네트워크 게임 프로그래밍
추진 계획서

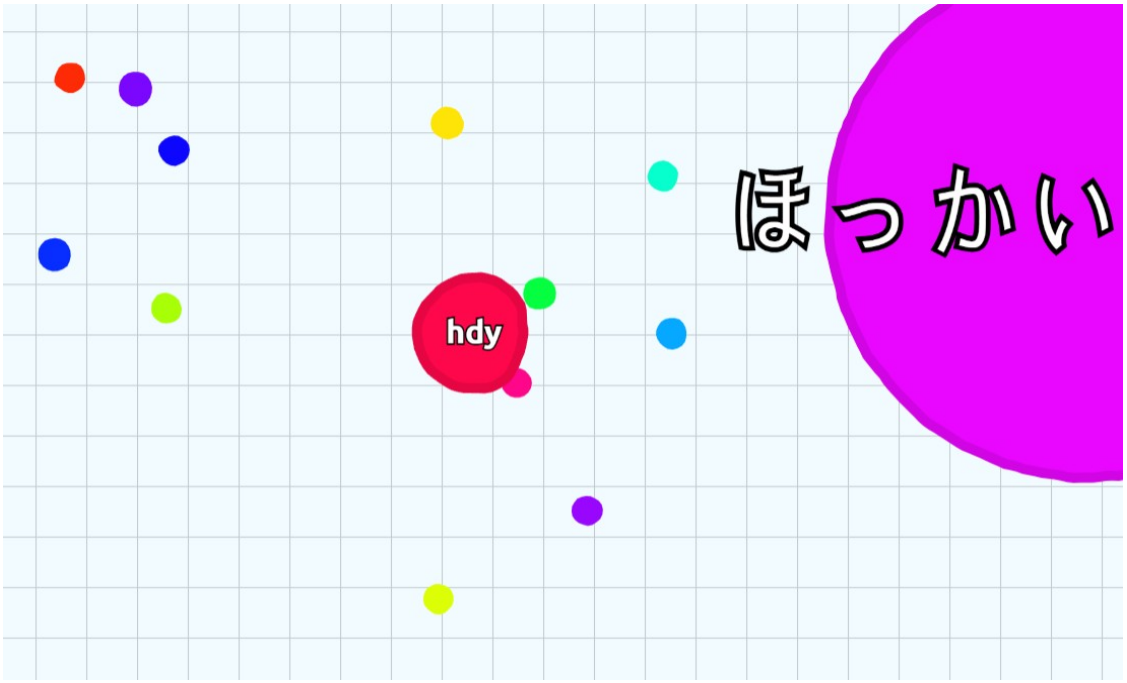
2020180044 현대윤

2020182013 김형일

목 차

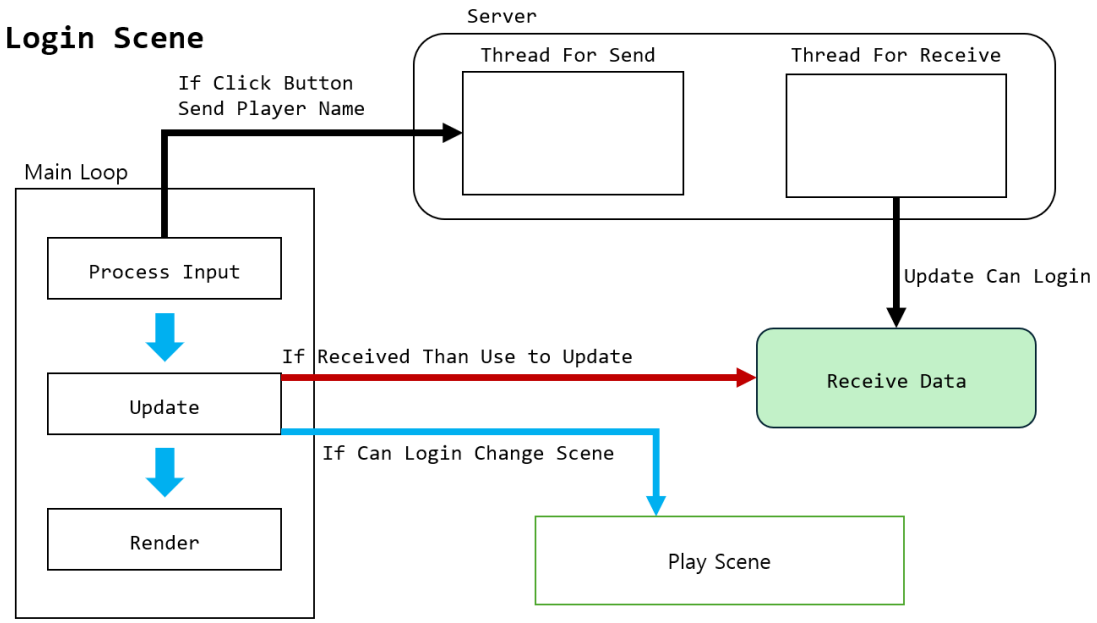
기획

현대윤이 윈도우 프로그래밍 중간 과제로 사용한 게임을 김형일이 SFML 로 재구성
