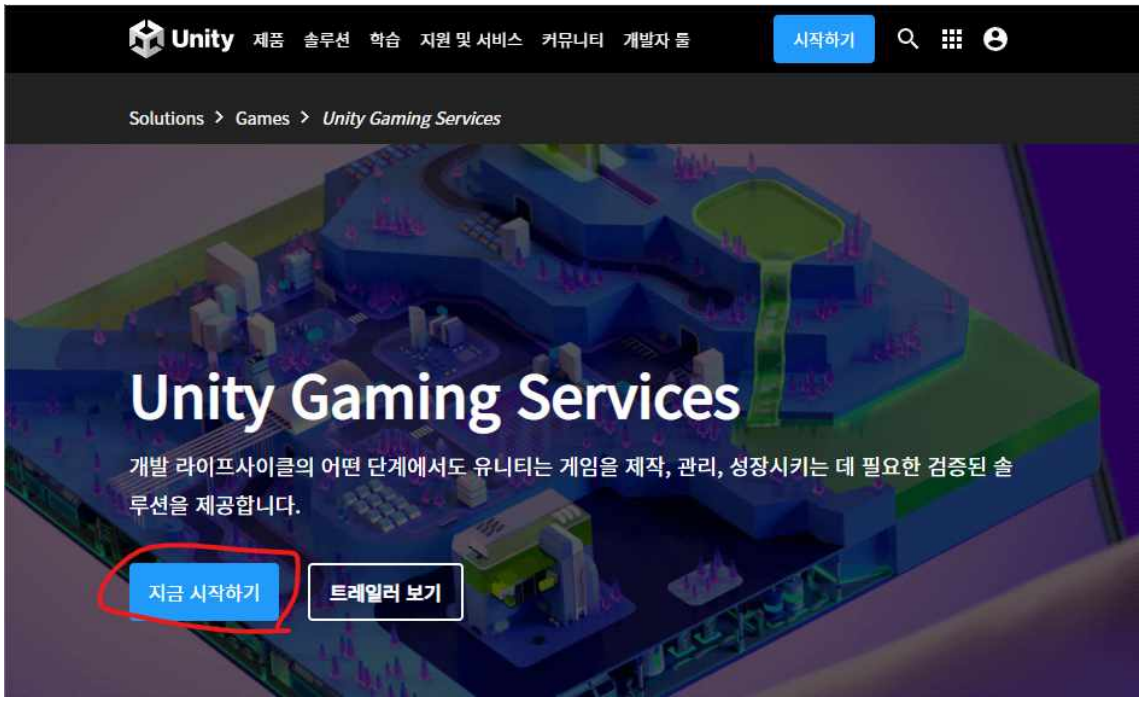
〈UGS〉

1. 개요

Unity Gaming Services(UGS)는 게임 개발자가 운영, 멀티플레이어, 경제 시스템, 원격 설정 등 핵심 기능을 빠르고 안정적으로 구현할 수 있도록 돕는 통합 서비스입니다. 본 문서는 회사 개발 자료로 활용하기 위해, Tistory의 4개 게시글과 추가적인 예제 코드를 바탕으로 UGS 학습 및 실습용 튜토리얼을 정리한 것입니다.



2. UGS 주요 기능

- Game Operations: 원격 설정(Remote Config), 클라우드 코드(Cloud Code), A/B 테스트, 콘텐츠 관리
- Multiplayer: 로비, Relay, Netcode for GameObjects(NGO)
- Economy: 가상 화폐, 아이템, 보상 시스템 관리
- Analytics: 플레이어 행동 추적, 대시보드 기반 분석
- LiveOps: 이벤트 관리, 일일 보상, 유지보수 기능

라이브 게임을 위한 완벽한 서비스 생태계

기반 구축

계정
플랫폼 전반 플레이어 로그인 및 Authentication 시스템을 활성화합니다.
멀티플레이어
확장성과 성능의 Multiplayer 게임 호스팅을 얻으십시오.
콘텐츠 관리
게임 콘텐츠를 배포하고 관리합니다.
DevOps
게임 개발을 위한 Version Control 및 Build Automation 탐색합니다.

플레이어 참여 유도

Analytics
Analytics 도구를 사용하여 데이터 기반 의사 결정을 추진합니다.
플레이어 참여
실험 및 참여 도구로 플레이어 경험을 최적화합니다.
커뮤니티
확장 가능한 음성 및 텍스트 채팅을 통해 플레이어를 연결하고 안전하게 유지하십시오.
크래시 리포트
충돌 보고 및 조사 도구를 사용하여 게임 안정성을 향상시킵니다.

게임 성장시키기

수익화
게임 내 광고로 수익 창출
사용자 확보
고객의 성장에 적합한 사용자 찾기.
광고 증재
광고 수요를 높이고 게임에서 더 많은 수익을 창출하십시오.
퍼블리싱
모바일 게임 아이디어를 비즈니스로 구축합니다.
게임 Economy
게임 내 Economy 인프라를 설계하고 앱 내 구매를 추가합니다.

