홍지형 Portfolio

목차

게임 개요

1) 게임명: 액션 PvP 캐주얼 하키 게임

2) 프로젝트 기간: 2020.03 ~ 2020.12

3) 게임 간단 요약:

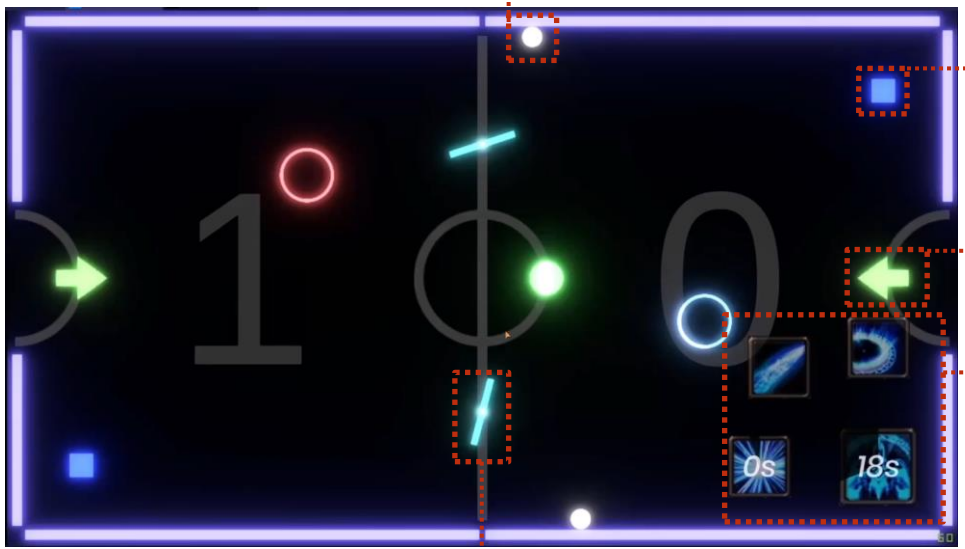
- 액션 게임의 스킬 기능을 전략적으로 활용하여 상대방과 대결하는 아케이드 게임입니다.
- PvP(1:1) 방식으로 게임이 진행되며, 중앙에 배치된 하키공을 부딪쳐 상대방의 골대에 넣으면 1점을 획득합니다.
- 5점을 먼저 획득하는 플레이어가 승리합니다.
- 화면의 아무 곳이나 터치하여 가상 조이스틱으로 공을 움직입니다.
- 배경 뒤의 큰 숫자로 각 플레이어가 득점한 점수를 나타내며, 각 플레이어는 가운데 선을 넘어서 이동할 수 없습니다.

4) 개발 환경:

- 개발 엔진: Unity
- 프로그래밍 언어: C#
- 멀티플레이 환경: Photon Unity Network
- 지원 플랫폼: Android

게임 설명

방해 공
•하키 공과 비슷한 오브젝트로 맵에 2개 배치
•이를 이용해서 상대방을 혼란시키는데 사용



순간 이동 장애물
•하키 공이나 방해 공이 해당 장애물과 부딪히게 되면 부딪힌 공을 속도를 유지한 채로 맵의 정중앙으로 순간 이동시킴

부스터 발판
•플레이어가 발판을 밟으면 3초간 플레이어의 이동 속도가 50% 증가

아이콘을 터치하여 스킬을 사용할 수 있습니다.
한번 사용하면 재사용 대기시간이 적용됩니다.
스킬은 Push, Shock, Blink, Swift로 총 4가지로 구성되어 있습니다.

회전 벽 장애물
•공이 부딪히면 가운데가 고정된 채로 회전하는 장애물

게임 설명

이동제한, 득점한 점수 표시

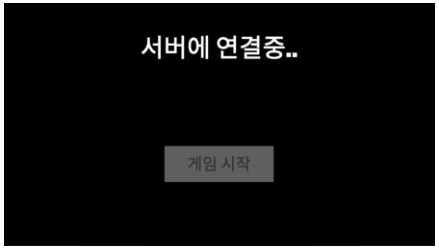
- 각 플레이어는 가운데 선을 넘어서 이동할 수 없습니다.
- 배경 뒤의 큰 숫자는 각 플레이어가 득점한 점수를 나타냅니다.

