# 기획팀과의 내용

* **brach: dev/23.02.3.001/quest-objective-type**

담당자:

| * 기획자: 김정우님 * 클라이언트팀: 이민오님 |
| --- |

<https://www.notion.so/Objective-Achieve-c242ed5266e54c00854930f89875e587>

* 퀘스트 임무 조건 추가 개발 - <http://redmine.supercat.in/issues/11942>
  + 업무 링크: [링크](https://docs.google.com/document/d/1YZ_Alro-xq35AuDHaqaoD-TTxaCgYLddtSbPO26aEv8/edit)
  + 레드 마인 이슈 생성해서 진행 해야 함
  + 우선 샘풀로 2개 먼저 작업
  + Stack

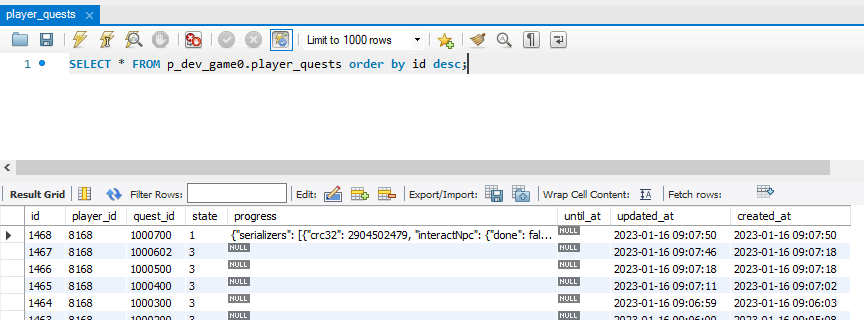
| * 일회성: 미션(퀘스트) 에서 쓰이는 타입   + 모든 몬스터 처치, 재화 사용, 퀘스트가 완료되면 정보 초기화,   + 위에 2가지 사항부터 작업 * 누적형: 업적, 칭호 에서 쓰이는 타입   + 추후에 작업 |
| --- |

* + 구두 논의 내용 정리합니다.

| 구두 논의 내용 정리합니다.  상기의 제가 제안 드린 내용이 가능한 것으로 얘기되었습니다.   * 서비스 오픈 전 모든 Objective는 개발 완료 되어 있어야 함 * 카운트 값은 각 Objective 타입 당 독립적으로 카운트 저장 * 데이터 작업에 구애 받지 않음 * 동일한 수행 조건인데도 일회성과 누적형은 구분지음 * 선 작업 요청 항목   + 일회성 - 모든 몬스터 처치   + 일회성 - 재화 사용 |
| --- |

1/31

| ev/23.1.3.001/quest-objective-type  오브젝트 1개별로 퀘스트 조건 데이터를 추가하여 올렸습니다.  추가 수정 요청 사항이 있습니다.  enum 이름 변경  EnhanceBulge -> EnhanceAttain  Target 인자로 좀더 세부 조정 가능 (Target ID가 아이템 메인 타입을 지정) 0이면 All  장비 아이템 강화 시도 횟수 = EnhanceTry  장비(무기,방어구) 아이템 강화 달성 횟수 = EnhanceAttain  검토 부탁드립니다. |
| --- |

