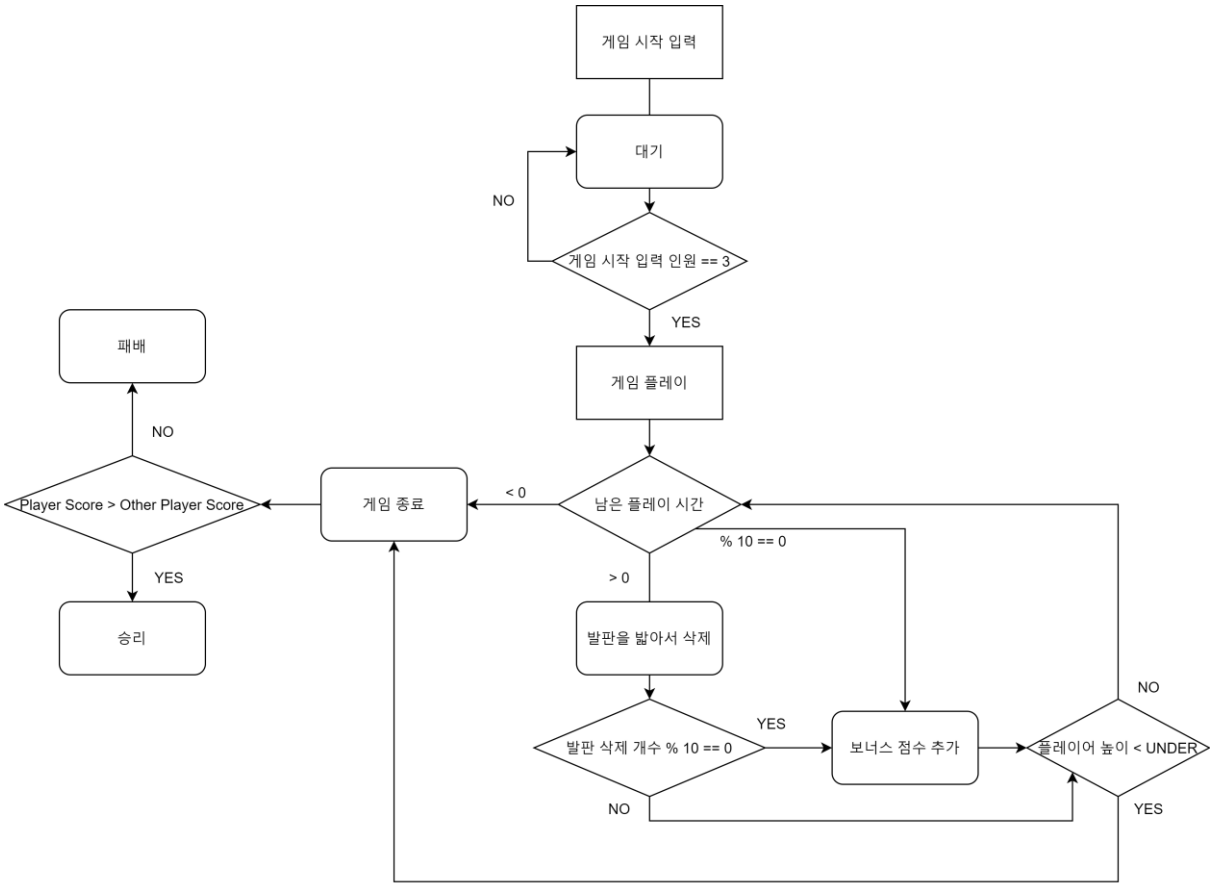


- **간단한 소개**

Fall Guys를 모티브로 하여 제작하였다. 주어진 시간 내에 가장 많은 발판을 밟아 없앤 플레이어가 이기는 게임이다.

: 기존 프로젝트는 OpenGL을 통해 개발되었다. 이는 'Fall Guys'의 '바닥 떨어져유' 스테이지를 솔로 플레이로 전환한 게임이다. 네트워크 통신 기능을 추가하여 2인 플레이 게임으로 변경하는 것이 애플리케이션 기획 목표이다.