조작 방법

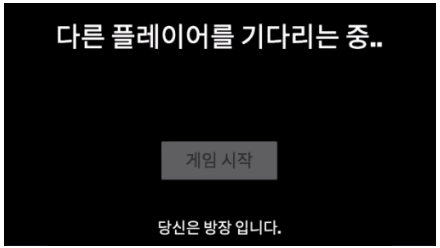
- 화면의 아무 곳이나 터치하여 가상 조이스틱을 움직입니다.
- 중앙에 배치된 하키공을 부딪쳐 상대방의 골대에 넣으면 1점을 획득합니다.

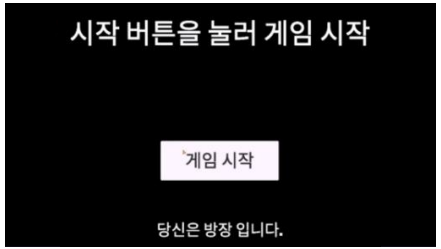
각 플레이어가 조작하는 공 하키공

서버 접속



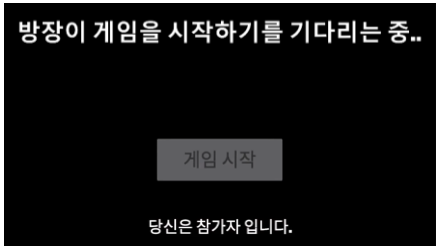
방장 시점





먼저 서버에 접속한 플레이어가 방장이 되어 다른 플레이어가 접속할 때까지 기다립니다.

참가자 시점



방장이 아닌 플레이어는 참가자로 접속해 방장이 게임을 시작할 때까지 기다립니다.

서버 접속

<NetworkManager.cs>

```
public class NetworkManager : MonoBehaviourPunCallbacks {
    [SerializeField] Text textConsole;
    [SerializeField] Button startButton;
    [SerializeField] Text playerInfo;

    private void Awake() {
        PhotonNetwork.AutomaticallySyncScene = true;
        startButton.interactable = false;
        if (!PhotonNetwork.IsConnectedAndReady) {
            textConsole.text = "서버에 연결중..";
            PhotonNetwork.ConnectUsingSettings();
        }
        else {
            if (PhotonNetwork.IsMasterClient) { // 방장
                PhotonNetwork.DestroyAll();
                if(PhotonNetwork.CurrentRoom.PlayerCount == 2) {
                    textConsole.text = "시작 버튼을 눌러 게임 시작";
            ...
            else { // 참가자
                textConsole.text = "방장이 게임을 시작하기를 기다리는 중..";
            }
            CheckPlayerInfo();
        }
    }
    private void CheckPlayerInfo() {
        playerInfo.text = $"당신은 {(PhotonNetwork.IsMasterClient ? "방장" : "참가자")} 입니다.";
}
```

게임을 시작하기 위해 2명의 플레이어가 룸에 접속하고 AutomaticallySyncScene을 true로 설정하여 씬을 동기화합니다. ConnectUsingSettings()를 이용하여 Photon Cloud 계정으로 호스팅한 포톤 서버에 연결합니다. 최초 게임 실행 시 먼저 접속한 플레이어를 방장(마스터 클라이언트)으로 정합니다. IsMasterClient를 이용하여 접속한 플레이어의 방장 여부를 확인합니다.

서버 접속

<GameManager.cs>

```
void Start() {
        // 플레이어를 멀티 객체로 생성하고, 초기화
        var playerID = PhotonNetwork.IsMasterClient ? 0 : 1;
        var viewID = PhotonNetwork.Instantiate(
            "Player",
            playerPositions[playerID].localPosition,
            Quaternion.identity
        ).GetComponent<PhotonView>().ViewID;
        photonView.RPC(nameof(SetPlayer), RpcTarget.All, playerID, viewID);
    }

[PunRPC]
void SetPlayer(int playerID, int viewID) {
     var player = PhotonView.Find(viewID);
     player.name = $"Player{playerID + 1}";
     player.GetComponent<Player>().
     Initialize(moveableAreas[playerID], playerColors[playerID]);
    }
```

GameManager에서 두 플레이어(playerID, viewID)가 접속하여 게임을 시작하면 Instantiate()로 멀티플레이 오브젝트를 생성하고 SetPlayer()로 초기화합니다. GetComponent()를 이용하여 접속한 플레이어의 정보를 가져옵니다.

