**[2D Game] Crazy Arcade**

**포트폴리오: WinAPI를 이용한 2D 게임 프로젝트**

**크레이지 아케이드 모작**

**제작 기간: 45일**

**작 성 자: 오민규**

**< Contents >**

1. **프로젝트 소개 및 개요**
2. 프로젝트 소개
3. 프로젝트 주요 기술 및 세부 내용
4. **시퀀스 다이어그램**
5. **최종 결과**
6. **프로젝트 소개 및 개요**
7. **프로젝트 소개**

2D 게임 프로젝트로 넥슨사의 크레이지 아케이드를 모작하였다. **멀티플레이 게임**으로 **2인**이 동시에 플레이할 경우 게임이 진행되며, 물풍선을 이용하여 오브젝트를 파괴하여 아이템을 획득하고 물풍선을 맵에 배치하여 플레이어끼리 대전하는 게임이다. **프로그래밍 언어는 C++**을 사용하였고 **클라이언트는 WinAPI**, **클라이언트 네트워크는 AsyncSelect, 서버는 IOCP**를 기반으로 제작하였다. 게임은 총 **4개의 씬(인트로, 로그인, 로비, 게임)**으로 구성되어있다. 간단한 프로젝트 소개를 마치고 프로젝트에서 사용한 기술 및 세부 내용을 살펴보자.

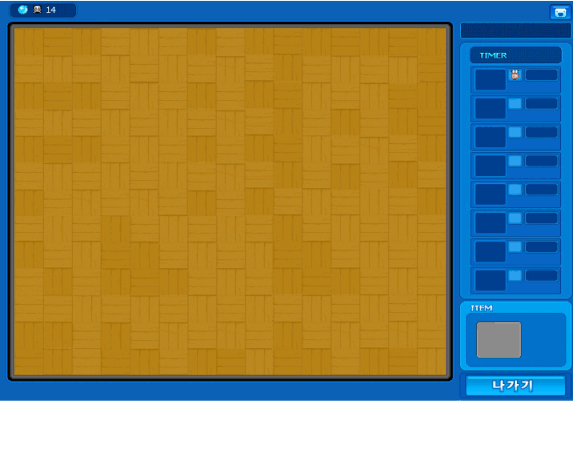
1. **프로젝트 주요 기술 및 세부 내용**

* **쿼드트리(QuadTree)**

쿼드트리란 **4진 트리**를 사용하여 맵을 분할하며 **각 분할된 영역을 노드(Node)**라고 한다. 각 노드를 재귀호출을 사용하여 분할하며 더 이상 분할할 수 없거나 분할 깊이를 정하여 한계까지 맵을 분할한다.

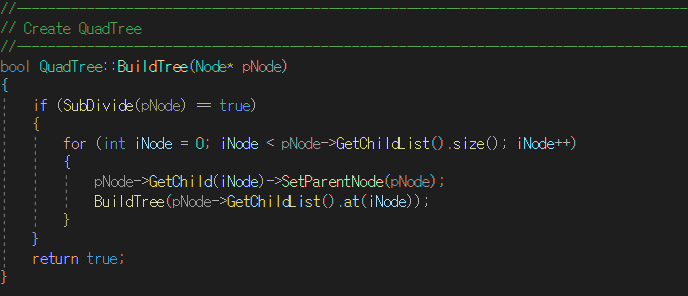
1. **맵(Map)**

**분할한 노드에 오브젝트**를 포함시켜 **충돌, 탐색 등**을 처리하기 위해 프로젝트에서 해당 기술을 사용하여 맵을 구축하였다. 다음은 쿼드트리를 이용하여 실제 맵을 분할하는 과정을 나타낸다.

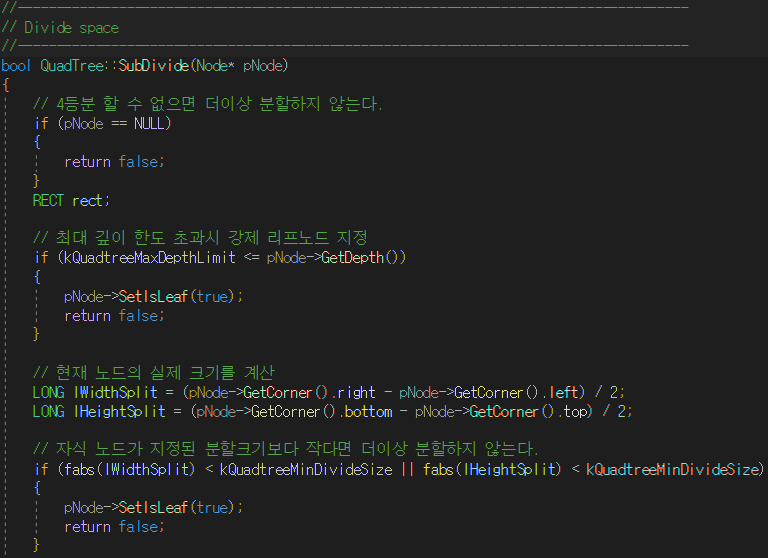


[그림 1-1] 쿼드트리 분할 과정

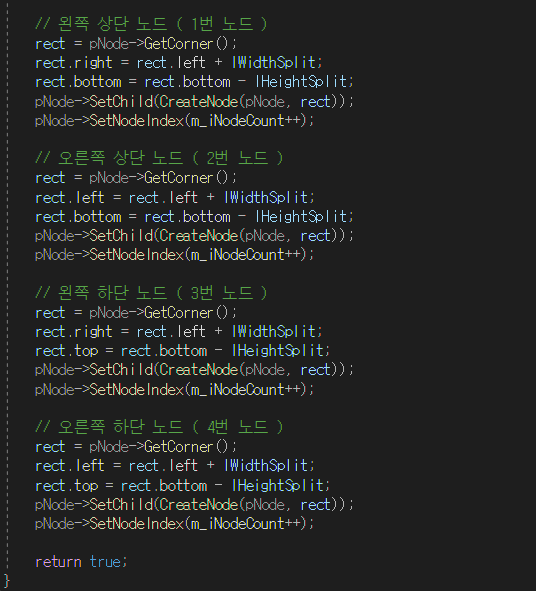
쿼드 트리를 생성할 때, 정해진 **깊이(Depth = 4)까지 분할**하여 쿼드 트리를 생성한다. 쿼드 트리를 생성과 분할에 대한 코드는 다음과 같다.



[그림 1-2] 쿼드트리 생성

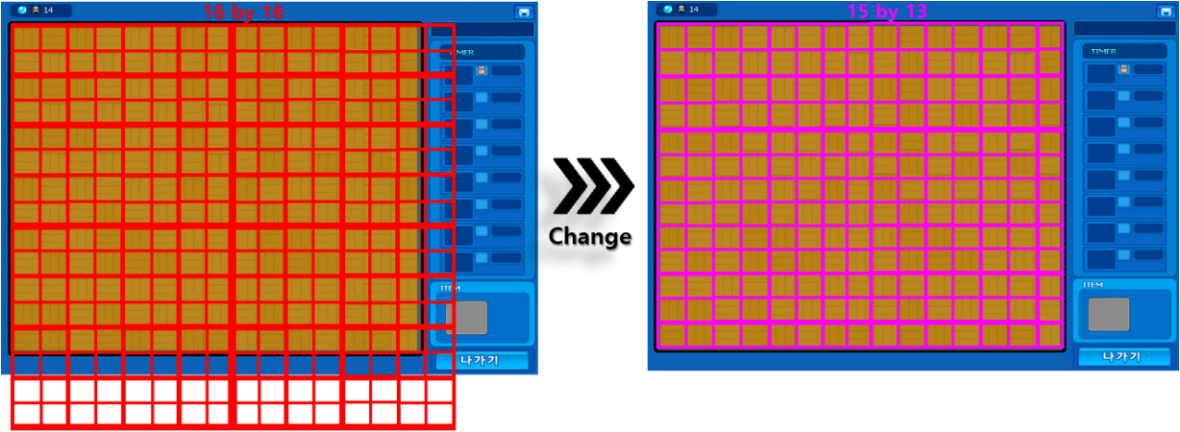


[그림 1-3] 쿼드트리 분할(1)