플레이어는 자신보다 큰 다른 플레이어를 피하고 작은 애들은 먹으며 성장하는 게임
하나의 월드만 있고 새로 접속하는 플레이어는 초기 상태로 들어간다.
2 인용 플레이 게임으로 진행되고 마우스 클릭을 하면 해당 위치로 이동하게 된다.

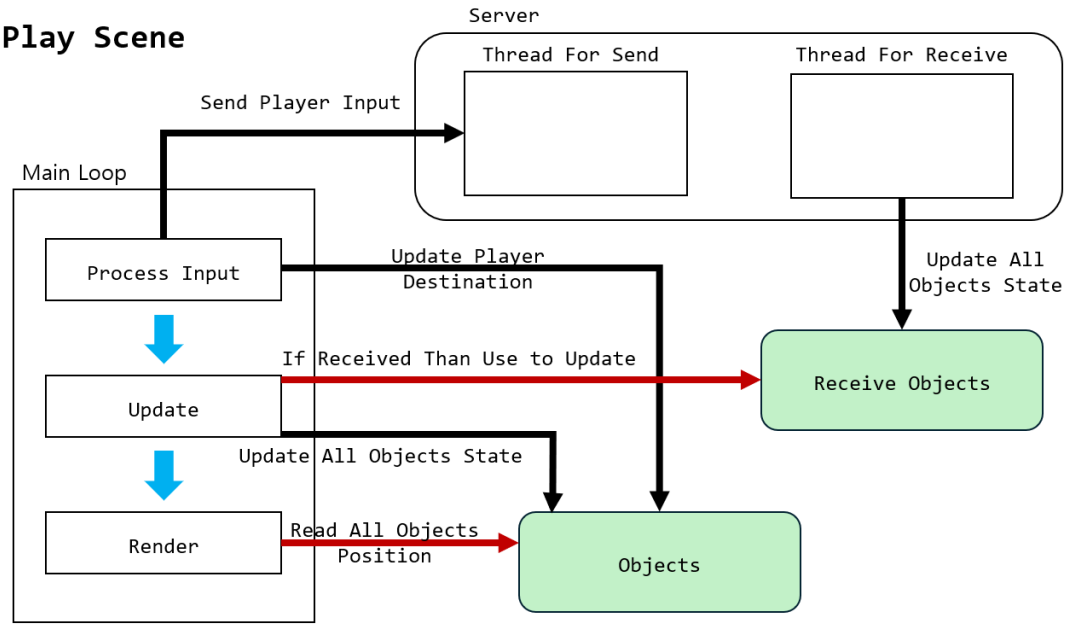


하이 레벨 디자인

◆ 클라이언트()