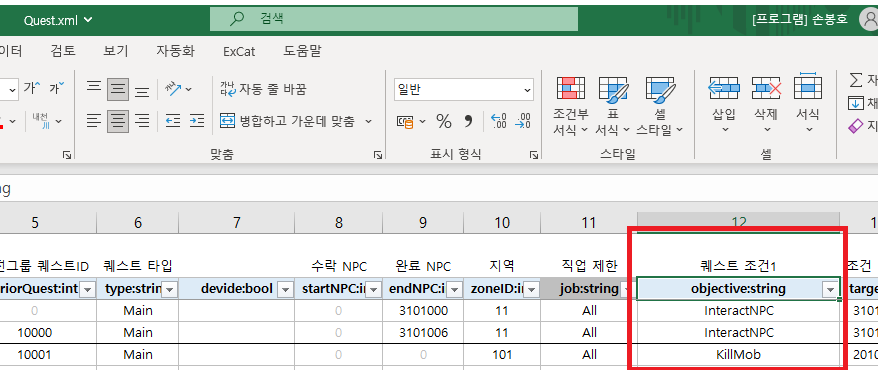


# 퀘스트 리소스

| public class RQuestSet : ICheckable  {  public void Load(FBDataLoader loader)  {  *// 퀘스트 아이디로 해싱*  \_questGroupById = loader.LoadQuestData()  .Select(x => RQuest.Of(x.Value))  .ToDictionary(x => x.Spec.Id, x => x);  \_questGroupById.TrimExcess();  *// 퀘스트 그룹 아이디로 해싱*  \_questGroupByGroupId = \_questGroupById  .Values  .GroupBy(x => x.Spec.QuestGroupID, x => x)  .ToDictionary(x => x.Key, x => x.ToImmutableList());  \_questGroupByGroupId.TrimExcess(); |
| --- |

## 퀘스트 리소스 로드

| public Dictionary<int, FBDataQuestElement> LoadQuestData()  {  ByteBuffer byteBuffer;  byteBuffer = ReadItems(typeof(FBDataQuest).Name);  return FBDataQuest.GetRootAsFBDataQuest(byteBuffer).ToDict();  } |
| --- |



* Obj에 Objective Type값이 들어가 있다.

| public static RQuest Of(FBDataQuestElement element)  {  var entity = new RQuest();  entity.Spec = entity.AddProperty(new RQuestProperty(element));  entity.Obj = entity.AddProperty(new RQuestObjectiveProperty(element));    return entity;  } |
| --- |

* 각 타입에 맞게 처리

| public RQuestObjectiveProperty(FBDataQuestElement element)  {  var objectives = new List<RObjectiveProperty>();  for (int i = 0; i < element.ObjectiveLength; ++i)  {  var objectiveName = element.Objective(i);  *//var objectiveType = ObjectiveType.OfNameWithThrow(objectiveName, 0 == i);*  *// TODO : 9월 이후 다시 정리해야 함,*  var objectiveType = ObjectiveType.OfNameWithThrow(objectiveName);  if (ObjectiveType.None == objectiveType)  continue;  int targetId = element.Target(i);  int targetCount = element.Count(i);  int chanceRate = element.ChanceRate(i);  var resObjective = new RObjectiveProperty(element.Id, objectiveType, targetId, targetCount, chanceRate);  objectives.Add(resObjective);  }  \_objectives = objectives;  } |
| --- |

# GamePlayer

| public class GamePlayer : XComponent  {  public static GamePlayer Of(XEntity entity, PlayerRow row, TAppearance appearance = null)  {  var player = entity.AddComponent<GamePlayer>();  player.Flux = entity.AddComponent<FluxComponent>();  player.Session = entity.AddComponent<PlayerSessionComponent>();  player.Timer = entity.AddComponent<PlayerTimerComponent>();  player.Psn = entity.AddComponent<PlayerPersonalComponent>();  player.Exp = entity.AddComponent<PlayerExpComponent>();  player.Wallet = entity.AddComponent<PlayerWalletComponent>();  player.Quest = entity.AddComponent<PlayerQuestComponent>();  player.Bag = entity.AddComponent<PlayerBagComponent>(); |
| --- |

## PlayerQuestComponent

* QuestBase
* QuestProgress

| public class PlayerQuestComponent : XComponent  {  private ConcurrentDictionary<int, QuestBase> \_quests;  public void Initialize()  {  var rSys = GetSystem<ResourceSystem>();  var db = GetSystem<DatabaseSystem>();  var rows = db.Game.PlayerQuest.GetAllAsync(\_player.Psn.Uid)  .Result  .AsList();  *// 진행 중인 퀘스트들*  \_quests = rows  .Select(x => Tuple.Create(x, QuestBuilder.OfRow(rSys, x)))  .Where(x => null != x.Item2)  .ToDictionary(x => x.Item1.quest\_id, x => x.Item2)  .ToConcurrentDictionary();  *// 변경 리액티브 결합*  \_quests.ForEach(knv =>  knv.Value.Subscribe(k => OnRxObjectiveChangedInternal(knv.Key, k)));  *// 진행 플럭스 결합*  \_quests.Values  .ForEach(x => x.ToFluxStore(\_player.Flux.AsFlux()));  } |
| --- |