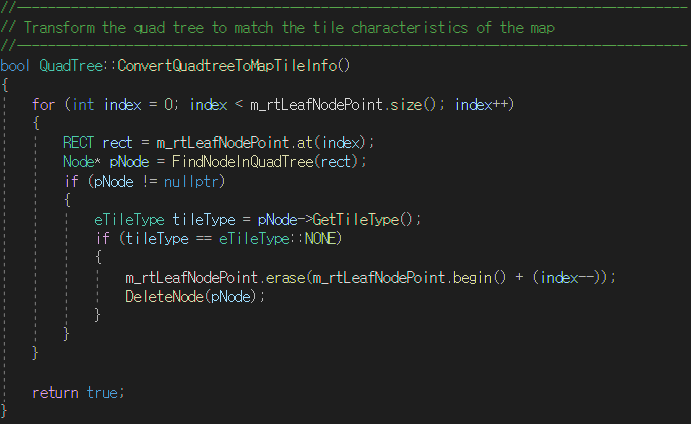


[그림 1-3] 쿼드트리 분할(2)

위의 코드를 통해 생성된 쿼드 트리의 리프 노드를 행렬로 나타내면**16 by 16 행렬**로 분리하였으며, **리프 노드(자식을 가지고 있지 않은 노드 = 깊이 4) 크기**는 **가로 40, 세로 40**으로 오브젝트의 크기와 같은 크기를 가지고 있다. 하지만 **실제 게임이 플레이되는 맵의 크기**는 **15 by 13**으로 쿼드트리의 모든 리프노드를 리스트에 저장하여 실제 **맵에 적용 되지 않는 리프 노드들을 삭제**한다.



[그림 1-4] 16 by 16 쿼드트리를 15 by 13 쿼드트리로 변경



[그림 1-5] 15 by 13으로 쿼드트리 변환 코드

1. **오브젝트(Object)**

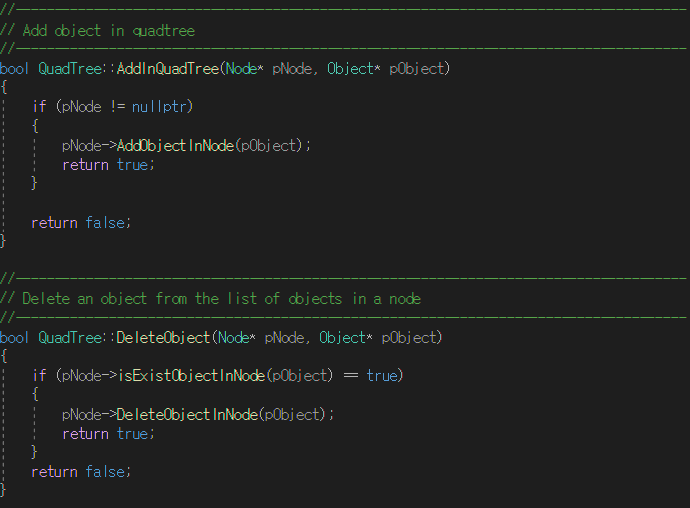
쿼드트리의 **노드**들은 오브젝트를 저장할 수 있는 **오브젝트 리스트**가 **존재**한다. 위치를 이용하여 **쿼드 트리 내의 오브젝트를 탐색**하며 **큐를 이용하여 찾는다.** 루트 노드부터 탐색을 시작하며 **4개의 자식 노드 중에서 전달받은 위치가 포함할 수 있는 자식 노드를 큐에 넣고 이전에 넣은 부모 노드를 빼고 같은 방법으로 탐색을 시작한다**. 적합한 위치에 들어갈 수 있는 노드를 찾는다면 해당 노드를 반환한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

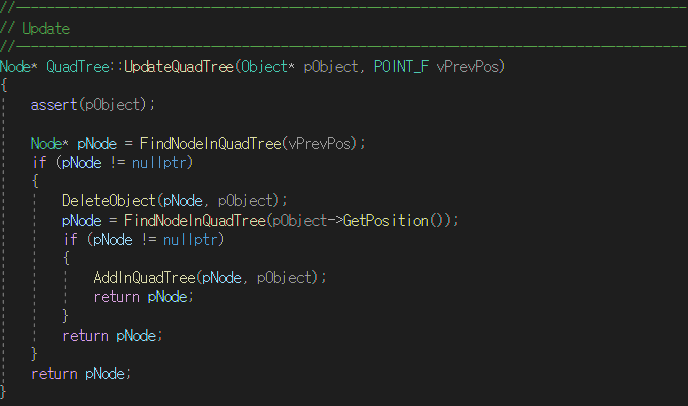
[그림 1-6] 쿼드트리 오브젝트 탐색

오브젝트가 포함할 수 있는 노드를 찾는다면 해당 노드에 오브젝트를 추가한다. 최초 맵을 생성할 때, 생성되는 **정적 오브젝트(박스, 아이템, 부서지지 않는 맵 오브젝트)는 생성과 동시에 노드에 추가**한다.



[그림 1-7] 쿼드트리에서 오브젝트 삽입/삭제

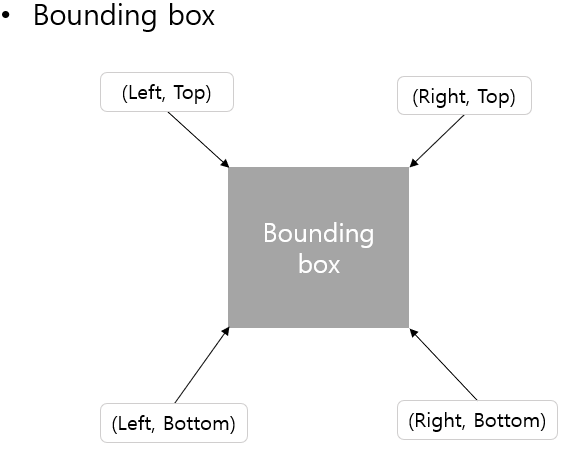
**동적 오브젝트(플레이어)**의 경우, 매 프레임 플레이어가 **이동**할 때, 쿼드 트리에서 **이전 노드와 현재 위치의 노드를 비교**하여 다를 경우 이전 노드를 찾아 오브젝트를 삭제하고 현재 위치에 해당하는 노드에 오브젝트를 추가한다.



[그림 1-8] 쿼드트리 갱신

* **충돌(Collision)**

오브젝트의 바운딩 박스 정보를 이용하여 충돌 처리를 한다. **바운딩 박스**는 **사각형**으로 구축하였다. 총 4개의 좌표 (x 좌표의 시작점, 끝점, y좌표의 시작점, 끝점)로 바운딩 박스가 구성된다.



[그림 1-9] 바운딩 박스

두 오브젝트 간 충돌이 발생하는 경우와 발생하지 않는 경우가 존재하는데 이에 대한 몇 가지 예시를 들어 보자.텍스트, 전자기기이(가) 표시된 사진

자동 생성된 설명

[그림 1-10] 두 바운딩 박스 충돌

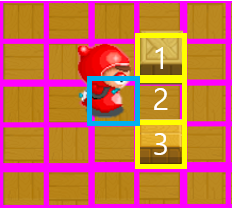
위 그림을 보면 충돌일 경우와 충돌하지 않았을 경우가 나타나 있다. 충돌 유무를 판단하려면 그림의 오른쪽에 있는 6개의 길이를 통해 판단이 가능하다.

텍스트이(가) 표시된 사진

자동 생성된 설명

[그림 1-11] 충돌 함수

아래의 그림은 게임에서 캐릭터가 이동하며 **오브젝트와 충돌 처리 방식**을 보여주고 있다. 캐릭터가 오른쪽으로 이동할 때, 캐릭터의 현재 노드(파란색 영역)로부터 1번 노드부터 충돌을 처리한다.