- **게임 진행 흐름도**



1.시작 버튼을 눌러 서버에 접속한다.

2.서버에 접속한 플레이어가 2인 이상이면 게임을 시작한다.

3.랜덤 위치에 플레이어를 배치한다.

4.플레이어가 밟은 발판이 떨어진다.

5.다른 발판으로 움직이거나, 아래 스테이지로 떨어진다

4~5번을 반복한다.

6.주어진 시간이 끝난다.

7.두 플레이어간 점수 비교

- **조작 방식**

게임 시작 : 엔터

상하좌우 이동 : w/W s/S a/A d/D키 입력

점프 : 스페이스바

- **점수**

게임 플레이 시간 동안 점수가 1점씩 증가한다.

~~발판을 밟아 없앤 횟수가 10의 배수일 때마다 보너스 점수를 받는다.~~

발판을 밟을 때마다 점수를 추가한다.

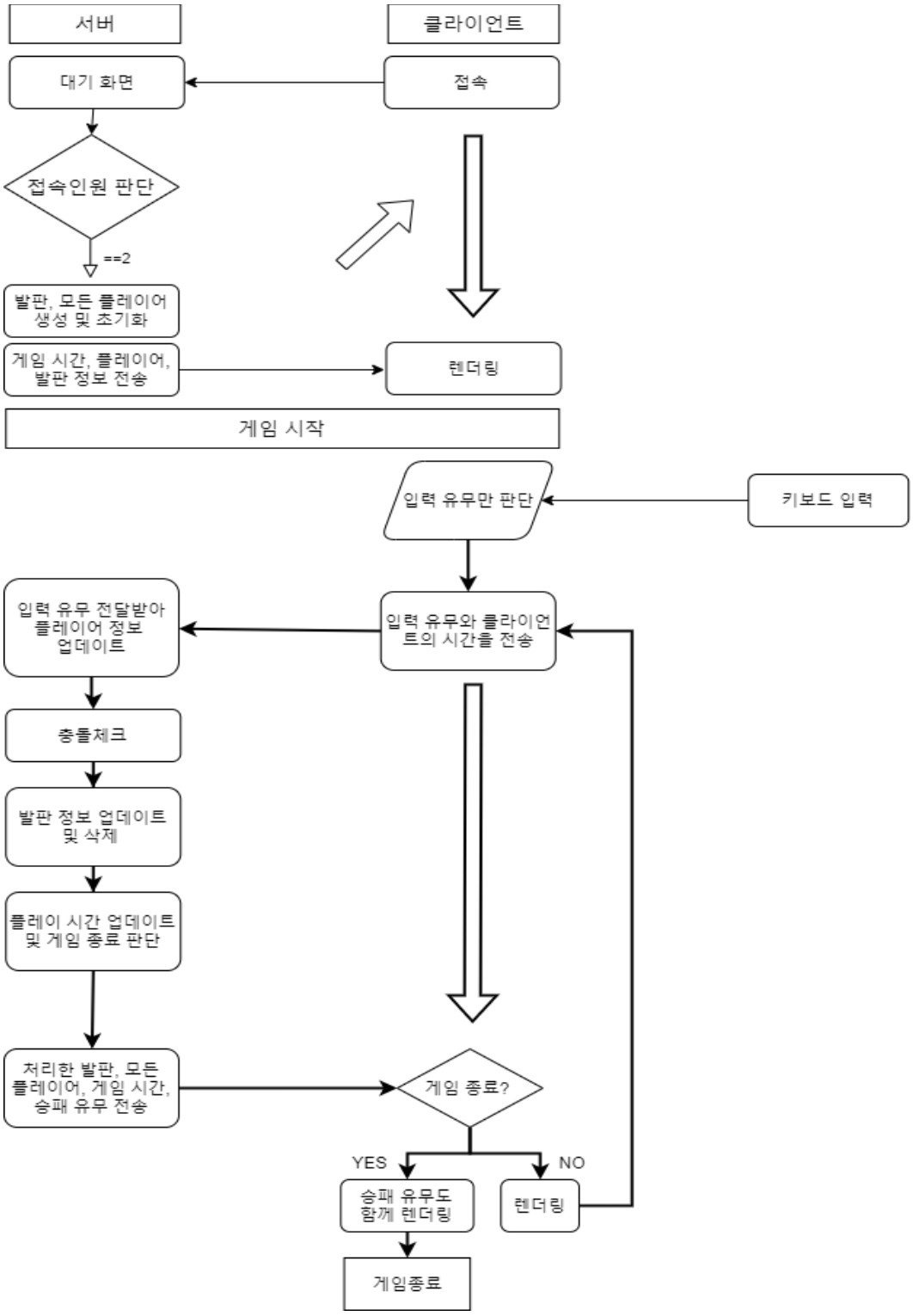
~~게임 플레이 타임(초)가 10의 배수일 때마다 보너스 점수를 받는다.~~

