Oyun Programlamaya Giriş Dönem Projesi

Hyper Casual Oyun Uygulaması

Rıdvan GONCA

212511015

Özet

Bu hypercasual oyunda oyuncu, bir yolda ilerlerken karşısına çıkan kapılardan geçerek “oyuncu sayısını” temsil eden değeri matematiksel işlemlerle (toplama, çıkarma, çarpma, bölme vb.) artırmaya veya azaltmaya çalışır. Kapıların üzerinde yazılı olan işlemleri doğru şekilde seçmek, oyuncunun sayı değerini büyütürken; yanlış veya olumsuz işlemler performansı düşürür. Yolda yer alan düşmanlar, sayınızı doğrudan azaltan veya ilerlemenizi engelleyen engeller olarak görev yapar; bu nedenle düşmanlarla çarpışmadan önce yeterli sayıya ulaşmak kritiktir. Oyuncu sayısının artışı, hem çevresel engelleri daha rahat aşmayı sağlar hem de final hedefe ulaşırken en yüksek değeri yakalama motivasyonunu güçlendirir. Genel olarak basit dokunmatik kontrollerle ilerleyen bu oyun, hızlı tempolu oynanışı ve matematiksel kapı mekaniği ile kullanıcıya hem eğlenceli hem de stratejik düşünme fırsatı sunar.

Proje Gereksinimleri

1-Prefab

Oyun içerisinde kullanılan renk paketleri, düşman, düşman grubu ve puan kapıları kodları, animasyonları ve karakterler için toplanma açıları ayarlanıp prefab haline getirildi. Ayrıca küçük, orta ve büyük olmak üzere chunklar ayarlandı ve seviyeler bunun üzerine kurgulandı.

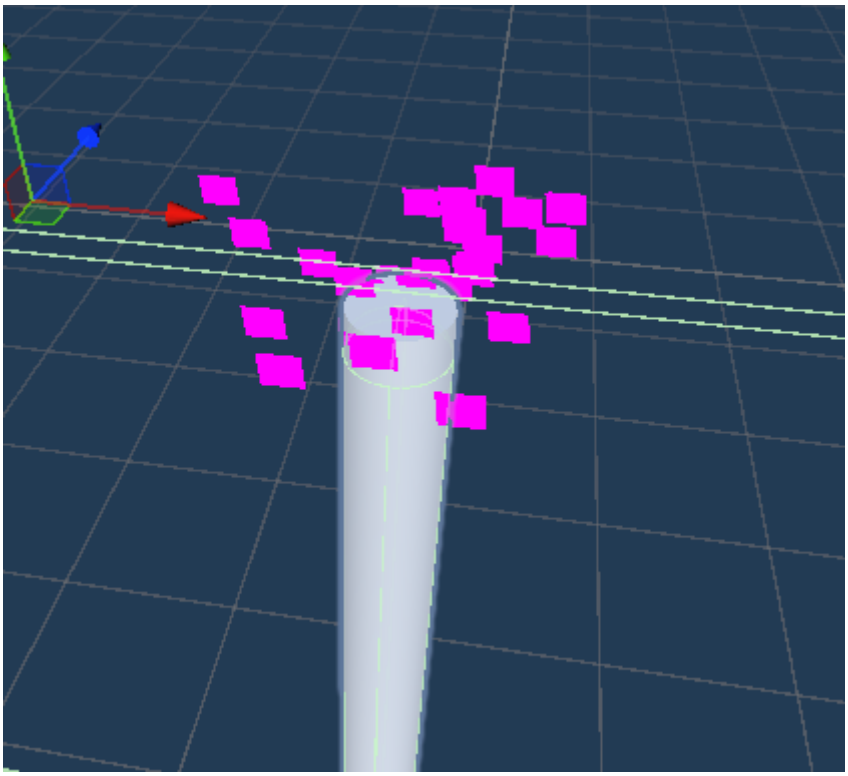
2- Vector3 veya Quaternion

Mixamo’dan aldığım karakterin düşman ve oyuncu hareketleri tamamiyle 3 boyutlu düzlemde çalışması için Vector3 üzerine kurgulanmıştır.

```
1 reference
private void MoveSpeedForward()
{
    transform.position += Vector3.forward * moveSpeed * Time.deltaTime;
}
```

3- Partikül efektleri

Kapıların üzerinde Unity üzerinden hazır elde ettiğimiz default partiküllerden kullanıldı.