[그림 1-12] 캐릭터 오브젝트 충돌

여기서 한 가지 의문점이 있을 것이다. **“이동하는 방향의 한 개의 노드와 충돌 처리를 하면 되지 왜 3개의 노드와 충돌 처리를 하는가?”**라는 의문 것이다. 3개의 노드와 비교하는 이유는 동적 오브젝트들은 움직일 때마다 위치를 통해 노드를 갱신하는데 캐릭터가 아래 그림과 같이 블록을 통과하는 **시각적인 문제**가 **발생**한다.

패브릭이(가) 표시된 사진

자동 생성된 설명

[그림 1-13] 쿼드트리에서 캐릭터 이동시 오브젝트와 겹치는 현상

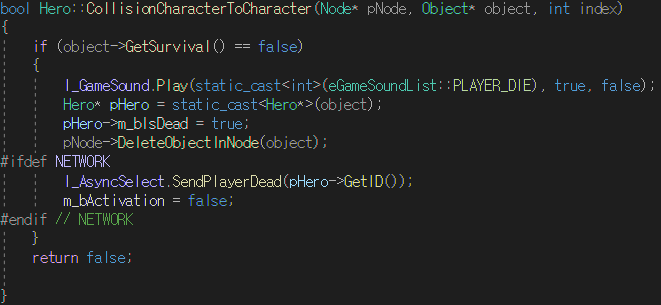
따라서 3개의 노드의 크기와 캐릭터의 바운딩 박스를 이용하여 **3개의 노드와 충돌 처리**한다.

1. **캐릭터 충돌**

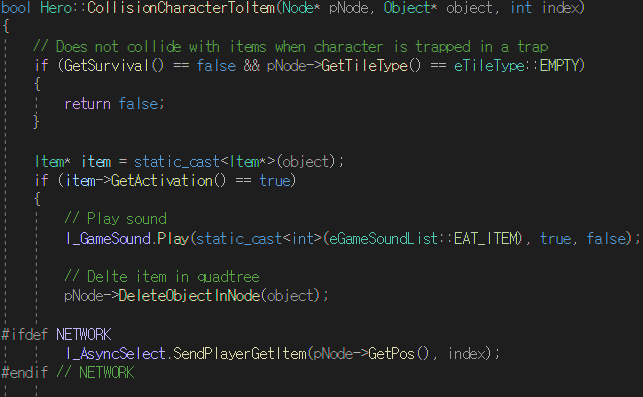
아래의 표에서는 오브젝트 타입에 따라 캐릭터와 충돌 처리하는 방법을 설명한다.

|  |  |
| --- | --- |
| **충돌** | **캐릭터** |
| **캐릭터** | 캐릭터와 캐릭터는 **플레이 도중에 충돌 처리를 하지 않는다.** 하지만 물풍선에 의해 **피격당한 플레이어가 Trap 상**태로 전환되었을 때, 상호 간 **충돌 처리를 진행**하게 된다. Trap 상태에서 충돌이 발생하게 된다면, 캐릭터는 Dead 상태로 변화하고 게임이 종료된다. |
| **맵 오브젝트** | 캐릭터와 정적 오브젝트 충돌은 캐릭터가 **이동한 위치의 노드를 탐색**하여 **오브젝트가 존재하는지 판단**한다. 오브젝트가 존재할 때, 해당 오브젝트가 **맵 오브젝트**라면 **캐릭터를 이동 전 위치로 이동**시킨다. |
| **아이템** | 아이템은 **박스와 물풍선 충돌**이 발생하였을 경우, **박스 위치에 아이템이 존재**한다면, **아이템이 활성화**된다. 충돌 방식은 정적 오브젝트와 같으며 아이템을 습득하였을 때, 아이템 특성에 맞게 캐릭터의 능력치가 상승한다. |

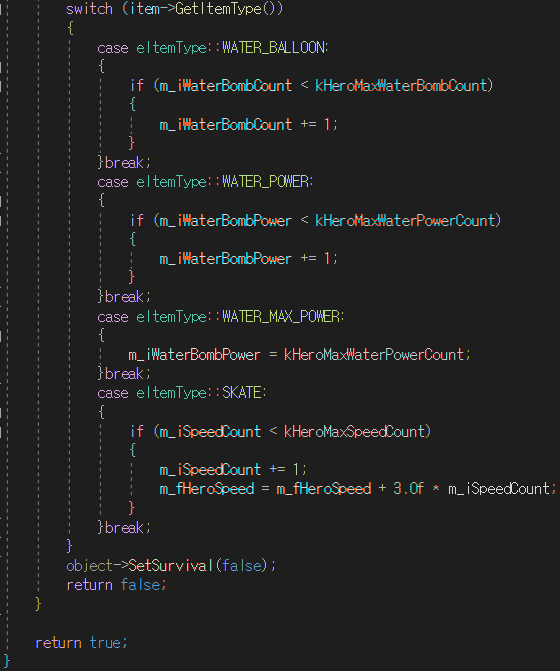
[표 1-1] 캐릭터와 오브젝트의 충돌처리



[그림 1-14] 캐릭터와 캐릭터 충돌



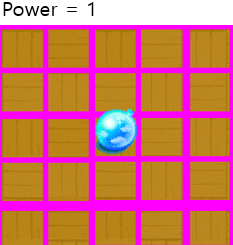
[그림 1-15] 캐릭터와 아이템 충돌(1)



[그림 1-16] 캐릭터와 아이템 충돌(2)