- **종료 조건**

기존 – 플레이어 추락 시 점수 출력 후 게임종료

현 애플리케이션 – 주어진 시간이 끝나면 높은 점수를 획득한 플레이어가 승리

2. High-Level 디자인


서버
클라이언트
대기 화면
접속
접속인원 판단
==2
발판, 모든 플레이어 생성 및 초기화
게임 시간, 플레이어, 발판 정보 전송
렌더링
게임 시작
입력 유무만 판단
키보드 입력
입력 유무 전달받아 플레이어 정보 업데이트
입력 유무와 클라이언트의 시간을 전송
충돌체크
발판 정보 업데이트 및 삭제
플레이 시간 업데이트 및 게임 종료 판단
처리한 발판, 모든 플레이어, 게임 시간, 승패 유무 전송
게임 종료?
YES
NO
승패 유무도 함께 렌더링
렌더링
게임종료

사용 통신 프로토콜: TCP

1. 클라이언트가 게임 시작 키를 누르면 대기 화면으로 전환되고 서버로 접속합니다.
2. 서버에서는 접속 인원이 2명이 될 때까지 대기합니다.
3. 접속인원이 충족되면 플레이 화면을 생성하기 위한 발판의 상태 정보와 플레이어의 캐릭터 생성 및 초기화 작업을 진행합니다.
4. 서버에서 게임 시작 시간을 기록한 후 각 플레이어에게 발판 정보, 서버 시간, 두 플레이어의 캐릭터 정보를 클라이언트 모두에게 전송합니다.
5. 클라이언트는 서버에서 받은 정보들을 받아 데이터를 포맷한 후 렌더링을 시작합니다.

   *렌더링 전 각각의 클라이언트 마다 클라이언트 시간을 저장합니다.
6. 클라이언트가 렌더링을 시작하게 되면 게임 시작 상태를 서버에게도 알립니다.
7. 게임 진행 상태 동안 클라이언트는 키보드의 입력 유무에 대한 정보를 저장한 후 클라이언트 시간과 함께 서버에게 보냅니다.
8. 서버에 도착한 소켓 시간을 비교하여 네트워크 속도 차이를 고려하며, 소켓에 저장된 클라이언트 - 클라이언트 간의 시간 차이를 보정하여 평균 시간을 설정합니다.

   *보정된 평균시간으로 플레이어-플레이어, 발판-플레이어 순으로 충돌 체크를 진행합니다. 또한, 클라이언트-클라이언트 간 플레이 시간 차이를 줄이도록 합니다.
9. 서버는 받은 키보드 정보에 따라 플레이어의 좌표를 변경하고
10. 변경된 두 플레이어 좌표에 대한 충돌체크를 진행합니다.
11. 두 플레이어 좌표의 충돌 처리에 따른 좌표를 결정한 후 각 플레이어와 발판 간의 충돌 체크를 진행합니다.
12. 이후 게임 종료 조건을 판단합니다.
13. 플레이어-발판의 충돌 처리로 발판에 저장된 정보 값을 변경하고, 점수 처리 및 애니메이션 여부도 판정합니다.
14. 서버 시간을 갱신합니다.
15. 서버는 변경된 두 플레이어의 좌표, 애니메이션여부, 발판의 상태 정보 값, 갱신된 서버 시간, 게임 종료 조건 충족 시 승패 유무를 소켓에 담아 각각의 클라이언트에게 전송합니다.