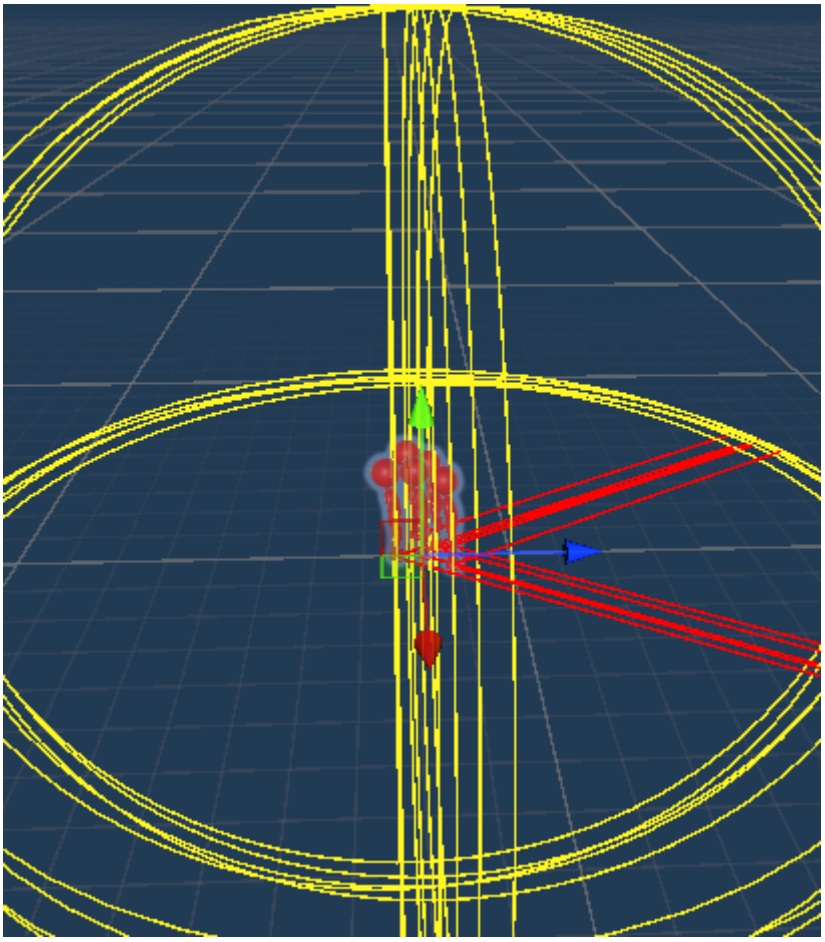
4- Instantiate

Düşmanların oluşumu, chunk sistemi Instantiate olmadan oluşamazlar. Bahsedilenler oyunun başında temelde olmadıkları için sonradan oluşmaktalar.

```
1 reference
private void CreateLevels(Chunk[] levelChunk)
{
    Vector3 chunkPosition = Vector3.zero;
    for (int i = 0; i < levelChunk.Length; i++)
    {
        Chunk chunkToCreate = levelChunk[i];

        if (i > 0)
            chunkPosition.z += chunkToCreate.GetLenght() / 2;

        Chunk chunkInstance = Instantiate(chunkToCreate, chunkPosition, Quaternion.identity, transform);
        chunkPosition.z += chunkToCreate.GetLenght() / 2;
    }
}
```

5- RayCast

Düşmanların oyuncuya saldırma mekaniği RayCast ile belirli tarama alanı içine girdiği zaman saldırması için ayarlandı.