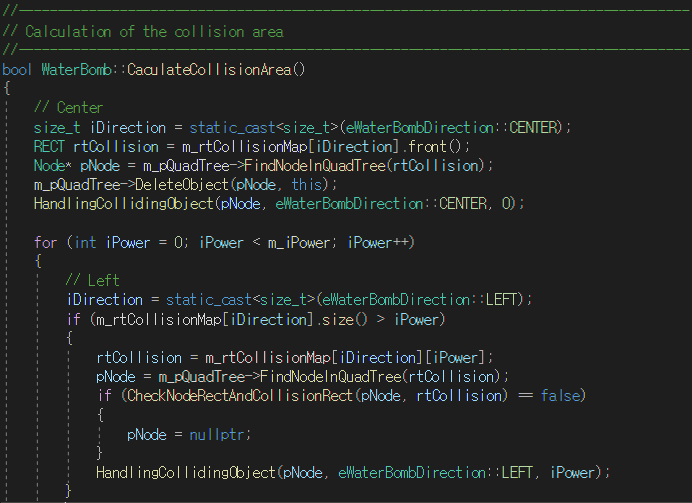
1. **물풍선과 오브젝트 충돌**

**물풍선이 폭발**할 때, 물풍선 위치에서 **5방향(중앙, 위, 아래, 오른쪽, 왼쪽)**으로 물줄기에 따라 **노드들을 탐색**한다**. (탐색 노드 수 = 중앙 + (물풍선 파워 \* 4))**

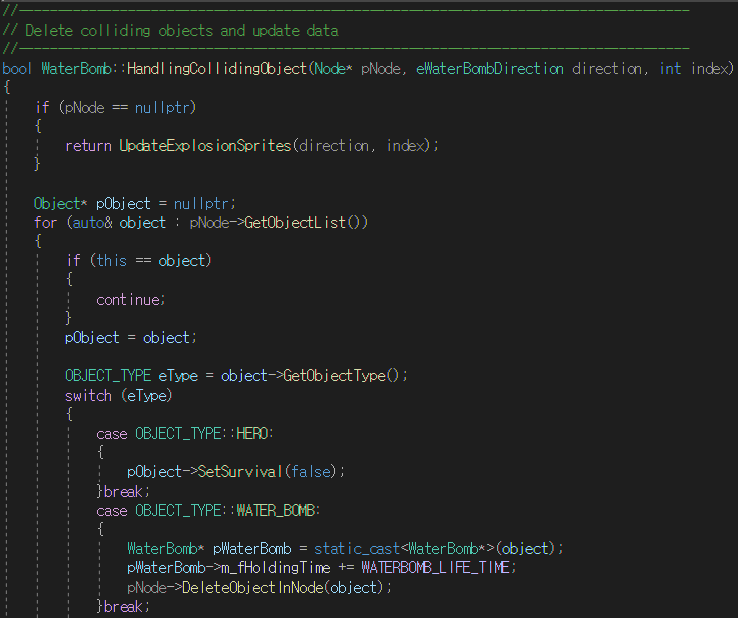
****

[그림 1-14] 물풍선 폭발 및 충돌 범위

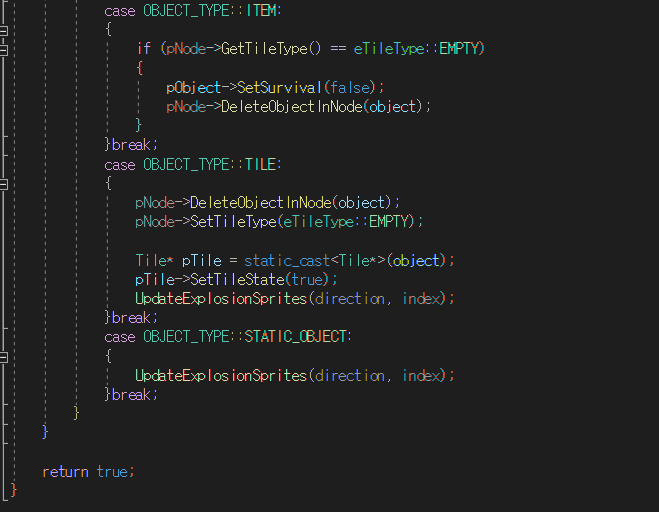
중앙을 제외한 4방향 노드는 같은 형태로 오브젝트와 충돌 판정을 하며, **충돌되었을 때, 오브젝트 타입에 따라서 충돌된 오브젝트를 처리**한다. 만약 **맵 오브젝트와 충돌하거나 맵의 끝에 물줄기가 닿는다면 물줄기의 끝을 표현하는 이미지를 렌더링**할 수 있게 적용한다.

****

[그림 1-15] 물풍선과 오브젝트 충돌 여부

****

[그림 1-16] 물풍선과 오브젝트 충돌 처리(1)

****

[그림 1-17] 물풍선과 오브젝트 충돌 처리(2)

* **싱글톤 패턴(Singleton pattern)**

싱글톤 패턴(Singleton pattern)이란 **메모리에 인스턴스 하나를 등록하여 여러 스레드에서 동시에 액세스가 가능하며 생성 디자인 패턴**이다. 프로젝트에서 인풋, 타이머, 각 씬, 매니저 클래스(비트맵, 스프라이트, 캐릭터, 오브젝트) 등 하나의 인스턴스만 필요하며 자주 참조되는 객체에서 사용하였다. 현재 **이른 초기화 (Eager initialization) 방법을 적용**하였으며, 이는 코어가 초기화하는 시점(Main의 Init() 함수)에서 인스턴스를 메모리에 등록한다. 하지만 추후 멀티스레드를 적용한다면, 이 방법에선 동기화 문제가 발생한다. 이때, 게으른 초기화(Lazy initialization) 방법을 적용하여 이 문제를 해결해야 한다. 게으른 초기화는 3가지 방법(Synchronized, Double Checking Locking, Lazy Holder)으로 구현이 가능하다.

|  |  |
| --- | --- |
| **방법** | **내용** |
| **Synchronized** | 함수에 synchronized 키워드를 사용하여 실행된 함수의 인스턴스는 하나의 스레드에서만 접근할 수 있도록 제한하여 동시성 문제를 해결할 수 있다. 하지만 인스턴스를 많이 가져오는 경우 성능 이슈가 발생한다. |
| **DCL**  **( Double Checking Locking )** | Synchronized 방법의 성능 이슈 단점을 보완한 방법으로 인스턴스가 null일 경우에만 Syncrhonized 블록에 접근할 수 있다. |
| **Lazy Holder** | Singleton 클래스 안에 클래스( private static class Lazyholder) 인스턴스를 final로 선언하는 방법을 이용하는 것이다. Finail로 선언한 인스턴스는 GetInstance() 호출 시, LazyHolder 클래스의 초기화가 이루어지면서 원장성이 보장된 상태로 단 한 번 생성된다. final 변수이므로 이후에 다시 인스턴스가 할당되는 것을 막을 수 있다. |

[표 1-2] 게으른 초기화 3가지 구현 방법

* **네트워크(Network)**

1. **서버(Server)**

**네트워크 서버** **IOCP를 이용하여 구현**하였다. IOCP는 Win32 커널 오브젝트의 하나로 입출력을 감시하여, 입출력에 대한 처리가 완료되었을 때, 단위 정보(완료 레코드)를 유지하여 프로그램이 처리할 수 있도록 완료 통지를 해주는 일련의 시스템이다. 현재 게임은 1vs1대전 시스템이므로 IOCP를 적용한다 해도 성능에 큰 차이는 없지만 추후 **플레이어의 수를 증가시킬 경우를 대비하여 IOCP를 채택**하였다. 서버는 StreamPacket, 유저 리스트, 뮤텍스, Acceptor 객체를 가지고 있다.

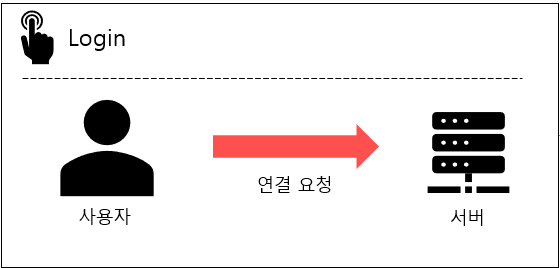
|  |  |
| --- | --- |
| **구분** | **내용** |
| **StreamPacket** | 스트림 패킷 객체는 **클라이언트로부터 받는 모든 패킷을 처리**하는 객체이다. **데이터의 전송량을 줄이기 위해 각 씬에 따른 패킷을 따로 처리**한다. 로비씬 패킷과, 게임씬 패킷 두 가지 패킷 리스트가 존재한다. |
| **UserList** | 현재 서버에 존재하는 유저를 저장하고 있으며, 이 리스트를 통해 서버에서 원하는 유저에게 데이터를 전송할 수 있다. |
| **Mutex** | **코드가 다중 실행되는 것을 방지**하기 위한 뮤텍스 객체이다. 서버에서 특정 처리(패킷 처리, 유저 이탈 등)를 하게 되는 경우, 중복되는 처리로 오류를 방지하기 위해 사용한다. |
| **Acceptor** | **플레이어가 서버에 접속하는지 감시**하는 객체이다. **감시하는** **스레드를 생성**하므로 **서버 프로세스가 실행 중에도 독립적으로 감시**할 수 있다. 유저가 추가되었을 때, **IOCP디바이스에 유저 소켓을 등록**하여 포트를 생성, 유저 데이터를 생성 및 서버에 등록하는 역할을 한다. |

[표 1-3] CrazyArcade서버의 주요 객체

서버 프로세스는 위의 객체들로 주축을 이루어 실행된다. 서버의 핵심 객체를 알아봤으니 다음은 서버에서 게임 데이터가 어떻게 상호작용하는지 알아보자.

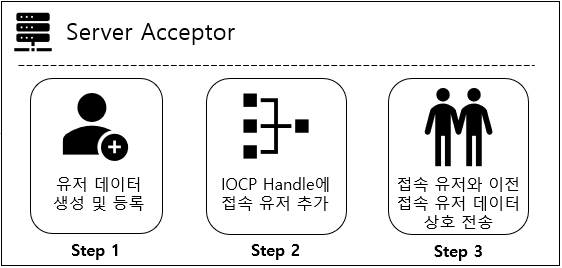
1. **연결**

클라이언트를 통해 로그인에 성공하면 서버로 연결 요청을 진행한다.



[그림 1-18] 서버 연결 요청

서버 프로세스가 문제없이 실행되고 있다면, 서버에서 Listen 상태로 감시 중인 Accept 객체에 의해 아래 그림과 같이 서버에 연결 과정을 거친다.