| // 퀘스트 진행 사항 알림  private void OnRxObjectiveChangedInternal(int questId, ObjectiveProgressBase objective)  {  var notify = new PlayerQuestProgressNotify  {  QuestId = questId,  Progress = objective.Serializer,  };  \_player.Session.Write(0, notify);  } |
| --- |

## QuestBuilder

| public static class QuestBuilder  {  public static QuestBase OfRow(ResourceSystem rSys, PlayerQuestRow row)  {  var resQuest = rSys.Quest.GetById(row.quest\_id);  if (null == resQuest)  return null;  if (QuestState.Progress.Value != row.state)  return new QuestBase(row.state);  var rsz = string.IsNullOrEmpty(row.progress)  ? new TRepeatedObjectiveSerializer()  : row.progress.FromJson<TRepeatedObjectiveSerializer>();  var objectives = ImmutableObjectivesOf(resQuest, rsz);  return new QuestProgress(objectives);  }  *// 퀘스트 옵션으로 오브젝티브 생성*  public static ImmutableArray<ObjectiveProgressBase> ImmutableObjectivesOf(  RQuest rQuest, TRepeatedObjectiveSerializer rsz)  {  var objectives = new List<ObjectiveProgressBase>();  foreach (var rObj in rQuest.Obj.AsArray())  {  *// 같은 Crc32 를 찾는다*  var sz = rsz.Serializers.FirstOrDefault(x => x.Crc32 == rObj.Crc32);  var objective = ObjectiveProgressFactory.Shared.Of(rObj, sz);  objectives.Add(objective);  }  return objectives.ToImmutableArray();  } |
| --- |

# ObjectiveProgressFactory

| public sealed class ObjectiveProgressFactory  {  public static ObjectiveProgressFactory Shared { get; } = new();  private readonly Dictionary<ObjectiveType, Func<RObjectiveProperty, TObjectiveSerializer, ObjectiveProgressBase>>  \_constructors;  private ObjectiveProgressFactory()  {  var temp =  new Dictionary<ObjectiveType, Func<RObjectiveProperty, TObjectiveSerializer, ObjectiveProgressBase>>();  temp.Add(ObjectiveType.KillMob,  (res, progress) => new KillMobObjectiveProgress(res, progress?.KillMob));  temp.Add(ObjectiveType.GetJob,  (res, progress) => new GetJobObjectiveProgress(res, progress?.GetJob));  temp.Add(ObjectiveType.ItemGet,  (res, progress) => new GetItemObjectiveProgress(res, progress?.GetItem));  temp.Add(ObjectiveType.InteractNpc,  (res, progress) => new InteractNpcObjectiveProgress(res, progress?.InteractNpc));  temp.Add(ObjectiveType.ArriveZone,  (res, progress) => new ArriveZoneObjectiveProgress(res, progress?.ArriveZone));  temp.Add(ObjectiveType.GuideMove,  (res, progress) => new GuideMoveObjectiveProgress(res)); *// 2022.09 빌드 이후, 규칙 정해서 만들어야 함*  temp.Add(ObjectiveType.KillItemDrop,  (res, progress) => new KillItemDropObjectiveProgress(res, progress?.KillItemDrop));  // 신규 ObjectiveType 추가 필요  temp.Add(ObjectiveType.KillMobAll ,  (res, progress) => new KillMobAllObjectiveProgress(res, progress?.KillMobAll));  \_constructors = temp;  }  public ObjectiveProgressBase Of(RObjectiveProperty resObj, TObjectiveSerializer sz)  {  bool has = \_constructors.TryGetValue(resObj.Type, out var constructor);  if (!has)  return null;  return constructor.Invoke(resObj, sz);  }  } |
| --- |