16. 서버로부터 받은 소켓에 담긴 정보를 클라이언트 데이터로 포맷합니다.
17. 게임 종료를 의미하는 데이터 값을 받으면 승패 유무에 따른 결과 화면을 렌더링합니다.
18. 게임 종료를 의미하는 데이터 값이 충족되지 않으면 포맷된 데이터 값에 따라 화면에 게임 월드와 애니메이션을 렌더링 하고, 7번~16번을 반복합니다.
19. 게임 종료 화면으로 넘어갑니다.

3. Low-Level 디자인

- **클라이언트**

역할	변수, 함수

키보드 입력 유무 판단에 사용하는 구조체	Struct InputData{
	bool bUp;
	bool bRight;
	bool bLeft;
	bool bDown;
	bool bSpace;
	Bool bEnter;
	}

점수를 관리하는 변수 (플레이어 멤버변수)	int CPlayer::m_nScore

클라이언트->서버 전송 구조체	struct SendPlayerData{
	InputData Input;
	~~clock_t ClientTime;~~
	}

	SendPlayerData myPlayer

서버->클라이언트 전송 구조체	struct SendGameData{
	~~PlayerMgr p1;~~
	~~PlayerMgr p2;~~
	PlayerMgr PMgr[CLIENT_NUM];
	~~clock_t ServerTime;~~
	~~Bool Win;~~
	vector<FootHold> Bottom
	}

	SendGameData ServerGameData
~~시간 갱신~~	~~TimerFunc()~~
~~플레이어 정보를 전송 구조체에 업데이트~~	~~void UpdateSendData()~~
서버와 통신할 클라이언트의 소켓 정보	SOCKET sock
~~서버의 데이터 수신 시 사용할 소켓 주소 구조체~~	~~SOCKETADDR_IN peeraddr~~
데이터 통신시 사용 소켓 주소 구조체 (지지역 IP,포트번호 설정	SOCKETADDR_IN serveraddr
~~서버의 아이피를 설정하는 변수~~	~~#define SERVERIP~~

서버의 아이피를 저장하는 변수	String ServerIP
서버 포트번호 설정에 사용하는 변수	#define SERVERPORT
~~서버에 게임 시작유무 전송 시 사용하는 변수~~	~~bool IsPlayingGame~~
총 인원수를 설정하는 변수	#define CLIENT_NUM
사망시 상대방 시점으로 카메라 전환하는 함수	void IsChangeCamera()
현재 게임 상태를 출력하는 함수	Print_GameState()

현재 게임 상태를 나타내는 enum class	Enum class EGameState{
	TITLE,
	WAITING,
	PLAYING,
	GAMEOVER
	};

현재 게임 상태를 저장하는 변수	Volatile int CurrentGameState
현재 남은 게임 시간을 저장하는 변수	Volatile int CurrentTime
모든 플레이어가 게임오버되었는지 저장하는 변수	Volitile bool AllPlayerGameOver
플레이어가 게임을 종료하고자 하는지 저장하는 변수	Volitile bool PlayerWantQuitGame
현재 모든 플레이어가 GAMEOVER State인지 검사하는 함수	Bool IsGameOverState()
플레이어 배열에서 자신의 인덱스를 저장하는 변수	Int myIndex
다른 플레이어의 인덱스임을 나타내는 변수	Int otherIndex
현재 카메라가 보여주는 플레이어의 인덱스를 나타내는 변수	Int showIndex
카메라가 보여주는 플레이어를 바꾸었는지 확인하는 변수	Bool bChangeCam

- **서버**

역할	변수
접속 인원을 저장하는 변수	Volitile Int custom_counter
발판 인덱스와 발판 상태 변수	vector<Foothold> Bottom