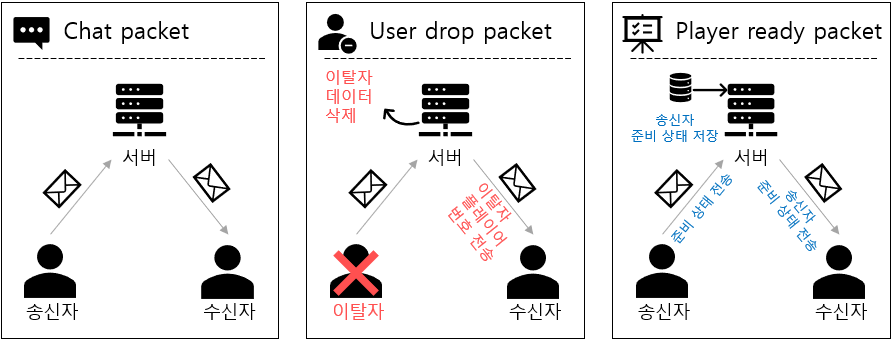
[그림 1-19] 서버 연결 과정

서버와 성공적으로 연결이 되었다면 **연결된 유저를 담당**하는 **스레드는 송수신 대기 상태**로 존재한다.

1. **로비**

로비 씬에서 서버는 **채팅 패킷, 플레이어 이탈 패킷, 플레이어 게임 준비 패킷 송수신과 게임 시작 패킷 전송, 모든 플레이어 게임 준비 상태 감시** 크게 5개의 작업을 처리하고 있다.

* **로비 패킷 송수신**



[그림 1-20] 로비 패킷 송수신 처리

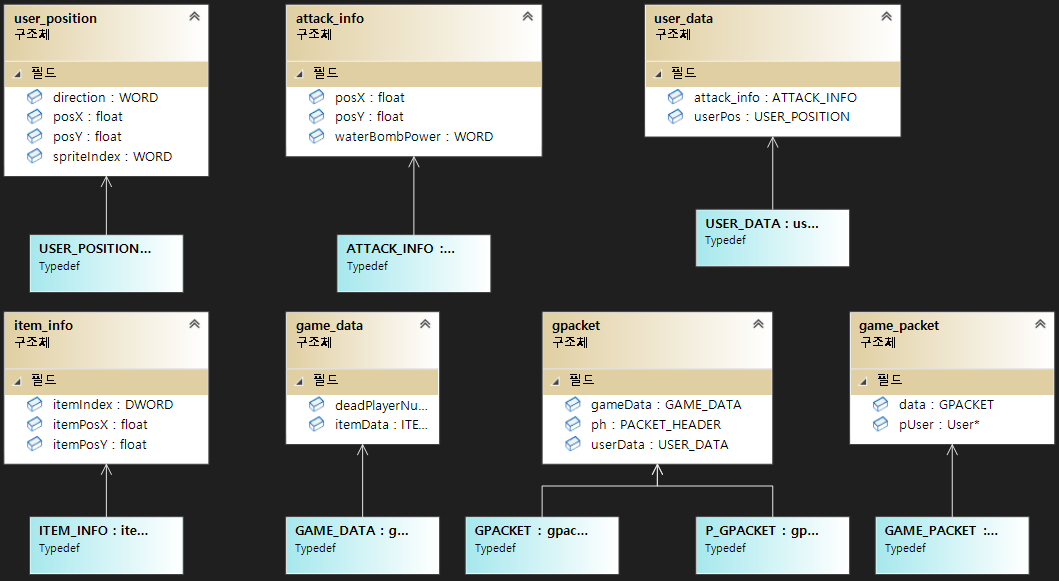
서버에서 **패킷 처리는 브로드캐스팅** 방식을 사용하여 **송신자가 보낸 패킷은 연결된 모든 수신자가 받는다.** 채팅 패킷을 처리하는 것과 달리 **유저 이탈 패킷이나 플레이어 준비 패킷** 같은 경우는 **서버 내부에서 데이터 갱신, 삭제**가 이루어진다.

* **플레이어 게임 준비 상태 감시 및 게임 시작 패킷 전송**

두 플레이어가 **모두 준비 상태**에 있다면 **게임이 시작**되어야 한다. 서버의 **유저 리스트에 등록된 유저 준비 정보를 실시간으로 검사**하여 두 플레이어가 모두 준비 상태에 있다면 **서버에서 각 클라이언트에게로 게임 시작 패킷을 전송**한다. 이때, 인 게임에서 **랜덤으로 스폰되는 캐릭터 위치, 아이템을 서버에서 생성**하고 통지하여 게임에서 **동일하게 생성**될 수 있도록 하였다.

1. **게임**

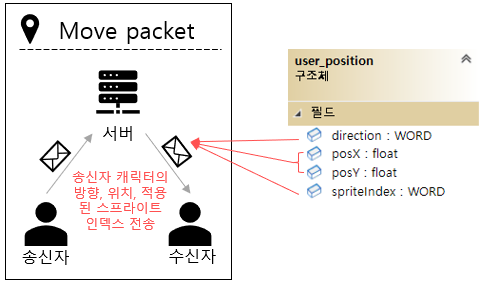
게임 씬에서 서버는 **캐릭터 이동 패킷, 공격 패킷, 아이템 획득 패킷, 게임 종료 패킷을 처리**한다. 모든 패킷은 게임 패킷으로 데이터를 패킹하여 전송한다



[그림 1-21] 게임 패킷

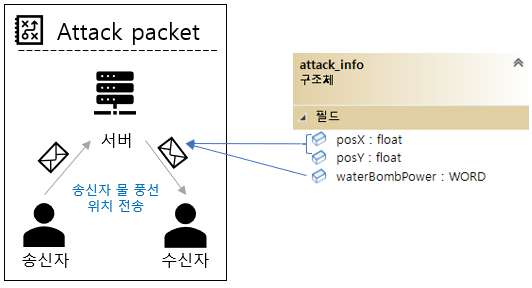
* **이동 및 공격 패킷**

이동과 공격 데이터는 게임 패킷의 userData 변수에 저장한다. 서버는 송신자를 제외한 플레이어에게 정보를 송신한다.



[그림 1-22] 이동 패킷 처리

**플레이어의 이동** 경우는 **패킷 송수신 빈도가 높기 때문에** **50프레임으로 고정**하여 패킷을 전송한다. 이동 시 **캐릭터의 애니메이션도 동기화**해주어야 하기 때문에 클라이언트에서 적용된 **스프라이트의 인덱스**를 함께 넘겨준다.

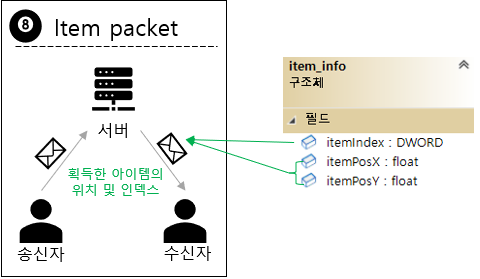


[그림 1-23] 공격 패킷 처리

**플레이어가 공격**할 경우, **물풍선의 위치와 물풍선 강도** 데이터를 넘겨준다. 서버에서 상대 플레이어에게 이 데이터를 송신하면 **상대방 클라이언트에서 물풍선을 생성**한다.

* **아이템 획특 패킷**

아이템에 따른 능력치는 캐릭터에 귀속되므로 상대방한테 전달할 필요는 없다. 따라서 **플레이어가 아이템과 충돌이 되어 획득하면, 양쪽 클라이언트에서 아이템이 제거**되어야 한다.