| public sealed class ObjectiveType  {  public static ObjectiveType None { get; } = new ObjectiveType(0, nameof(None), "모름");  public static ObjectiveType KillMob { get; } = new ObjectiveType(1, nameof(KillMob), "몬스터 처치");  public static ObjectiveType GetJob { get; } = new ObjectiveType(2, nameof(GetJob), "직업 갖기");  public static ObjectiveType ItemGet { get; } = new ObjectiveType(3, nameof(ItemGet), "몬스터 처치 후 드랍 아이템 획득");  public static ObjectiveType InteractNpc { get; } = new ObjectiveType(4, nameof(InteractNpc), "NPC 인터렉션");  public static ObjectiveType ArriveZone { get; } = new ObjectiveType(5, nameof(ArriveZone), "특정 존 입장");  public static ObjectiveType GuideMove { get; } = new ObjectiveType(6, nameof(GuideMove), "이동 위치 도달");  public static ObjectiveType KillItemDrop { get; } = new ObjectiveType(7, nameof(KillItemDrop), "퀘스트 아이템 수집");  *// 신규 개발 필요 사항들*  public static ObjectiveType KillMobAll { get; } = new ObjectiveType(8, nameof(KillMobAll), "모든 몬스터 처치");  public static ObjectiveType ItemUse { get; } = new ObjectiveType(9, nameof(ItemUse), "아이템 사용");  public static ObjectiveType CurrencyUse { get; } = new ObjectiveType(10, nameof(CurrencyUse), "재화 사용");  public static ObjectiveType ItemEquip { get; } = new ObjectiveType(11, nameof(ItemEquip), "아이템 장비 착용");  public static ObjectiveType EnhanceTry { get; } = new ObjectiveType(12, nameof(EnhanceTry), "장비 아이템 강화 시도 횟수");  public static ObjectiveType EnhanceSuccess { get; } = new ObjectiveType(13, nameof(EnhanceSuccess), "장비 아이템 강화 성공 횟수");  public static ObjectiveType EnhanceFail { get; } = new ObjectiveType(14, nameof(EnhanceFail), "장비 아이템 강화 실패 횟수");  public static ObjectiveType EnhanceLv { get; } = new ObjectiveType(15, nameof(EnhanceLv), "모든 부위 아이템 강화 단계 최초 달성");  public static ObjectiveType EnhanceWeaponLv { get; } = new ObjectiveType(16, nameof(EnhanceWeaponLv), "무기 아이템 강화 단계 최초 달성");  public static ObjectiveType EnhanceArmorLv { get; } = new ObjectiveType(17, nameof(EnhanceArmorLv), "방어구 아이템 강화 단계 최초 달성");  public static ObjectiveType EnhanceAccLv { get; } = new ObjectiveType(18, nameof(EnhanceAccLv), "장신구 아이템 강화 단계 최초 달성");  public static ObjectiveType EnhanceItemLv { get; } = new ObjectiveType(19, nameof(EnhanceItemLv), "특정 아이템 장비의 강화 최초 달성");  public static ObjectiveType PotionUse { get; } = new ObjectiveType(20, nameof(PotionUse), "포션 아이템 누적 n 회 사용");  public static ObjectiveType CharacterLevel { get; } = new ObjectiveType(21, nameof(CharacterLevel), "캐릭터 레벨 달성");  public static ObjectiveType CharacterDie { get; } = new ObjectiveType(22, nameof(CharacterDie), "캐릭터 누적 사망 횟수");  public static ObjectiveType CurrencyAcquire { get; } = new ObjectiveType(23, nameof(CurrencyAcquire), "누적 재화 습득량");  public static ObjectiveType CurrencySpend { get; } = new ObjectiveType(24, nameof(CurrencySpend), "누적 재화 소모");  public static ObjectiveType MonsterKill { get; } = new ObjectiveType(25, nameof(MonsterKill), "몬스터 타입 n마리 처치 누적 횟수");  public static ObjectiveType QuestClear { get; } = new ObjectiveType(26, nameof(QuestClear), "퀘스트 타입별 클리어 누적 횟수");  public static ObjectiveType FatalDamage { get; } = new ObjectiveType(27, nameof(FatalDamage), "한번에 최대 체력 %의 n% 이상의 피해 받기");  public static ObjectiveType AttainAttack { get; } = new ObjectiveType(28, nameof(AttainAttack), "파괴력 (공격력) 최초 n 값 달성");  public static ObjectiveType AttainDefence { get; } = new ObjectiveType(29, nameof(AttainDefence), "무장도 (방어력) 최초 n 값 달성");  public static ObjectiveType AttainHP { get; } = new ObjectiveType(30, nameof(AttainHP), "최대 체력 최초 n 값 달성");  public static ObjectiveType AttainMP { get; } = new ObjectiveType(31, nameof(AttainMP), "최대 마나 최초 n 값 달성");  public static ObjectiveType PetRideDie { get; } = new ObjectiveType(32, nameof(PetRideDie), "환수 탑승 중 사망 횟수 n회 달성"); |
| --- |