모든 플레이어를 관리하는 구조체	Struct PlayerMgr{
	~~DWORD portnum;~~
	DWORD threadId;
	CPlayer player;
	Bool Win;
	Bool Mine = false;
	Bool bGameOver = false;
	}

	PlayerMgr Players[]
플레이어와 스레드 ID를 저장하여 관리하는 map	Map<DWORD, PlayerMgr*> ClientManager
데이터를 보낼 때 동기화를 위한 임계 영역 변수	CRITICAL_SECTION cs
~~서버 생성 시간을 저장하는 변수~~	~~clock_t serverInit_time~~
~~현재 서버 시간을 저장하는 변수~~	~~clock_t serverPre_time~~
~~서버 시간의 변화량을 저장하는 변수~~	~~clock_t serverDelta_time~~
~~발판과 플레이어간 충돌 여부를 저장하는 변수~~	~~Bool IscollisionPandF~~
~~플레이어 간 충돌 여부를 저장하는 변수~~	~~Bool IscollisionP1andP2~~
~~플레이어의 위치 정보 갱신 필요여부를 저장하는 변수~~	~~Bool IsNeedUpdateLoc~~
~~게임 시작과 종료를 나타내는 변수~~	~~Bool IsPlayingGame~~
~~플레이어 A,B 식별 변수~~	~~Int portnum~~
~~플레이어의 승패 정보를 저장할 변수~~	~~Bool Win~~
클라이언트와 통신할 서버의 소켓 정보	SOCKET client_socks[CLIENT_NUM]
서버 소켓 주소 구조체	SOCKETADDR_IN serveraddr
클라이언트 소켓 주소 구조체	SOCKETADDR_IN clientaddr
서버 포트번호 설정에 사용하는 변수	#define SERVERPORT

서버의 아이피를 저장하는 변수	String ServerIP
윈속 초기화 시 사용하는 변수	WSADATA wsa
서버를 생성하여 플레이어의 초기위치를 선정하고 서버 데이터를 초기화 함수	ServerInit()
~~접속 인원 조건이 모두 충족되었는지 확인~~	~~Bool IsOkGameStart()~~
~~서버 시간 기록함수~~	~~Void RecTimer()~~
~~서버 시간 갱신~~	~~void UpdateTimer()~~
~~각 클라이언트 간의 시간 오차 보정 함수~~	~~Float timeInterpolation()~~
~~게임 화면에 관련된 정보 생성~~	~~CreateMainGameScene()~~
발판 충돌 체크 함수 (플레이어 간/각 플레이어마다)	bool IsCollideFootholdByPlayer (Foothold, CPlayer*)
발판 충돌 처리 함수	void CheckCollideFoothold(vector<Foothold>&)
~~두 플레이어간 충돌 체크 함수~~	~~bool IsCollidePlayerByPlayer()~~ ~~bool IsCollidePlayerByPlayer(CPlayer& a, CPlayer& b)~~
~~두 플레이어간 충돌 처리 함수~~	~~void CheckCollidePlayers()~~
플레이어의 위치정보 갱신 함수	void UpdatePlayerLocation(CPlayer* p, InputData input)
~~플레이어의 키보드 입력 처리 함수~~	~~void UpdateClientKeyInput()~~
게임 종료 조건 판별 함수(2가지)	~~void CheckGameOver()~~ Bool IsGameOver(CPlayer* player)
모든 클라이언트가 게임 오버되었는지 판별하는 함수	Bool IsAllPlayerGameOver()
클라이언트에게 받은 소켓의 포트번호를 식별하여 필요한 데이터를 설정하는 함수	~~Void SetCilentData(DWORD portnum)~~ Void SetClientData()
발판, 플레이어 생성 및 초기화	FootHoldInit(), PlayerInit()
~~클라이언트로부터 플레이 준비 완료 상태를 전달받는 함수~~	~~Bool IsReadytoPlay(bool isReady)~~
클라이언트와 데이터 통신을 위한 쓰레드 함수	DWORD WINAPI ProcessClient (LPVOID arg)
클라이언트에게 시간 정보를 보내기 위한 스레드 함수	DWORD WINAPI ProcessTime(LPVOID arg)