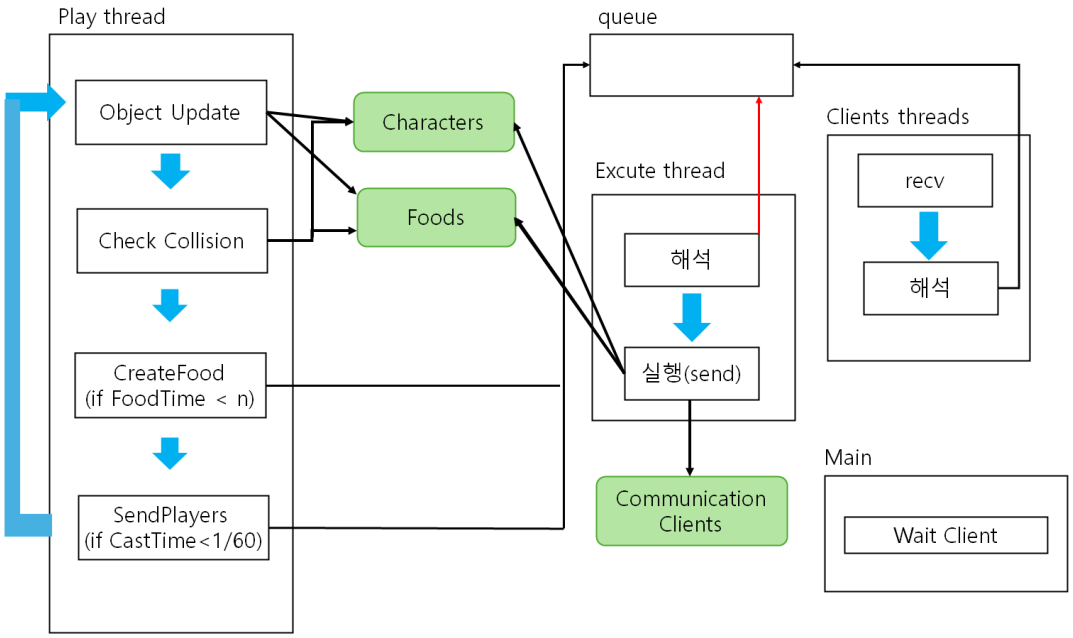


로그인 씬 필요성 미비하다고 판단, 콘솔창으로 대체



하이 레벨 디자인

◆ 서버



Command

Int type

Variant content

COMMAND_TYPE

LOGIN_TRY
INPUT
PLAYER_RESTART
CREATE_FOOD
CREATE_NPC
BROADCAST
LOGOUT

LOGIN_TRY

SOCKET sock
Char name[16]

PLAYER_RESTART

SOCKET sock

Int id

INPUT

POINT distance

LOGOUT

SOCKET sock

Int id

로우 레벨 디자인

◆ 패킷

Common

COLOR

```
Red, blue, green, etc…
```

PACKET_TYPE

```
LOGIN_TRY
LOGIN_FAIL
LOGIN_SUCCESS
APPEND_PLAYER
```

LOGIN_TRY

```
char name[16]
```

LOGIN_FAIL

```
char reason[64]
```

APPEND_PLAYER

```
char name[16]
POINT position
int color
```

FOOD_DATA

```
Int id
POINT position
Int color
```

LOGIN_SUCCESS

```
vector<PLAYER_DATA> player
vector<FOOD_DATA> food
```

PLAYER_DATA

```
int id
char name[16]
PINT position
POINT dest
Int size
int speed
int color
```

Common

PACKET_TYPE

```
INPUT
RECREATE_FOOD
RECREATE_PLAYER
SEND_PLAYERS
LOGOUT
```

RECREATE_FOOD

```
vector<FOOD_DATA> food
```

SEND_PLAYERS

```
Vector<PLAYER_DATA> player
Vector<int> id
```

INPUT

```
Char keycode
POINT distance
```

FOOD_DATA

```
Int id
POINT position
Int color
```

LOGOUT

```
Int id
```

PLAYER_DATA

```
int id
char name[16]
PINT position
POINT dest
Int size
int speed
int color
```

RECREATE_PLAYER

```
Int id
POINT position
Int colot
```

(12.02. 추가)