6- FixedUpdate veya LateUpdate

Karakter hareketleri bazen kontrolden çıktığı için kare hızına orantılı olması adına Update yerine FixedUpdate kullanıldı.

```
private void FixedUpdate()
{
    if (canMove)
    {
        MoveSpeedForward();
        ManageControl();
    }
}
```

7- DeltaTime

Oyuncunun ve düşmanların hareketi üzerinde oynayan herkes için aynı kare oranında hızlanması adına DeltaTime kullanıldı.

```
private void MoveSpeedForward()
{
    transform.position += Vector3.forward * moveSpeed * Time.deltaTime;
}
```

8- Arrays

Level sistemi oluşması için chunkların bir araya gelmesi gerekli. Bunun için ise bir arada tutmamız gerekiyor. Bu nedenden ötürü level sistemi array üzerine kurgulandı.

```
private void GenerateLevels()
{
    int currentLevel = GetLevels();
    currentLevel = currentLevel % levels.Length;
    LevelManager level = levels[currentLevel];

    CreateLevels(level.chunks);
}

1 reference
private void CreateLevels(Chunk[] levelChunk)
{
    Vector3 chunkPosition = Vector3.zero;
    for (int i = 0; i < levelChunk.Length; i++)
    {
        Chunk chunkToCreate = levelChunk[i];

        if (i > 0)
            chunkPosition.z += chunkToCreate.GetLenght() / 2;

        Chunk chunkInstance = Instantiate(chunkToCreate, chunkPosition, Quaternion.identity, transform);
        chunkPosition.z += chunkToCreate.GetLenght() / 2;
    }
}
```