클라이언트와 데이터 통신을 위한 쓰레드 핸들 변수	HANDLE hClientThread[CLIENT_NUM]
클라이언트에게 시간 정보를 보내기 위한 스레드 핸들 변수	HANDLE hTimeThread
발판 동기화 작업을 위한 이벤트 핸들 변수	HANDLE hFootholdEvent;
~~서버->클라이언트 전송구조체 초기화를 위한 변수~~	~~void InitServerSendData()~~
보내는 데이터의 유형을 저장하는 변수	Short opcode
클라이언트가 자신의 정보와 타인의 정보 구분을 위한 멤버 변수 세팅을 위한 함수	Void SettingPlayersMine(DWORD ThreadId)
스레드 ID를 통해 플레이어를 구분하여 map으로 관리하는 함수	Void CheckInsertPlayerMgrData(DWORD ThreadId)
PlayerMgr끼리 비교 연산을 할 때 사용하는 비교 함수	Bool compare(PlayerMgr& p1, PlayerMgr& p2)
플레이어들의 점수를 비교해 승자를 구분하는 함수	CheckGameWin(DWORD ThreadId)

- **기존 내용**

-서버 도입 전 기존 프로그램 내부 Class 구조와 함수

함수	
GLvoid drawScene(GLvoid);	화면 출력을 위한 씬을 그리는 콜백 함수입니다.
GLvoid Reshape(int w, int h);	씬을 다시 그리기 위한 콜백 함수입니다.
GLvoid Keyboard(unsigned char key, int x, int y);	키를 누를 때 호출되는 콜백함수입니다. -key: 입력 키보드 (ASCII) -x, y: 키보드 입력할 때의 마우스의 위치
GLvoid KeyboardUp(unsigned char key, int x, int y);	키보드를 뗐을 때 호출되는 콜백 함수입니다.
void Timerfunction(int value);	게임 월드 내부 객체의 애니메이션 구현을 위한 타이머 설정 함수입니다.
void make_fragmentShader();	프래그먼트 쉐이더를 읽어 저장한 후 컴파일 하는 함수입니다.
void renderBitmapCharacher(float, float , float, void*, char*);	지정한 위치에 문자열을 출력합니다.
void Print_word(float, float, float, float, int, char*);	지정한 위치에 지정한 길이만큼 문자열을 출력합니다.
void check_GameOver();	플레이어의 위치를 비교하여 플레이어가 추락하였으면 게임오버 상태로 만들고, 이 정보를 화면에 출력합니다.
void check_Bonus();	발판을 밟은 총 개수가 10의 배수이면 개수*5의 보너스 점수를 부여한다. 현재 총 플레이 타임 시간(초 단위)이 10의 배수이면 해당 시간(초)만큼의 보너스 점수를 부여한다.
void check_collide();	현재 플레이어와 충돌한 발판이 있으면, 플레이어는 추락하지 않았다고 설정하고 충돌한 발판은 삭제하기 시작할 것이라고 설정합니다. 삭제한다고 설정한 변수는 발판 벡터 내에서 삭제합니다.
bool collide_box(Foothold, Robot&);	플레이어가 발판 위에 있는 상태인지 확인합니다. 플레이어의 위치는 발판보다 조금 위쪽에 위치하도록 보정합니다.
void Time_score();	현재 게임이 진행되고 있으면 1점을 추가합니다.
void Init_Game();	메인 화면을 생성하고 생성한 값들의 데이터 값을 초기화 하는 함수입니다.