RECEATE_PLAYER 는 서버에게 보내는 것은 다시 살아날 플레이어의 **id** 보내기

클라이언트에게 보내는 것은 위와 같음

로우 레벨 디자인

◆ 오브젝트

```
class Entity {
public:
    virtual void ProcessInput(std::vector<PACKET_DATA>& recved); // 입력 처리 함수 for(data : recved) data type보고 그에 맞게 처리
    virtual void Update(float deltaTime) {} // 업데이트 함수 Client의 경우 서버의 Update를 예측하는 의미
    virtual void Render(); // 반지름이 size인 원을 position에 색은 color로 그림
    virtual void OnCollision(const Entity* collision) {} // 충돌 처리 함수
private:
    int id; // Scene에 중복되지 않는 유일한 값
    sf::Vector2f position; // World 좌표계
    int size; // 음수면 비활성화되었다는 의미
    int color; // Common에 있는 color의 인덱스
};
```

```
class Player : public Entity { // shape = CIRCLE
public:
    virtual void ProcessInput(std::vector<PACKET_DATA>&); // Entity::ProcessInput + if Input == L_BUTTON destination 업데이트 SEND
    virtual void Update(float deltaTime); // destination - position 방향으로 speed * deltaTime 만큼 position 변경
    virtual void Render(); // Entity::Render + 중앙에 이름을 그림
    virtual void OnCollision(const Entity* collision); // 먹이와 충돌 시 size 증가, 다른 플레이어와 충돌 시 size 비교 후 작다면 size = -1 크다면 size 증가
private:
    char name[16];
    int speed;
    sf::Vector2f destination; // World 좌표계
};
```

```
class Food : public Entity { // shape = CIRCLE
public:
    static constexpr float recreateTime; // 재성성에 필요한 시간

    virtual void Update(float deltaTime); // 비활성화된 상태라면 deathTime += deltaTime
    virtual void Render(); // Entity::Render
    virtual void OnCollision(const Entity* collision); // 플레이어와 충돌 시 사망

    bool canRecreate() const { return deathTime >= recreateTime; }
    void Recreate(sf::Vector2f newPos) // position = newPos, deathTime = 0;
private:
    float deathTime; // 사망 이후 시간
};
```

◆ 게임

```
struct Game {
private:
    Game(); // 생성자 호출 불가
    ~Game(); // DeleteCriticalSection
public:
    static Game& Instance(); // 싱글톤 패턴을 위한 instance

    void Init(const std::string& serverIP, short serverPort); // server socket 생성 및 connect, std::thread(RECVEIVE), InitializeCriticalSection
    void Run(); // timer를 이용해 지역 변수 float deltaTime 기록

    void Receive(); // while(true) recv(server, type) recv(server, context); lock() recvData.push_back() unlock()

    template<class T>
    typename std::enable_if<std::is_base_of< Scene,T>::value, void> ChangeScene(void* context) // waitScene을 std::shared_ptr<T>로 바꿈

    void ProcessInput(); // events.clear 후 들어온 sf::Event를 저장 waitScene != nullptr scene = std::move(waitScene);
    void Update(); // scene에 있는 Entity들 Update(지역 변수 deltaTime) 호출
    void Render(); // scene에 있는 Entity들 Render 호출

    SOCKET server;
    std::unique_ptr<Scene> scene;
    std::unique_ptr<Scene> waitScene = nullptr;
    std::vector<sf::Event> events;

    std::vector<PACKET_DATA> recvData;

    CRITICAL_SECTION cs;

    sf::Clock timer;
    sf::View camera; // Player 위치를 중심으로 두는 카메라
};
```

◆ 쓰레드 함수

```
void SEND(SOCKET& sock, const PACKET_DATA& packet); // send(sock, &packet.type, sizeof(int),0) send(sock, packet.context, size는 타입에 따라 다르게)
void RECVEIVE() { Game::Instance().Receive(); }
```

(11.14. 추가) 통신 전용 클래스를 만들도록 변경, Send 와 Recv 가 쓰레드 함수

```
class ClientNetworkManager {
public:
    ClientNetworkManager();
    ~ClientNetworkManager();

    void AddPacket(PACKET packet);
    std::vector<PACKET> GetPacket();

    void Send();
    void Recv();
private:
    SOCKET sock;
    std::queue<PACKET> sendQueue;
    std::list<PACKET> recvPacket;

    std::condition_variable cv;
    std::mutex sendMtx;
    std::mutex recvMtx;
};
```