3. 개발 환경 세팅

3.1 Unity 버전: 2020.3 이상 권장

3.2 Unity Dashboard

- 조직 및 프로젝트 생성
- UGS 서비스 활성화

3.3 필수 패키지 설치

- Remote Config, Cloud Code, Economy, Netcode, Relay 등

3.4 샘플 프로젝트 확보

-GitHub: [GameOperationsSamples](링크)

-GitHub: [GameLobbySample](링크)

-GitHub: [BossRoom](링크)

4. UGS Multiplayer 기본 예제

4.1 준비사항

Package Manager 설치

- Netcode for GameObjects
- Unity Transport (UGS Multiplayer 기본 전송 계층)

4.2 UGS Multiplayer 서비스 연결

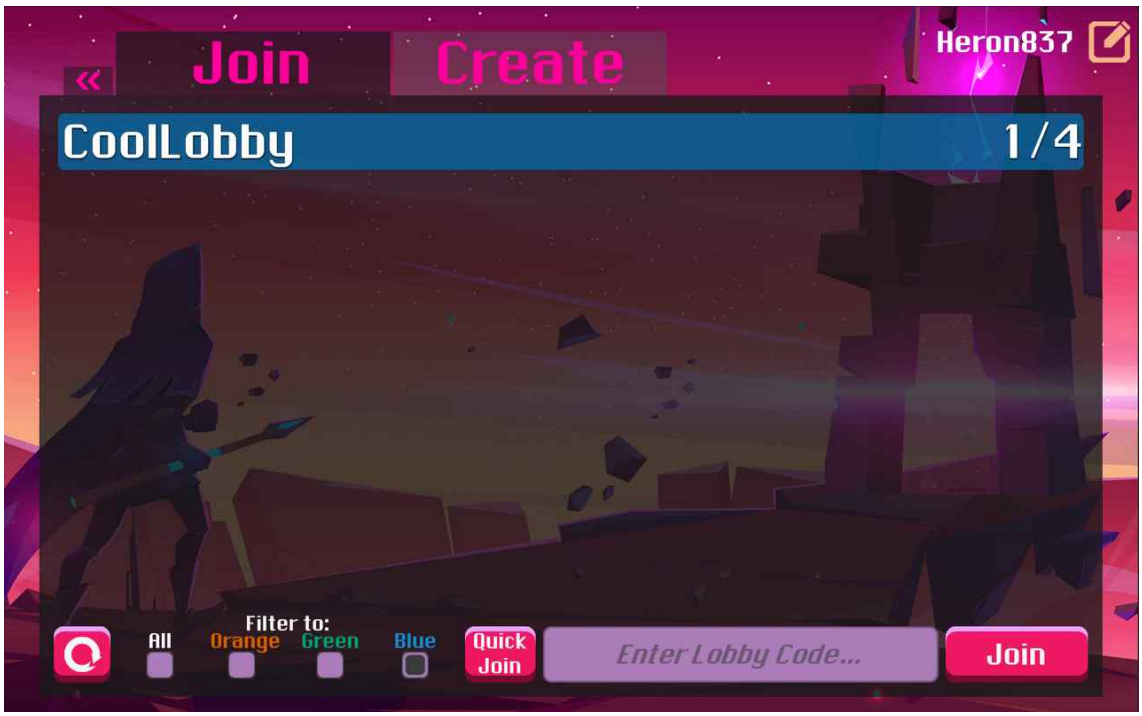
Unity 메뉴 상단에서 Services→Unity 계정 로그인

프로젝트 ID 생성 (또는 기존 프로젝트 연결)

Multiplayer 서비스 활성화

Relay 활성화

Lobby 활성화 (매칭용)



4.3 기본 네트워크 구조 (예제 코드)

```
using Unity.Netcode;
using UnityEngine;
public class GameNetworkManager : MonoBehaviour
{    void OnGUI()
   {
      GUILayout.BeginArea(new Rect(10, 10, 200, 200));
      if                    (!NetworkManager.Singleton.IsClient                    &&
!NetworkManager.Singleton.IsServer)
      {
         if (GUILayout.Button("Host 시작"))
         {
            NetworkManager.Singleton.StartHost();
         }
         if (GUILayout.Button("클라이언트 접속"))
         {
            NetworkManager.Singleton.StartClient();
         }
         if (GUILayout.Button("서버 전용 시작"))
         {
            NetworkManager.Singleton.StartServer();
         }
      }
      else
      {
         if (GUILayout.Button("연결 종료"))
         {
            NetworkManager.Singleton.Shutdown();
         }
      }
      GUILayout.EndArea();
   }
}
```

설명

StartHost() : 서버 + 클라이언트 동시에 실행

StartClient() : 호스트에 접속

StartServer() : 서버 전용 (주로 테스트용)