# QuestBuilder

| *// 퀘스트 옵션으로 오브젝티브 생성*  public static ImmutableArray<ObjectiveProgressBase> ImmutableObjectivesOf(  RQuest rQuest, TRepeatedObjectiveSerializer rsz)  {  var objectives = new List<ObjectiveProgressBase>();  foreach (var rObj in rQuest.Obj.AsArray())  {  *// 같은 Crc32 를 찾는다*  var sz = rsz.Serializers.FirstOrDefault(x => x.Crc32 == rObj.Crc32);  var objective = ObjectiveProgressFactory.Shared.Of(rObj, sz);  objectives.Add(objective);  }  return objectives.ToImmutableArray();  } |
| --- |

# GameSession

session.PlayerQuest = entity.AddComponent<GamePlayerQuestController>();

session.Router = entity.AddComponent<GameSessionRouter>();

## GameSessionRouter

# GamePlayerQuestController

# ObjectiveType

| *// DB 에 저장 되는 값이기 때문에, 변경시 항상 서버 프로그래머와 이야기 해서 변경해도 되는지 확인해야 합니다*  public sealed class ObjectiveType  {  public static ObjectiveType None { get; } = new ObjectiveType(0, nameof(None), "모름");  public static ObjectiveType KillMob { get; } = new ObjectiveType(1, nameof(KillMob), "몬스터 처치");  public static ObjectiveType GetJob { get; } = new ObjectiveType(2, nameof(GetJob), "직업 갖기");  public static ObjectiveType ItemGet { get; } = new ObjectiveType(3, nameof(ItemGet), "몬스터 처치 후 드랍 아이템 획득");  public static ObjectiveType InteractNpc { get; } = new ObjectiveType(4, nameof(InteractNpc), "NPC 인터렉션");  public static ObjectiveType ArriveZone { get; } = new ObjectiveType(5, nameof(ArriveZone), "특정 존 입장");  public static ObjectiveType GuideMove { get; } = new ObjectiveType(6, nameof(GuideMove), "이동 위치 도달");  public static ObjectiveType KillItemDrop { get; } = new ObjectiveType(7, nameof(KillItemDrop), "퀘스트 아이템 수집");  public static ImmutableArray<ObjectiveType> Values { get; } = ImmutableArray.Create(  None,  KillMob,  GetJob,  ItemGet,  InteractNpc,  ArriveZone,  GuideMove,  KillItemDrop  );  } |
| --- |