동기화를 위해서 Photon Unity Network에서 제공하는 photonView를 이용합니다. 서로의 플레이어에게 동일하게 메소드를 호출하기 위해서 SetPlayer()에 [PunRPC] 애트리뷰트를 적용시킵니다.

공의 이동 및 조작

<Joystick.cs>

```
public class Joystick : MonoBehaviour, IDragHandler,IPointerDownHandler, IPointerUpHandler {
    static Joystick instance;
    [SerializeField] RectTransform touchArea;   // 터치 구역
    ...
    float stickRadius;      // 스틱 반지름
    Vector2 moveAmount;     // 이동량
    Vector2 prevMoveAmount; // 이전 프레임의 이동량
    public static Vector2 MoveAmount => instance.moveAmount;
    public static Vector2 PrevMoveAmount => instance.prevMoveAmount;
    ...

    public void OnDrag(PointerEventData eventData) {
        if (!activated) {return};
        if (RectTransformUtility.ScreenPointToLocalPointInRectangle(
                    touchArea, eventData.position,
                    eventData.pressEventCamera, out var touchPos)
           ) {  // 드래그 거리 계산
            var currPos = new Vector2(outerStick.localPosition.x, outerStick.localPosition.y);
            var vector = (touchPos - currPos).normalized;
            var distance = Vector2.Distance(touchPos, currPos) / stickRadius;
            distance = Mathf.Clamp(distance, 0, 1);
            // 드래그 거리만큼 이동량 설정
            moveAmount = vector * distance;
            prevMoveAmount = moveAmount;
            innerStick.localPosition = moveAmount * stickRadius;
        }
  }
```

[가상 조이스틱]

가상 조이스틱을 조작하면 벡터값과 드래그한 거리를 계산하여 그 값만큼 공의 이동을 결정합니다.

처음 터치한 지점(touchPos)과 드래그하여 터치하고 있는 지점(currPos)의 x, y좌표를 벡터값으로 활용합니다.

드래그한 거리를 가상 조이스틱의 반지름(stickRadius)만큼 나눈 값으로, 얼마나 멀리 움직일지(distance)를 정합니다.

최종적으로 MoveAmount에 공의 이동량이 벡터값과 함께 저장됩니다.

공의 이동 및 조작

<Player.cs>

```
public class Player : MonoBehaviourPunCallbacks {
    public enum Effect { Boost, Shock, Push, Blink}
    [Header("Speed")]
    [SerializeField] float moveSpeed;   // 이동 속도
    [SerializeField] float speedFactor; // 속도 증가 배율
    …
    private void Movement() {
              rb.AddForce(
                        Joystick.MoveAmount * moveSpeed *
                        speedFactor * Time.fixedDeltaTime * 100f,
                        ForceMode2D.Force
              );
    }
```

<Ball.cs>

```
    private void FixedUpdate() {
           if (photonView.IsMine){ Movement(); }
           FixPosition();
    }
    // 충돌시 효과음 재생
    private void OnCollisionEnter2D(Collision2D collision) {
        if (photonView.IsMine) { AudioManager.PlaySE(AudioManager.SoundType.Collision); }
    }
```

Joystick 클래스의 MoveAmount 값을 참조해 Player 클래스의 Movement()에서 플레이어의 공이 움직이도록 합니다. 공은 AddForce()를 이용하여 Rigidbody2D로 만들어진 공에게 물리적 힘을 가하여 움직입니다. moveSpeed는 유니티 에디터에서 설정할 수 있는 기본 속도 변수이며, speedFactor는 부스트 발판과 스킬에 의한 속도 증가 배율 변수입니다.
공 조작 시 자신의 공에만 움직임과 효과음 재생이 동작하도록 제한하기 위해 photonView의 IsMine을 이용합니다.