(11.20. 추가) 서버측 쓰레드 함수는 서버 프레임 워크로 대체 프레임 워크는 뒷장에 첨부

```
void playThread(std::vector<Character>& characters, std::vector<Food>& foods, std::vector<Player>& players) {

    while (running) {
        //시간을 재서 해당시간이 되면 해당하는 각 함수를 실행 foodtime casetime
        objectUpdate(characters, foods);
        checkCollision(characters, foods);
        if(foodtime<n)
            createFood();
        if(casetime<1)
            sendPlayers();
    }
}

void clientsThread() {
    while (running) {
        //받으면 어떤 패킷인지 보고 이름과 정보를 큐에 넘겨준다
    }
}

void executeThread() {
    while (running) {
        //큐에 있는 데이터를 하나씩 꺼내서 이름을 보고 각 이름에 대해
        // 어떤 패킷을 보내줄지 결정한다
        // 즉, send를 해주는 쓰레드이다
    }
}
```

◆ 씬

```
class Scene {
public:
    Scene(void* context = nullptr); // if context context를 사용해서 초기화

    void ProcessInput(); // vector<PACKET_DATA> recved lock() while(Game::redvData != empty)
    //recved.push_back(Game::redvData.front) Game::redvData.dequeue unlock() entities.ProcessInput(recved)
    void Update(); // entities.Update()
    void Render(); // entities.Render()

    std::vector<std::shared_ptr<Entity*>> entities;
};
```

```
class LoginScene: public Scene {
public:
    void ProcessInput(); // Scene::ProcessInput + if if Input enter SEND
    void Update(); // entities.Update()
    void Render(); // entities.Render()

    std::string userName;
};

class PlayScene {
public:
    PlayScene(void* context); // LOGIN_SUCCESS를 받아 entities 초기화

    void ProcessInput(); // Scene::ProcessInput
    void Update(); // entities.Update()
    void Render(); // entities.Render()

    std::unique_ptr<PoolManager> poolManager;
};
```

12.02. 로그인 씬 구현 필요성 부족하다 판단 콘솔창 입력으로 대체

◆ 풀 매니저

```
class PoolManager {
public:
    template<class T>
    std::shared_ptr<T> Get(); // pools[typeid(T).name()] 중 활성화 되지 않은 shadred_ptr<T>를 반환

    template<class T>
    void Add(std::shared_ptr<Entity> entity); // pools[typeid(T).name()]에 entity를 추가
private:
    std::unordered_map<std::string, std::vector<std::shared_ptr<Entity>>> pools;
};
```

11.27. 필요성 부족하다고 판단하여 단순 함수로 구현

```
for (const auto& food : foods) {
    if (!food->active) {
        i++;
        food->Reset();
        data.emplace_back(*food);
    }
    if (i >= maxReCnt) {
        break;
    }
}
```

◆ 사용자 정의 함수

```
// 엔디안 변환 함수
template<typename T>  <T> IntelliSense에 대한 샘플 템플릿 인수를 제공합니다.
T swapEndian(T value) {
    //입력 값을 바이트 배열로 변환하여 source에 저장합니다.
    //source의 바이트 배열을 역순으로 dest에 저장합니다.
    return dest.value;
}


void sendPlayers() {
  //큐에 send_Players 보낸다
}
```

◆ 서버 프레임 워크(11.20. 추가)

```
public:
    Server();
    ~Server();

    void Run();

private:
    void AcceptClient();
    void ProcessClient(SOCKET client_sock);
    void Excute();

    void CheckCollision();
    void Update(double deltaTime);

private:
    SOCKET listen_sock;

    std::array<std::mutex, MUTEX_CATEGORY::SIZE> mutexes;

    std::condition_variable cv;

    std::queue<std::unique_ptr<Command>> excuteQueue;
    std::list<SOCKET> clients;

    std::list<std::unique_ptr<Player>> players;
    std::vector<std::unique_ptr<Food>> foods;

    float recreateDeltaTime = 0;
};
```