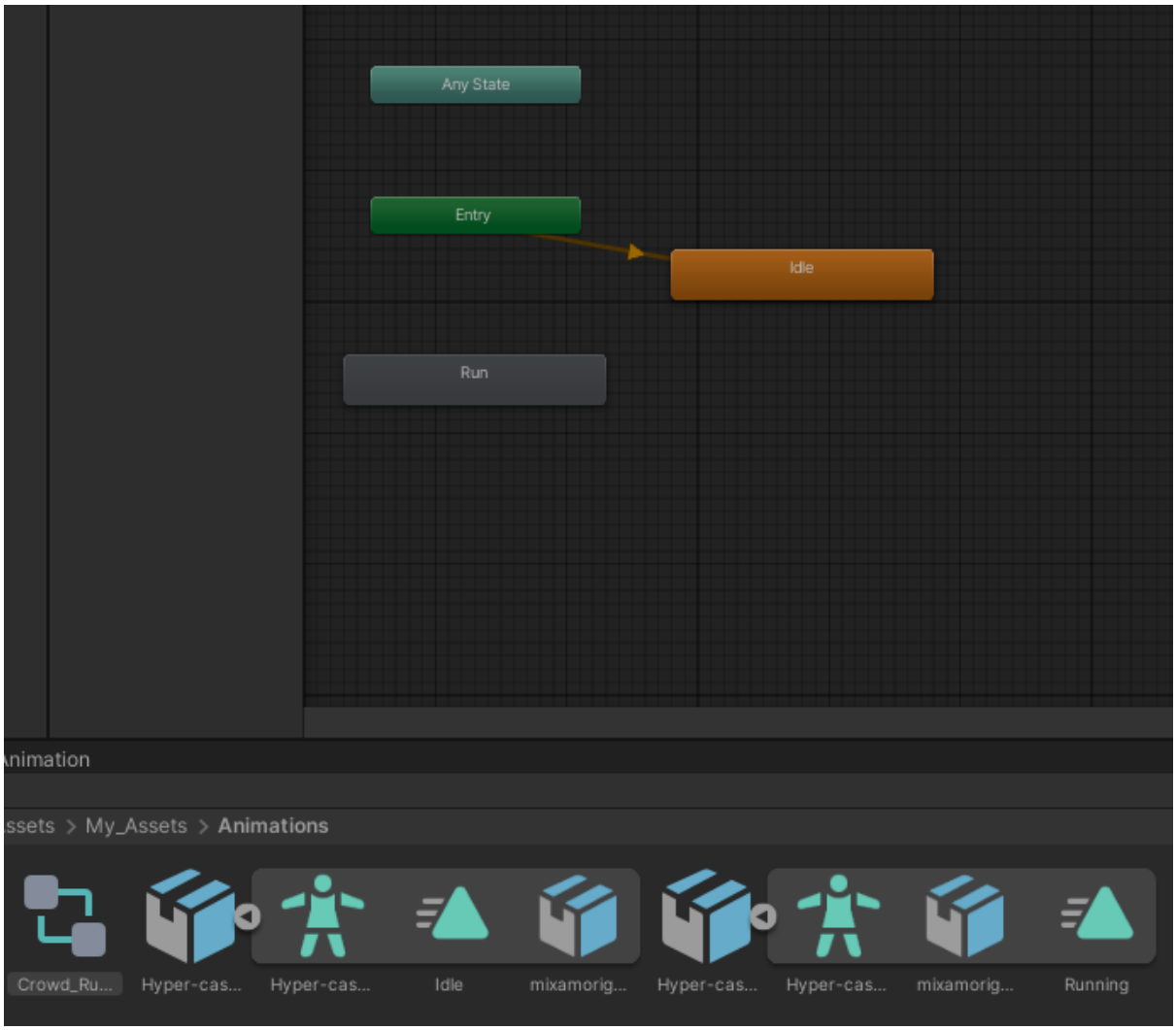
9- Trigger veya Collision

Karakter sayısı kapı üzerindeki matematiksel işlemlere bağlı. Bu işlemlerin tetiklenebilmesi için karakter ile kapı arasında bir tetikleyici gerekli. Kapı prefabları üzerinde Collision Box ile karakter geçişleri kontrol edilmektedir.

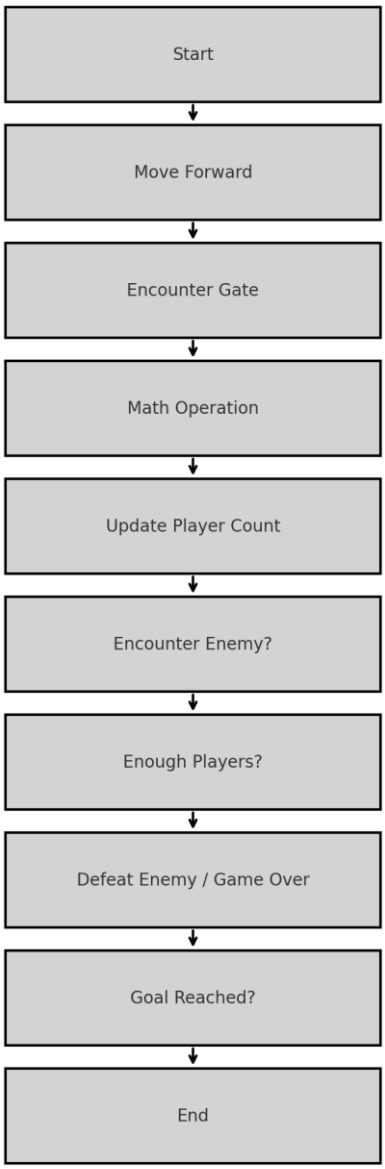
```
private void DetectDoors()
{
    Collider[] detectColliders = Physics.OverlapSphere(transform.position, 1);

    for (int i = 0; i < detectColliders.Length; i++)
    {
        if (detectColliders[i].TryGetComponent(out DoorController doors))
        {
            Debug.Log("Player detected a door");

            int bonusAmount = doors.GetBonusAmount(transform.position.x);
            BonusType bonusType = doors.GetBonusType(transform.position.x);

            doors.DisableDoorCollider();
            crowdSystem.ApplyBonus(bonusType, bonusAmount);
        }

        else if (detectColliders[i].tag == "Finish")
        {
            PlayerPrefs.SetInt("Level", PlayerPrefs.GetInt("Level") + 0);
            GameManager.instance.SetGameState(GameManager.GameState.LevelComplete);
            //SceneManager.LoadScene(0);
        }
    }

}
```

10-Animation-animator kullanımı

Karakterler Mixamo’dan alındığı için karakter paketleri üzerinden gelen hazır animasyon paketleri (koşma ve yerinde durma) olarak 2 tane animasyon yapıldı.



Akış Diyagramı


Start
Move Forward
Encounter Gate
Math Operation
Update Player Count
Encounter Enemy?
Enough Players?
Defeat Enemy / Game Over
Goal Reached?
End