| private ObjectiveProgressFactory()  {  var temp =  new Dictionary<ObjectiveType, Func<RObjectiveProperty, TObjectiveSerializer, ObjectiveProgressBase>>();  temp.Add(ObjectiveType.KillMob,  (res, progress) => new KillMobObjectiveProgress(res, progress?.KillMob));  temp.Add(ObjectiveType.GetJob,  (res, progress) => new GetJobObjectiveProgress(res, progress?.GetJob));  temp.Add(ObjectiveType.ItemGet,  (res, progress) => new GetItemObjectiveProgress(res, progress?.GetItem));  temp.Add(ObjectiveType.InteractNpc,  (res, progress) => new InteractNpcObjectiveProgress(res, progress?.InteractNpc));  temp.Add(ObjectiveType.ArriveZone,  (res, progress) => new ArriveZoneObjectiveProgress(res, progress?.ArriveZone));  temp.Add(ObjectiveType.GuideMove,  (res, progress) => new GuideMoveObjectiveProgress(res)); *// 2022.09 빌드 이후, 규칙 정해서 만들어야 함*  temp.Add(ObjectiveType.KillItemDrop,  (res, progress) => new KillItemDropObjectiveProgress(res, progress?.KillItemDrop));  *// 이곳에 작업하면 될듯!!??*  *// 일반 몬스터 처치*  *temp.Add(ObjectiveType.KillMobNormal,*  *(res, progress) => new KillMobNormalObjectiveProgress(res, progress?.KillMobNormal));*  *// 아이템 사용*  *temp.Add(ObjectiveType.UseItem,*  *(res, progress) => new UseItemObjectiveProgress(res, progress?.UseItem));*  *// 골드 소모*  *temp.Add(ObjectiveType.UseGold,*  *(res, progress) => new UseGoldObjectiveProgress(res, progress?.UseGold));*    \_constructors = temp;  } |
| --- |

## objectiveType 추가 (t\_common.proto )

| // DB에 저장되기 때문에, 타입명, 필드 넘버 바꾸지 말것  message TObjectiveSerializer {  uint32 crc32 = 1;  oneof value {  TArriveZoneObjective arrive\_zone = 5;  TGetJobObjective get\_job = 6;  TInteractNpcObjective interact\_npc = 7;  TGetItemObjective get\_item = 8;  TKillMobObjective kill\_mob = 9;  TGuideMoveObjective guide\_move = 10;  TKillItemDropObjective kill\_item\_drop = 11;  TKillMobAllObjective kill\_mob\_all = 12;  TItemUseObjective item\_use = 13;  TCurrencyUseObjective currency\_use = 14;  }  } |
| --- |

# 객체 Emit

// FxActorKill

| var actorKill = new FxActorKill()  {  ZoneId = \_actor.Location.Zone?.Spec.ZoneId ?? 0,  KillerUid = attacker?.Uid ?? 0,  DeathActorUid = \_actor.Uid, *// 죽은이*  DeathActorDataId = \_actor.Spec.ActorDataId,  Count = 1  };  *// 죽인 유저 전파*  var zone = \_actor.Location.Zone;  if (null != zone) {  damages.ForEach(dm => zone.Actor.Emit(dm.Key, actorKill));  } |
| --- |

| *// 죽인 엑터에 TargetActor 정리*  {  var actorTargetRemove = new FxActorTargetRemove()  {  DeathActorUid = \_actor.Uid,  };  attacker?.Flux.Emit(actorTargetRemove);  } |
| --- |

| *// 아이템 사용*  private void EmitUseItem(int dataId, int qty)  {  *// Wallet 사용*  var fx = new FxUseItem  {  DataId = dataId,  Qty = qty,  };  \_player?.Flux.Emit(fx);  } |
| --- |

# Currency 타입 아이템 소모(FxPlayerWalletSpent)

기존

| public int Spend(ItemOriginFrom origin, int walletType, long amt)  {  return \_wallets.TryGetValue(walletType, out var wallet)  ? wallet.Spend(origin, amt)  : StatusCodeEx.**NotFoundWallet**;  } |
| --- |