struct Part	로봇을 이루는 부분의 이동, 크기, 회전과 관련된 변환에 관련된 행렬과 색상 정보를 관리하는 구조체
struct Robot	위치와 각도, 로봇을 이루는 각 부분, 움직일 때 애니메이션을 보여주기 위한 행렬 정보를 가지는 구조체
void Part::makePart(int type, bool leftPart)	플레이어 모델링을 위한 정점 데이터를 설정하는 함수입니다.
Robot::Robot()	로봇을 이루는 박스 오브젝트를 생성하고, 위치와 각도, 애니메이션 관련 행렬을 초기화합니다.
void Robot::Update()	현재 각도와 추락했는지 여부, 애니메이션으로 인해 얼마나 움직일 것인지, 로봇을 이루는 각 부분의 위치와 회전 정보를 갱신합니다.
void Robot::Locate()	로봇을 이루는 각 부분의 행렬 정보를 이용하여 위치와 회전 정보를 갱신합니다.
void Robot::Swing()	liftend
void Robot::Walk_anim()	
void Robot::Jump()	함수를 호출하면 로봇의 상태를 나타내는 fall변수를 true로 설정한 후 dy의 증가량을 설정합니다.
void Robot::Fall()	현재 플레이어가 추락 중이라면 dy 값을 점차 감소시켜 아래로 떨어지게 합니다.
void Robot::get_angle()	회전하는 각도를 구한다.

#define foothold_sizex,foothold_sizey,foothold_sizez,	발판 Class의 생성자 호출 시 사용되는 변수 발판의 Scale값의 기준
#define N	발판 개수의 기준 (N의 세제곱=발판 총 개수)
#define foothold_vertex	발판 오브젝트의 정점 개수
#define MAX	실수 값을 랜덤하게 표현하기 위해 사용하는 변수
#define UNDER	게임오버를 판단하는 기준 (플레이어 높이와 비교 시 사용)
class Foothold	발판을 관리하는 Class
void Foothold::Init() {	렌더링 전 이전의 Drawing 정보를 모두 초기화

	후 현재 발판의 Scale, Move, Rotate를 는 함수정한
void Foothold::Pos_Drawing()	Scale,Move,Rotate로 렌더링에 사용 할 Drawing 변수를 업데이트 해주는 함수
void Foothold::Draw_Start()	Foothold::Init(), Pos_Drawing()을 호출하여 발판의 렌더링을 준비하는 함수
void Foothold::Delete()	발판 애니메이션 관리 및 발판 삭제유무 관리
void MakeFoothold(vector<Foothold> &Bottom)	모든 발판을 관리하는 vector 변수인 Bottom를 인자로 받아와 발판을 Bottom에 push_back 해주는 함수

- **개발환경**
- Git (소스트리)
- Visual Studio 2019
- OpenGL

- **팀원 별 역할분담**

변승은	이양희	임보배
서버)플레이어 좌표 업데이트	화면 전환 관련 (타이틀 state, 메인 게임 state 생성)	~~게임 시작 시 서버 시간 기록~~
서버)플레이어와 발판 사이 충돌체크	접속 인원 수 판단	~~클라이언트에서 키보드 입력 처리~~
서버) 발판 정보 업데이트	서버의 발판, 캐릭터 정보 초기화	~~클라이언트-클라이언트 간 시간 보정~~
서버)처리 완료된 데이터들들을 클라이언트로 전송	서버에서 클라이언트로 발판 정보, 게임 시작 시간, 플레이어 정보 전송	
클라) 서버로부터 받아온 데이터를 클라이언트 데이터로 갱신	서버에서 보낸 발판 정보, 게임 시작 시간, 플레이어 정보를 클라이언트에서 수신	~~게임 종료 조건 판단 (플레이 시간 측정 시 또는 발판 충돌 처리 이후)~~
클라) 게임종료 여부에 따라서 승패 여부 출력 (결과 화면)	클라이언트가 수신한 정보를 토대로 발판, 캐릭터 정보 초기화	~~클라이언트에서 서버로 키보드 입력 데이터와 클라이언트 시간 정보 전송~~
클라) 패배 시 생존한 플레이어로 시점 전환하는 기능 추가	게임이 시작되면 게임이 시작되었음을 클라이언트가 서버에게 알림	~~클라이언트에서 받은 키보드 입력 데이터와 클라이언트 시간 데이터를 서버 데이터로 포맷~~
클라이언트에서 키보드 입력 처리	게임 시작 시 서버 시간 기록	
게임 종료 조건 판단 (플레이 시간 측정 시 또는 발판	클라이언트-클라이언트 간 시간 보정	