[그림 1-24] 아이템 획득 패킷 처리

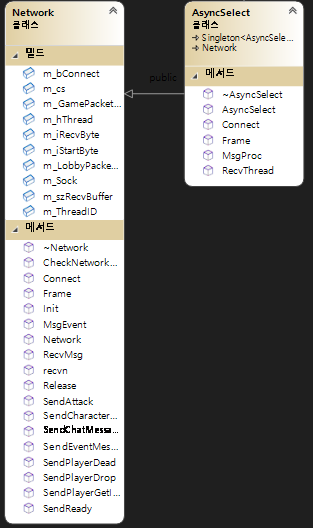
**상대방에게 전달해야 할 아이템 정보는 위치와 인덱스이다.** 아이템 패킷을 받은 클라이언트는 전달 받은 아이템 위치를 통해 노드를 찾아낸다. 이후 노드에 존재하는 오브젝트 리스트에서 아이템 인덱스를 통해 오브젝트를 찾아 송신자가 획득한 아이템을 삭제한다.

* **게임 종료 패킷**

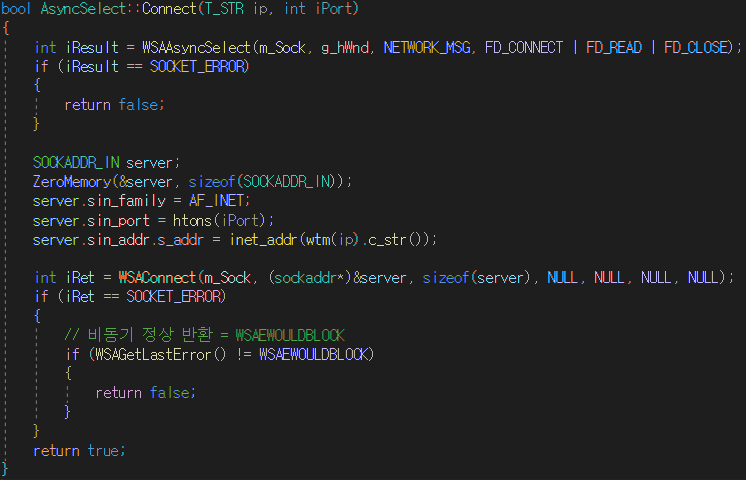
**특정 플레이어가 사망**할 경우, 게임이 종료되는데 사망했을 때, **다른 플레이어에게 통지**해주어야 한다. 사망한 플레이어 번호를 게임 패킷에 존재하는 **게임 데이터 패킷 변수 안에 deadPlayerNumber에 저장하여 상대방에게 전달**하고 이를 받은 클라이언트는 자신의 플레이어 번호와 전달 받은 번호를 통해 승패 처리를 한다.

1. **클라이언트(Client)**

클라이언트에서 **네트워크 처리는 AsyncSelect모델을 기반으로 제작**하였다. **AsyncSelect 모델은 윈도우 메시지를 통해 멀티스레드를 사용하지 않고 네트워크 이벤트 처리를 하며 비동기적으로 소켓을 활용** 수 있기 때문에 해당 모델을 채택하여 클라이언트에 적용하였다. 클라이언트에서 로그인 후 로비씬으로 전환하면서 서버에 연결을 요청한다.

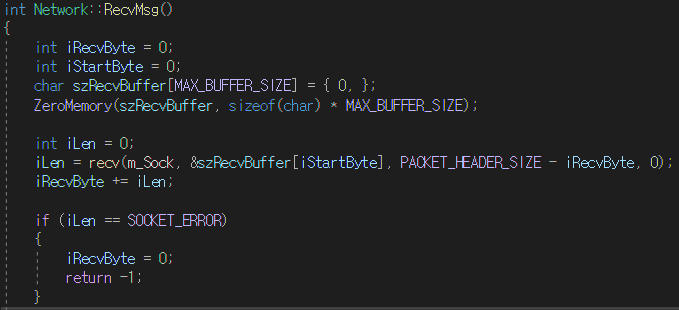


[그림 1-25] 클라이언트의 네트워크 클래스 다이어그램

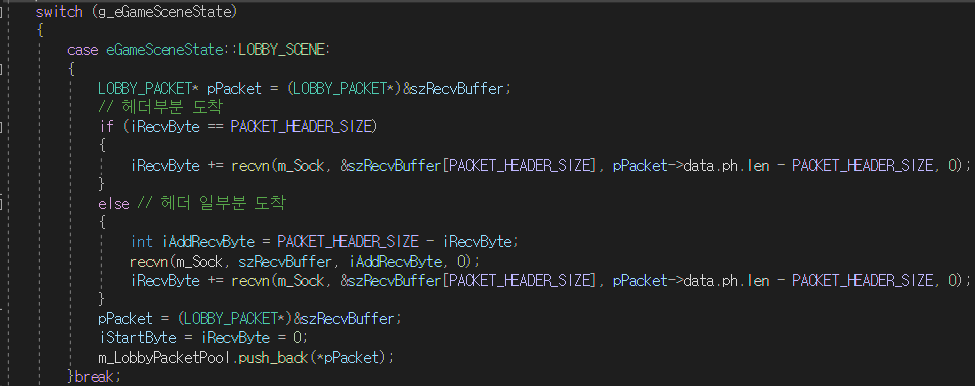


[그림 1-25] 클라이언트에서 서버 연결

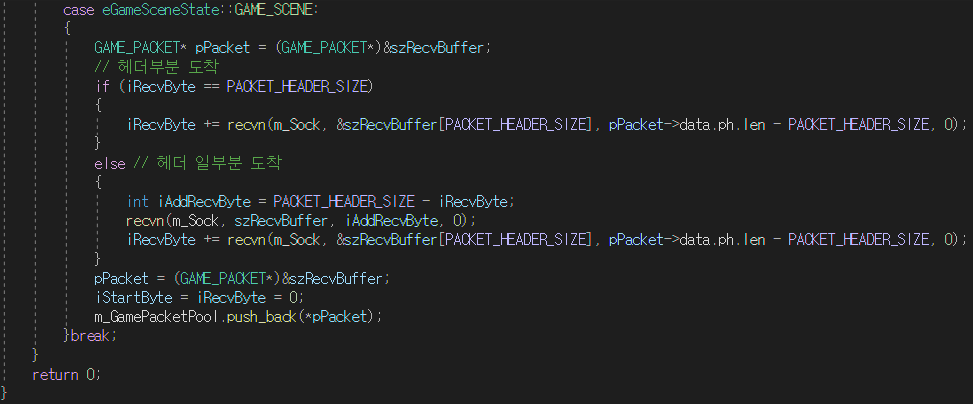
**서버로부터 패킷을 수신**하면 씬 상태에 따라 각 패킷 풀에 패킷을 넣어준다. 이후 각 씬 객체에서 프레임마다 실행되는 **NetworkProcess() 함수에서 패킷 풀에 있는 모든 패킷을 처리**하며 패킷 타입에 따라 알맞은 처리를 담당한다. **모든 처리가 종료되었을 때, 패킷 풀에 있는 모든 패킷을 삭제**한다.



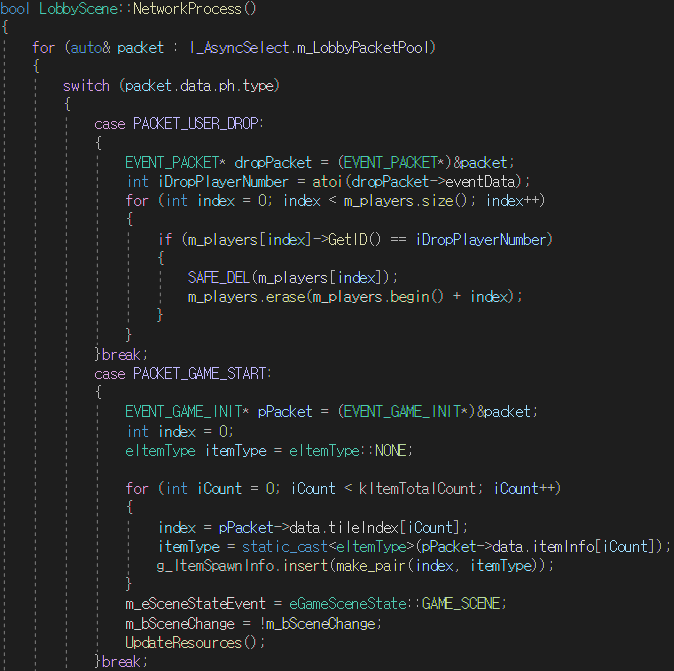
[그림 1-26] 패킷 처리 함수(1)