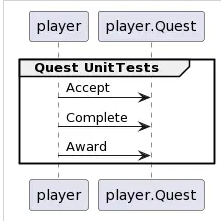
**수정**

| public int Spend(ItemOriginFrom origin, int walletTypeValue, long amt)  {  var has = \_wallets.TryGetValue(walletTypeValue, out var wallet);  if (!has)  {  return StatusCodeEx.**NotFoundWallet**;  }  var status = wallet.Spend(origin, amt);  if (!StatusCodeEx.IsSuccess(status))  {  return status;  }  *// Wallet 사용*  var fx = new FxPlayerWalletSpent  {  Amt = amt,  WalletTypeValue = walletTypeValue,  };  \_player?.Flux.Emit(fx);  return StatusCodeEx.**Success**;  } |
| --- |

# UnitTests

* 전체 보상 테스트

| [Test]  public void TestAllReward()  {  using var entity = TestWorld.DummyGamePlayer.Create();  var player = entity.GetComponent<GamePlayer>();  var questNodes = TestWorld.Res.Quest.RootNodes;  foreach (var questNode in questNodes)  {  var currentNode = questNode;  while (currentNode != null)  {  int status = player.Quest.Accept(currentNode.Id);  Assert.That(status, Is.EqualTo(StatusCodeEx.**Success**),  $"failed to accept - playerUid({player.Psn.Uid}) questId({currentNode.Id})");  status = player.Quest.Complete(currentNode.Id, true);  Assert.That(status, Is.EqualTo(StatusCodeEx.**Success**));  status = player.Quest.Award(currentNode.Id);  Assert.That(status, Is.EqualTo(StatusCodeEx.**Success**));  if (0 == currentNode.Next.Count)  break;  *// 다음 노드 수동 선택*  int nextIdx = Random.Shared.Next(0, currentNode.Next.Count);  int questId = currentNode.Next[nextIdx];  currentNode = TestWorld.Res.Quest.GetNodeById(questId);  }  }  } |
| --- |



| @startuml  group Quest UnitTests  player -> player.Quest: Accept  player -> player.Quest: Complete  player -> player.Quest: Award  end  @enduml |
| --- |

# Monster생성

| public static MonsterActor Of(XEntity entity, MonsterActorOption option)  \_actor = GetComponent<MonsterActor>(); |
| --- |

| public static ActorType None { get; } = new ActorType(0, nameof(None));  public static ActorType Player { get; } = new ActorType(1, nameof(Player));  public static ActorType Monster { get; } = new ActorType(2, nameof(Monster));  public static ActorType Pet { get; } = new ActorType(3, nameof(Pet));  public static ActorType Npc { get; } = new ActorType(4, nameof(Npc));  public static ActorType Portal { get; } = new ActorType(5, nameof(Portal));  public static ActorType Item { get; } = new ActorType(6, nameof(Item)); |
| --- |

* **게임내 actor 생성**

| public class GameActorSystem : XSystem  {  private static readonly ILoggerEx Logger = LoggerFactoryEx.CreateLogger<GameActorSystem>();  private readonly object \_lock;  private readonly ConcurrentDictionary<long, GameActor> \_actors;  private readonly Dictionary<ActorType, Func<XEntity, ActorOption, GameActor>> \_constructors;  public GameActorSystem()  {  \_lock = new();  \_actors = new();  \_constructors = new();  \_constructors.Add(ActorType.Player, (entity, opt) => PlayerActor.Of(entity, opt as PlayerActorOption));  \_constructors.Add(ActorType.Monster, (entity, opt) => MonsterActor.Of(entity, opt as MonsterActorOption));  \_constructors.Add(ActorType.Npc, (entity, opt) => NpcActor.Of(entity, opt as NpcActorOption));  \_constructors.Add(ActorType.Portal, (entity, opt) => PortalActor.Of(entity, opt as PortalActorOption));  \_constructors.Add(ActorType.Item, (entity, opt) => ItemActor.Of(entity, opt as ItemActorOption));  \_constructors.TrimExcess();  }  public GameActor CreateActor(ActorOption option)  {  bool hasConstructor = \_constructors.TryGetValue(option.Type, out var constructor);  if (!hasConstructor)  {  Logger.Error($"not found actor type - type({option.Type.Name})");  return null;  }  var entity = GetWorld().Create(option.Type.Name);  GameActor actor = null;  try  {  actor = constructor.Invoke(entity, option);  if (null == actor)  return null;  \_actors.TryAdd(actor.Uid, actor);  return actor;  }  catch (Exception e)  {  Logger.Error($"", e);  }  finally  {  if (null == actor)  entity.Destroy();  }  return null;  } |
| --- |