충돌 처리 이후)		
클라이언트에서 서버로 키보드 입력 데이터 전송		

- **개발일정**

-변승은

11월						
일	월	화	수	목	금	토
	1	2	3	4	5	6
7	8	9	10	11	12	13
				개발시작: 깃 생성, 프레임 워크 짜기 (팀원 다 같이)		Gvoid Keyboard: 키 입력유무 판단 수정 UpdatePlayerLocation() -> 서버) 좌표 업데이트
14	15	16	17	18	19	20

IsCollideFootholdByPlayer, UpdateFootholdbyPlayer 발판, 플레이어 충돌 및 처리(테스트X)	InitServerData()구현, ProcessClient에서 클라이언트로 ServerGameData 전송(테스트X)	Clinet의 DrawScene 정리(모든 플레이어 렌더링), ServerGameData 수신(테스트X)		클라이언트에서 자신의 인덱스 구분(ClientMain 수정), 이벤트 설정		~19 까지 각 팀원들 구현한 내용들 Merge, 충돌내용 검토
21	22	23	24	25	26	27
20일에 작업한 내용 중 버그난 부분 수정 승패 판단(테스트X) :IsGameOverState(), IsGameOver(Cplayer &),CheckGameWin(DWORD)		void ~~check_GameOver()~~ Print_GameState() 로 승패여부 출력 전송 구조체 SendGameData 수정, void ChangeCamera()로 사망 시 카메라 전환기능 구현	디버깅 테스트 (플레이어 업데이트 및 송수신 오류 수정)	~~void ChangeCamera()로 사망 시 카메라 전환기능 구현~~ 점수처리(발판 밟을 시 점수) 전송 오류 수정		~26 까지 각 팀원들 구현한 내용들 Merge, 충돌내용 검토

28	29	30				
본인이 구현한 내용 잘 돌아가는지 확인, 버그수정		~~팀원 만나서 게임 돌아가는지테스트~~				
12월						
일	월	화	수	목	금	토
			1	2	3	4
			~~최종 병합 및 테스트~~ 병합, 충돌 내용 검토			버그 테스트
5	6	7	8	9	10	11
버그 테스트	~~팀원 구현 내용 최종 테스트~~	~~프로젝트 제출물 정리~~ remote 테스트				

-이양희

11월						
일	월	화	수	목	금	토
	1	2	3	4	5	6
7	8	9	10	11	12	13

				Git, 프레임워크		IsOkGameStart(), FootHoldInit(), PlayerInit()
14	15	16	17	18	19	20
ServerInit()	전송 구조체 통일 (Send Game Data는 한번만 송/수신)	~~*클라이언트가 데이터를 받아 게임을 초기화하고 시작하는 부분 구현 IsPlayingGame*~~	화면전환 구현 Print_GameState(), CurrentGameState	화면 전환에 따라 키 입력 다르게 하도록 처리 Keyboard(),KeyboardUp() 입장한 플레이어 수 수신 (완성X)	플레이어 수 수신 오류수정	~~병합 후 충돌 내용 확인, 오류가 있다면 파악 및 수정~~
21	22	23	24	25	26	27
플레이어 수 수신 오류해결, 병합 후 충돌 내용 확인 및 오류파악	~~서버 시간,~~ **~21 병합 충돌내용 해결**		서버 시간 (미완)	~~화면 전환 구조 구현 마무리~~ ~~CreateMainGameScene()~~		~~병합 후 테스트, 오류가 있다면 파악 및 수정~~
28	29	30				
~~구현 내용 검토~~		~~팀원 만나서 게임 돌아가는지 테스트~~				

12월						

일	월	화	수	목	금	토
			1	2	3	4
			병합 테스트			테스트 및 오류 수정 서버 시간
5	6	7	8	9	10	11
테스트 및 오류 수정		~~최종~~ 정리 remote 테스트 버그수정			서버 시간 병합	코드 정리 게임오버 수정
12	13					
	게임 종료 처리, 코드 정리					