공의 이동 및 조작

<Ball.cs>

```
public class Ball : MonoBehaviourPun {
    Rigidbody2D rb;
    const float speedLimit = 17f;

    private void Awake() {
        rb = GetComponent<Rigidbody2D>();
    }
    private void FixedUpdate() {
        // 속도 제한 걸기
        if (rb.velocity.magnitude > speedLimit)
            { photonView.RPC(nameof(LimitVelocity), RpcTarget.All); }
    }
    public void RPC_AddForce(Vector2 force) {
        photonView.RPC(nameof(AddForce), RpcTarget.All, force.x, force.y);
    }
    [PunRPC]
    void AddForce(float x, float y) {
        rb.AddForce(new Vector2(x, y), ForceMode2D.Impulse);
    }
    [PunRPC]
    void LimitVelocity() {
        rb.velocity = rb.velocity.normalized * speedLimit;
    }
}
```

`FixedUpdate()`에서 공의 속도를 너무 빨라지지 않게 제한하고 `RPC_AddForce()`에서 공의 움직임(이동,밀치기)을 동기화합니다.

두 메소드 모두 동기화를 위한 `photonView`의 `RPC()`를 이용합니다.

`RPC`를 호출하기 위해서 `AddForce()`와 `LimitVelocity()`에 `[PunRPC]` 애트리뷰트를 적용시킵니다.

스킬

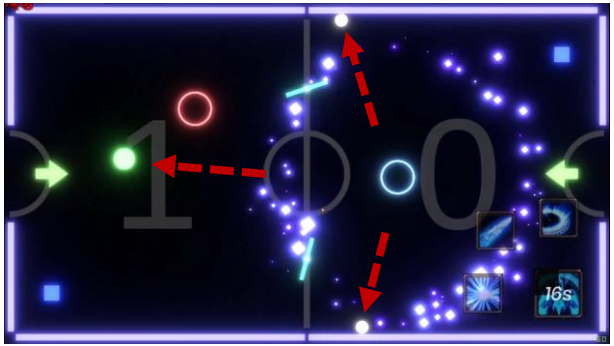
(Shock): 플레이어 사방의 오브젝트를 밀치는 스킬

<SkillManager.cs>

```
void Shock() {
        const float range = 2.5f;
        const float power = 15f;
        foreach (var obj in Physics2D.CircleCastAll(
             player.transform.position, range, Vector2.zero, 0f)) {
               if (obj.transform.TryGetComponent(out Ball ball))
                   { ball.RPC_AddForce(-obj.normal * power); }
        }
        player.ParticleEffect(Player.Effect.Shock);
    }
```

<Ball.cs>

```
public void RPC_AddForce(Vector2 force) {
        photonView.RPC(nameof(AddForce), RpcTarget.All, force.x, force.y);
    }
    [PunRPC]
    void AddForce(float x, float y) {
        rb.AddForce(new Vector2(x, y), ForceMode2D.Impulse);
    }
public void ParticleEffect(Effect type, bool play = true, float rotation = 0f) {
        photonView.RPC(nameof(RPC_Effect), RpcTarget.All, (int)type, play, rotation);
    }
    [PunRPC]
    void RPC_Effect(int type, bool play, float rotation) {
        if (play) { effects[type].Play(); }
        else { effects[type].Stop(); }
   }
```



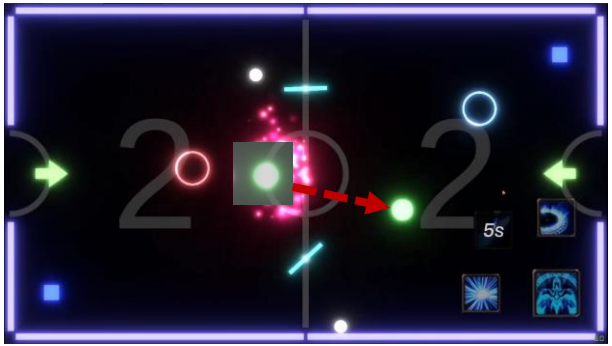
`CircleCastAll()`을 이용해 범위 내 모든 공을 감지합니다. 감지된 공들은 동기화를 위해 `photonView`를 적용시킨 `RPC_AddForce()`로 플레이어 공으로부터 반대 방향으로 힘을 가합니다. 스킬을 사용하는 순간에만 일시적으로 힘을 가하기 위해서 `ForceMode2D`를 `Impulse`로 적용시킵니다. 모든 스킬들은 `ParticleEffect()`를 이용하여 스킬 사용시 파티클 이펙트를 출력합니다. `AddForce()`와 `RPC_Effect()`에 `[PunRPC]` 애트리뷰트를 적용시켜 같은 룸에 있는 플레이어 모두에게 메소드를 동시에 호출시키도록 합니다.