# DummyActorSystem

| public class DummyActorSystem : XSystem  {  private static readonly ILoggerEx Logger = LoggerFactoryEx.CreateLogger<DummyActorSystem>();  public XEntity Create()  {  var world = GetWorld();  var entity = world.Create(nameof(DummyActor));  var actor = DummyActor.Of(entity);  return entity;  }  } |
| --- |

## 

## DummyActor

| public static DummyActor Of(XEntity entity)  {  *// 기본*  var actor = entity.AddComponent<DummyActor>();  actor.Flux = entity.AddComponent<FluxComponent>();  actor.Transform = entity.AddComponent<ActorTransform>();  actor.Net = entity.AddComponent<ActorNetDriver>();  actor.Spec = entity.AddComponent<ActorSpecComponent>();    actor.Attr = entity.AddComponent<ActorAttrComponent>();  actor.Health = entity.AddComponent<ActorHealthComponent>();  actor.Mana = entity.AddComponent<ActorManaComponent>();    return actor;  } |
| --- |

### 

| public static DummyActorOption Of(RActor rActor)  {  return new DummyActorOption  {  ResActor = rActor,  SpawnPosition = new GVector3(0),  ReSpawnCrc32 = 0  };  } |
| --- |

### 

### ActorHealthComponent

| public int Damage(long amt, GameActor attacker = null)  {  …  var actorKill = new FxActorKill()  {  ZoneId = \_actor.Location.Zone?.Spec.ZoneId ?? 0,  KillerUid = attacker?.Uid ?? 0,  DeathActorUid = \_actor.Uid, *// 죽은이*  DeathActorDataId = \_actor.Spec.ActorDataId,  Count = 1  }; |
| --- |

# GameActorSystem

| public class GameActorSystem : XSystem  {  public GameActorSystem()  {  \_lock = new();  \_actors = new();  \_constructors = new();  \_constructors.Add(ActorType.Player, (entity, opt) => PlayerActor.Of(entity, opt as PlayerActorOption));  \_constructors.Add(ActorType.Monster, (entity, opt) => MonsterActor.Of(entity, opt as MonsterActorOption));  \_constructors.Add(ActorType.Npc, (entity, opt) => NpcActor.Of(entity, opt as NpcActorOption));  \_constructors.Add(ActorType.Portal, (entity, opt) => PortalActor.Of(entity, opt as PortalActorOption));  \_constructors.Add(ActorType.Item, (entity, opt) => ItemActor.Of(entity, opt as ItemActorOption));  \_constructors.TrimExcess();  } |
| --- |

| public RActorMonsterProperty(FBDataActorMonsterElement element)  {  Id = element.Id;  GroupID = element.GroupID;  GraphAsset = element.GraphAsset; |
| --- |

# 리뷰

## 1차

1. Fx 이벤트류 전부 required get; init 로 변경

예)

public class FxPlayerUseItem : IFluxAction

{

public required int DataId { get; init; }

public required int Qty { get; init; }

}

2. 가방에 아이템 삭제 이유를 넣을 수 있게 수정 필요

3. UseGoldObjectiveProgress 를 UseCurrencyObjectiveProgress 로 변경하고

- 다양한 커런시를 수용할 수 있는 구조로 변경 필요

## 2차

1. 커런시 소모 퀘스트에서 월렛 타입이 존재하지 않는 타겟 아이디에 대한 체크가 필요함 (수정완료)

* 데이터 체크에 대해서는 민오님과 이야기 나눠서 진행해주세요

2. 이벤트는 발생 위치에서는 이벤트 받는 객체에 대한 코드를 넣으면 안되요(수정완료)