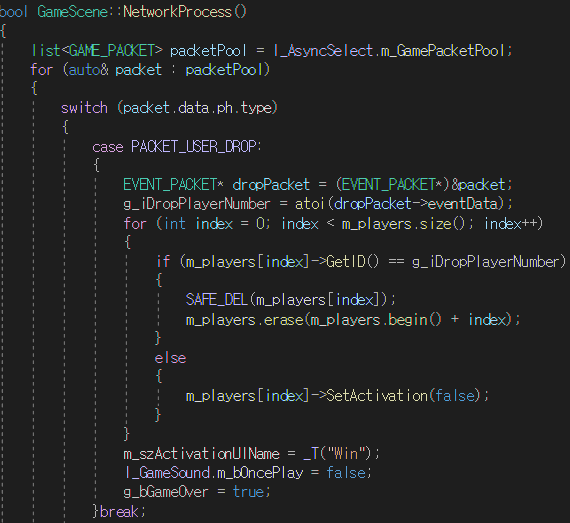
[그림 1-27] 패킷 처리 함수(2)



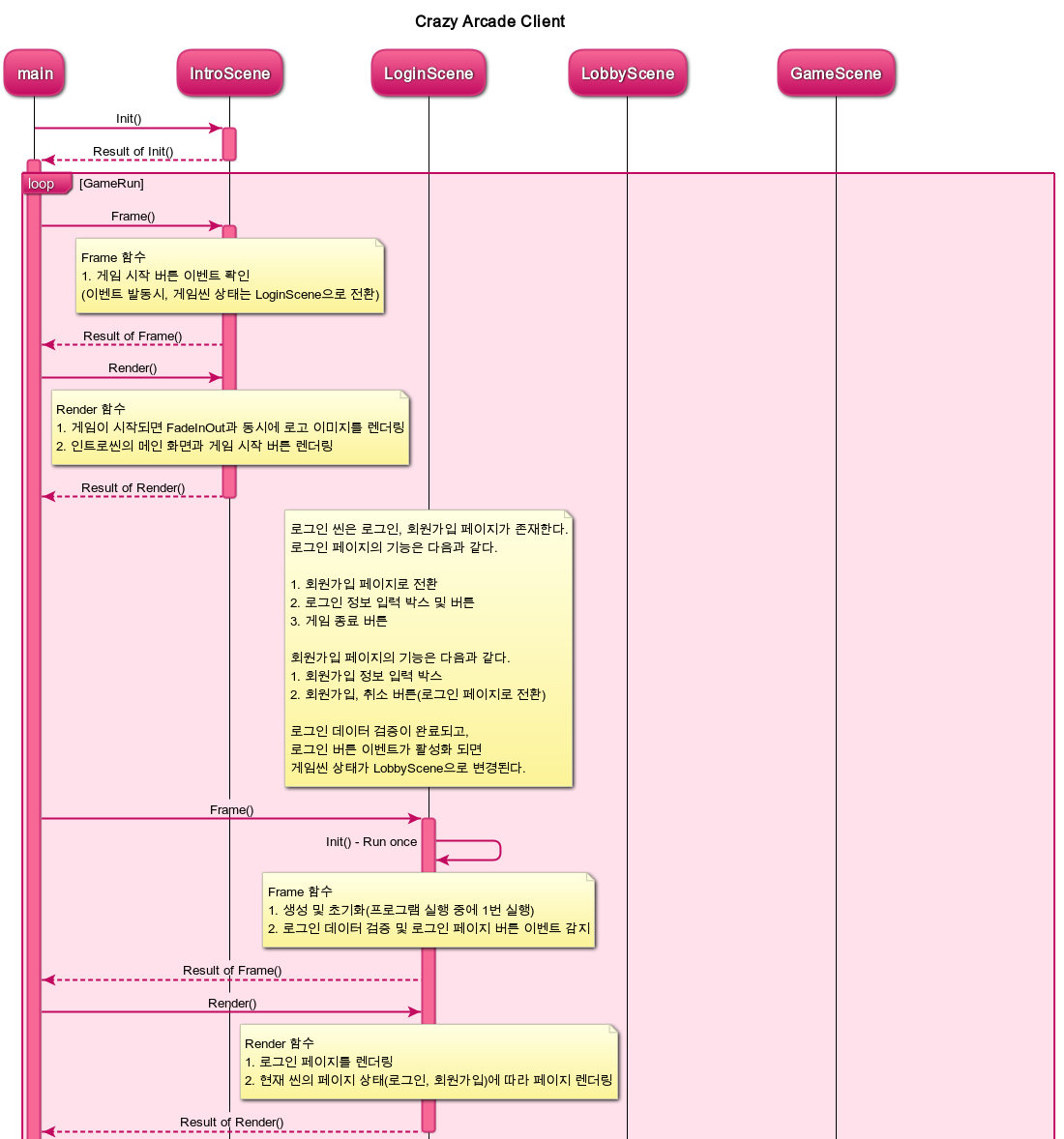
[그림 1-28] 패킷 처리 함수(3)