동기화

ObjectUpdate, Excute 간 Character, Food 컨테이너에 동시에 접근할 때

CheckCollison, Excute 간 Character, Food 컨테이너에 동시에 접근할 때

->임계영역으로 처리->11.20. mutex 로 처리

(11.20. 추가) condition_variable 을 이용해 excute 쓰레드 실행 순서 동기화

역할 분담

현대윤 서버 프레임워크, 서버가 클라와 통신하는 쓰레드, Common, 병행 큐, 바이트 정렬함수

김형일 게임 프레임워크, 게임 오브젝트, Common, 클라가 서버와 통신 쓰레드, collision, polling, **(11.20. 추가) 서버 프레임 워크**

개발 환경

Visual Studio

SFML

GitHub

개발 일정

김형일

일	월	화	수	목	금	토
					1	2
3	4	5 Common.h Entity Base-Render PlayScene -Render	6 MovingObj -Update Camera -Follow	7 Handle Input -screen to world MovingObj -SetDest	8 Camera -Clamp PlayScene -Update	9
10 PlayScene Init Use Created recv data	11 recv PACKET_TYPE::LOGIN_SUCCESS	12 **PlayScene Init Use recv data(11.20.)**	13 send PACKET_TYPE::INPUT	14 PlayScene -ProcessInput PACKET_TYPE::INPUT **NetworkManager**	15 **Server framework: Excute, Command Client**	16 recv PACKET_TYPE::BROADCAST
17 PlayScene -Update use PACKET_TYPE::BROADCAST	18 **recv PACKET_TYPE:: RECREATE_FOOD(11.27.)**	19 **PlayScene -Update use recv data (11.20.)**	20 PlayScene Player Die, send PACKET_TYPE::RESTART **Server: LoginSuccess PlayerAppend PlayerInput Logout PlayScene -Update, Init use recv data**	21 **recv PACKET_TYPE:: RECREATE_PLAYER(12.02)**	22 PlayScene Restart	23 **Entity: Food PACKET: FoodInfo**
24 **recv PACKET_TYPE::PLAYER_APPEND (11.20.)**	25 **Server: LoginSucess Food**	26 **recv PACKET_TYPE:: RECREATE_PLAYER other player(12.02) PACKET::RECREATE FOOD**	27 **recv PACKET::RECREATE FOOD**	28 **PlayScene Exit send PACKET::LOGOUT(11.20.)**	29 **recv PACKET::LOGOUT other player(11.20)**	30
1 **recv PACKET::LOGOUT(11.20.)**	2 **Server: RESTART_PLAYER Game: RESTART_PLAYE**	3 PlayScene Restart, Exit use button	4	5 LoginScene Render	6 LoginScene Make Button	7 LoginScene send PACKET_TYPE::LOGIN_TRY

	R					
8	9 recv LOGIN_FAIL LOGIN_SUCCESS	10 SceneChange	11	12	13	14

개발 일정

현대윤

일	월	화	수	목	금	토
					1	2
3	4	5 MainThread ClientThreads- recv,해석	6 큐-다중생산자, 단 일 소 비 자 병행큐 PlayThread	7 ExcuteThread- INPUT	8 CreateFood ExcuteThread- CREATE_FOOD	9 SendPlayers ExcuteThread- SEND_PLAYERS
10 패 킷 구 조 체 바 이 트 정 렬 해주는 함수	11 send- LOGIN_SUCCES S	12 recv-INPUT	13	14 충돌체크 함수	15 ObjectUpdate,E xcute 간 컨테이너 접근 락처리	16
17 CheckCollison, Excute 간 컨테이너 접근 락처리	18 send- 로 그 인 패킷 제외	19	20 ExcuteThread - PLAYER_RESTA RT	21	22 Communication Clients	23
24 ExcuteThread - PLAYER_APPEN D	25	26 ExcuteThread- PLAYER_RESTA RT 다수 클라	27	28 ExcuteThread- LOGOUT	29	30 Handle Client Crash -send LOGOUT
1	2 다수 클라 recv - 로 그 인 패 킷 제외	3 다수 클라 send - 로 그 인 패 킷 제외	4	5	6 ExcuteThread- LOGIN_TRY	7 다수 클라 recv -LOGIN_TRY
8	9 send LOGIN_FAIL LOGIN_SUCCES S	10 ExcuteThread - PLAYER_APPEN D	11	12	13	14