스킬

(Push): 특정 방향으로만 오브젝트를 밀치는 스킬

<SkillManager.cs>

```
IEnumerator PushRoutine() {
        const float range = 0.5f;
        const float distance = 2f;
        const float power = 15f;
        ...
        while (true) {
            yield return null;
            // 화면에서 터치를 때면,
            if (Input.GetMouseButtonUp(0)) {
                var direction = Joystick.PrevMoveAmount.normalized;
                // 범위 내 모든 공을 찾아 충격을 가함
                foreach (var obj in Physics2D.CircleCastAll(
                          player.transform.position, range, direction, distance))
                    if (obj.transform.TryGetComponent(out Ball ball))
                        { ball.RPC_AddForce(-obj.normal * power); }
                player.ParticleEffect(
                       Player.Effect.Push, true,
                       Quaternion.FromToRotation(Vector3.up, direction).eulerAngles.z
                );
                ...
                yield break;
            }
        }
}
```



Shock 스킬과 마찬가지로 `CircleCastAll()`을 이용해 범위 내 모든 공을 감지합니다. 감지 범위는 조이스틱으로 입력한 방향이 저장된 `direction`으로 제한합니다. 감지된 공들은 동기화를 위해 `photonView`를 적용시킨 `RPC_AddForce()`로 플레이어 공으로부터 반대 방향으로 힘을 가합니다. `ParticleEffect()`를 이용하여 스킬 사용 시 파티클 이펙트를 출력합니다.

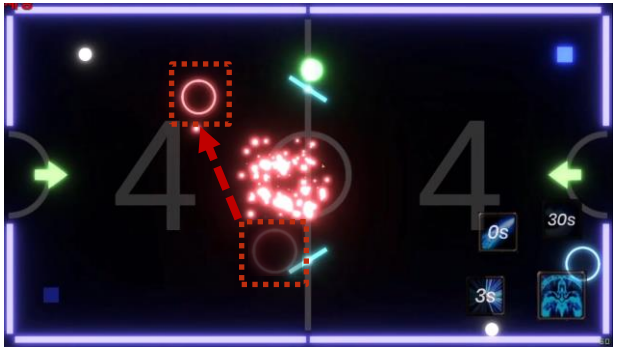
추가적인 조작이 필요하기 때문에 코루틴 기능을 사용하여 구현합니다.

스킬

(Blink): 원하는 곳으로 1회 순간 이동하는 스킬

<SkillManager.cs>

```
IEnumerator BlinkRoutine() {
        var moveableRect = player.MoveableRect;
        Vector3 pos;
        ...
        while (true) {
            yield return null;
            if (Input.GetMouseButtonUp(0)) {
                // 터치 좌표 구하기
                pos = Camera.main.ScreenToWorldPoint(Input.mousePosition);
                // 터치 좌표를 이동가능 구역으로 제한
                pos.x = Mathf.Clamp(pos.x, moveableRect.xMin, moveableRect.xMax);
                pos.y = Mathf.Clamp(pos.y, moveableRect.yMin, moveableRect.yMax);
                // 이동시키기
                player.GetComponent<Rigidbody2D>().MovePosition(pos);
                ...
                player.ParticleEffect(Player.Effect.Blink);
                yield break;
            }
        }
    }
```



ScreenToWorldPoint()를 이용해 화면을 터치한 좌표를 구합니다. Clamp()로 이동 가능한 구역을 제한합니다. MovePosition()을 이용하여 터치한 좌표로 이동합니다. ParticleEffect()를 이용하여 스킬 사용 시 파티클 이펙트를 출력합니다. 추가적인 조작이 필요하기 때문에 코루틴 기능을 사용하여 구현합니다.

감사합니다!