4.4 네트워크 오브젝트 만들기

이동할 플레이어 프리팹 생성 (큐브/캡슐)

NetworkObject 컴포넌트 추가

NetworkTransform 컴포넌트 추가 (위치 동기화 자동)

NetworkManager 오브젝트에 Player Prefab 지정

4.5 간단한 플레이어 움직임 동기화

```
using Unity.Netcode;
using UnityEngine;
public class PlayerMovement : NetworkBehaviour
{    public float speed = 5f;
    void Update()
    {
        if (!IsOwner) return; // 자기 클라이언트만 조작
        float h = Input.GetAxis("Horizontal");
        float v = Input.GetAxis("Vertical");
        transform.Translate(new Vector3(h, 0, v) * speed * Time.deltaTime);
    }
}
```

[설명]

IsOwner: 해당 NetworkObject가 현재 클라이언트에게 소유권이 있는지 여부를 알려주는 속성입니다.

만약

IsOwner == true

– 이 오브젝트는 현재 클라이언트가 소유한 오브젝트. 따라서 입력(조작) 같은 클라이언트 전용 로직을 실행할 수 있습니다.

IsOwner == false

– 다른 클라이언트가 소유한 오브젝트. 현재 클라이언트에서는 직접 조작할 수 없고, 네트워크 동기화된 상태만 받아옵니다.

움직임은 NetworkTransform이 자동으로 다른 클라이언트에 동기화

4.6 Relay & Lobby 적용 (온라인 플레이)

Relay: NAT 방화벽 뒤에서도 연결 가능

Lobby: 로비 생성/매칭 지원

흐름

- Lobby 생성

LobbyService.Instance.CreateLobbyAsync()

- Relay 할당

RelayService.Instance.CreateAllocationAsync()

- 호스트는 Relay 코드를 공유, 클라이언트는 JoinLobbyAsync()로 참여

4.7 빌드 & 테스트

빌드 실행

움직임이 네트워크를 통해 동기화되는지 확인

성공 시 온라인 플레이 기초 구현 완료

5. 단계별 튜토리얼 (샘플 중심)

5.1 Game Operations Sample 실습

- Loot Boxes 구현: Economy + Cloud Code
- 일일 보상 시스템: Remote Config로 보상 주기 변경 가능
- A/B 테스트: 콘텐츠 난이도/보상 차이를 실험
- 실습: 간단한 보상 상자 구현 후, Remote Config로 파라미터 수정

5.2 Lobby & Relay 심화

- Lobby 생성 및 참가: Game Lobby Sample 활용
- Relay 서비스 연동: 안정적인 P2P 연결
- Netcode for GameObjects: UTP 기반 통신 구조 이해
- 실습: 2인 플레이 로비 프로젝트 제작

5.3 Boss Room Sample 분석

- 협동 멀티플레이 구조 학습: 서버–클라이언트 아키텍처
- 네트워크 동기화: NetworkedVars, RPC 사용
- 멀티플랫폼 테스트: Windows, Mac, iOS, Android 빌드
- 실습: Boss Room에 새로운 캐릭터/스킬 추가

5.4 배포 및 운영 고려사항
– 보안 설계: 서버 권위(authoritative) 모델 적용
– 지연(latency) 처리: 입력 예측, 보정 기술
– 운영: 로그 관리, 분석, 업데이트 자동화
– 비즈니스: Economy 설계, 과금 모델 연동

6. 실습 프로젝트 제안

〉{협동 퀘스트 + 보상 시스템 게임}

– 플레이어: 로비 생성 – 멀티플레이 퀘스트 수행 – Economy 보상 획득
– 운영: Remote Config로 난이도/보상 변경, A/B 테스트 적용
– 배포: Android/iOS 빌드 후 운영 데이터 수집

7. 학습 흐름

7.1 개념 이해 (UGS 개요)
7.2 환경 구축 (Unity + Dashboard)
7.3 샘플 실습 (GameOps-Lobby-Boss Room)
7.4 운영/배포 전략 학습
7.5 회사 프로젝트 적용

8. 결론

UGS는 개발 생산성을 높이고, 운영 효율성을 강화하며, 플레이어 경험을 최적화하는 핵심 도구입니다. 본 튜토리얼을 기반으로 회사 내 프로젝트에 UGS를 적용하면 멀티플레이어 지원, 이벤트 운영, 경제 시스템 구현을 체계적으로 구축할 수 있습니다.

[문서작성 참고자료: 티스토리]

1. 소개
2. Lobby
3. GameOperations
4. BossRoom

[Unity 공식사이트]

1. 공식 홈페이지
2. 가격 기준

[추가 질문]

1. Unity Gaming Service 멀티플레이 유니티에서 구현하는 법 설명 튜토리얼로
2. 이거 근데 어몽어스류에 쓸건데 정보 관리는 어떻게 해?
3. P2P 유지하면서 할 순 없는건가?
4. 보안도 계속 유지 가능해?
5. 공식퍼블릭매칭서비스