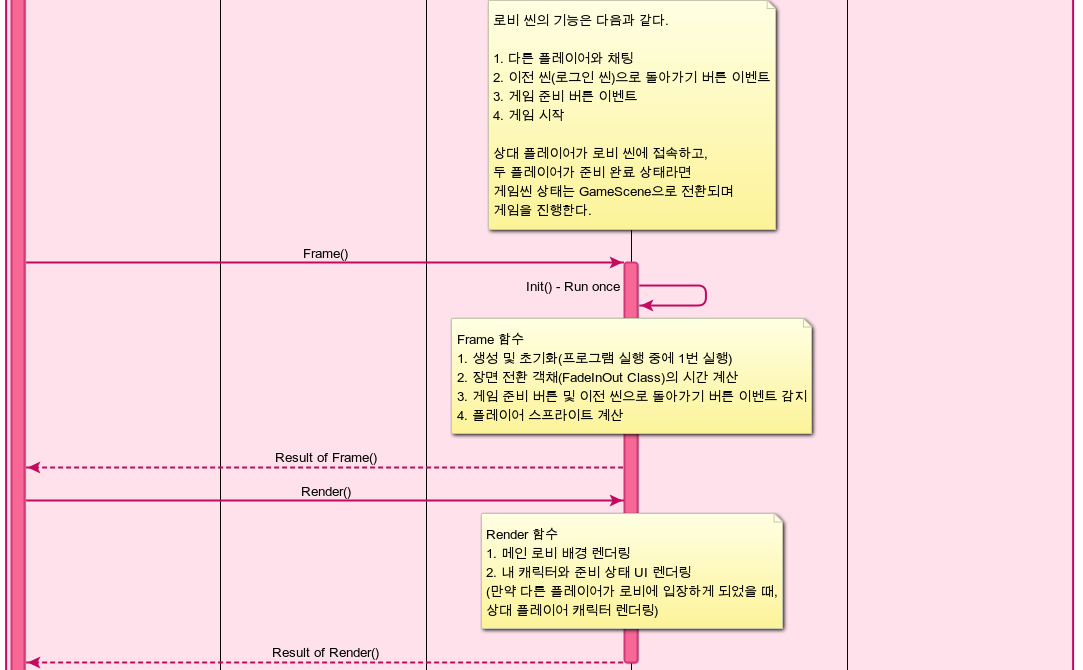
[그림 1-29] 로비씬 패킷 처리 방식

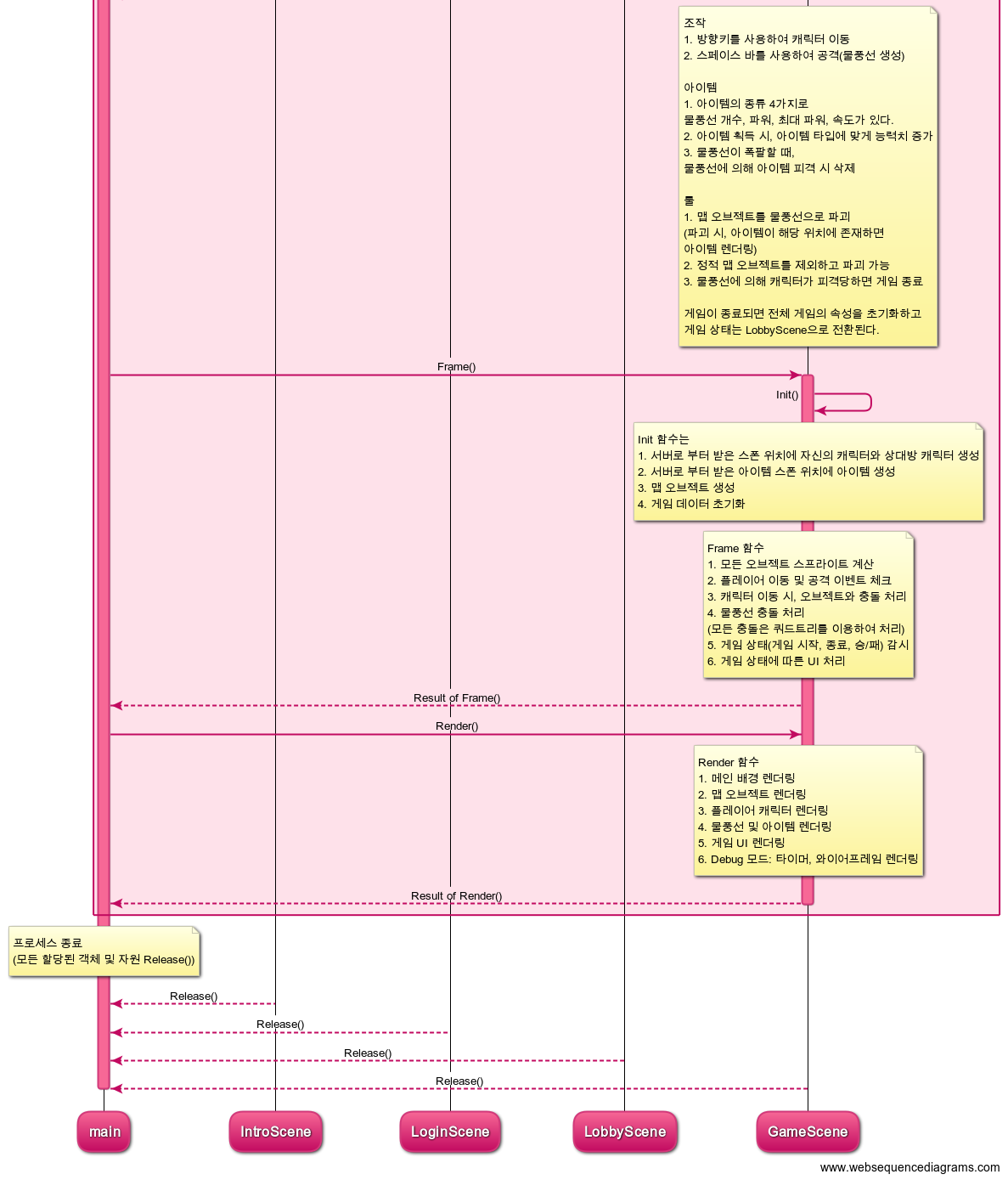


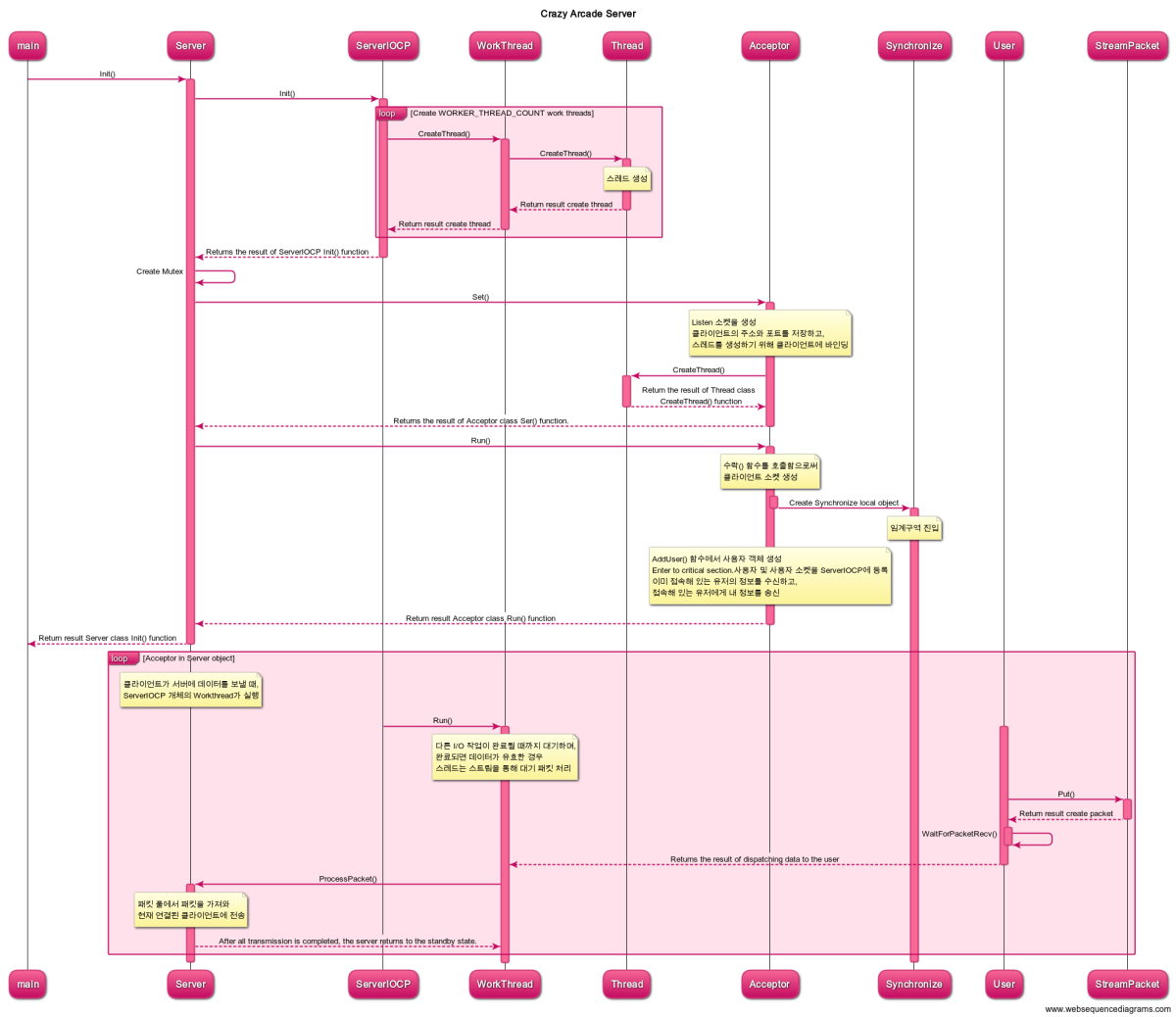
[그림 1-30] 게임씬 패킷 처리 방식

1. **시퀀스 다이어그램(Sequence diagram)**

[그림 2-1] 클라이언트 인트로/로그인 씬 시퀀스 다이어그램

****[그림 2-2] 클라이언트 로비씬 시퀀스 다이어그램

****[그림 2-3] 클라이언트 게임씬 시퀀스 다이어그램

****[그림 2-4] 서버 시퀀스 다이어그램

1. **최종 결과**

* **게임 영상**

**<https://youtu.be/6yqhOGqjNo4>**

* **게임 풀 소스**

[**https://github.com/MingyuOh/WinAPI\_2D\_CrazyArcade**](https://github.com/MingyuOh/WinAPI_2D_CrazyArcade)