Battle Simulator

Installation GUIDE

> ⚠
> - Unity Version: 6000.0.2f1
> - ECS Version: 1.2.1

Introduction

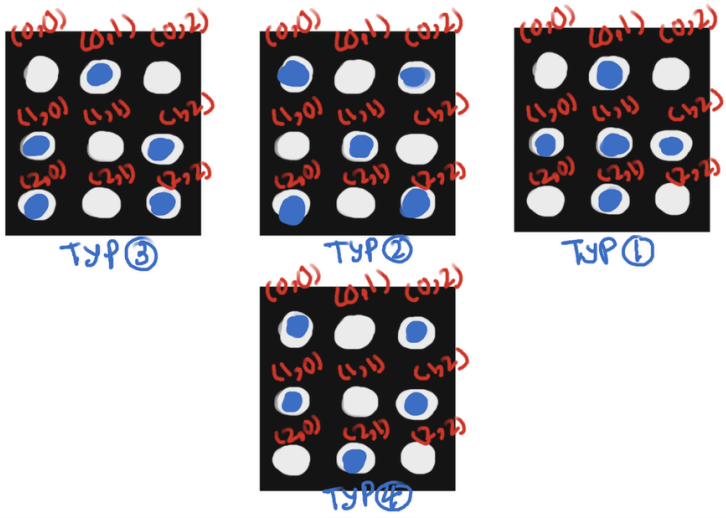
The game is about two opposing teams - a blue team and a red team. The red team's lineup is randomly generated, while the blue team's lineup is chosen by the player to create the best possible lineup to defeat the red team.

Technical Documentation

Initial Setup

Spawning: The team members will appear on a 3x3 grid determined by the team's lineup, you can add team lineup data by adding a new scriptable object. and the system will spawn the units depending on the data. for the prototype, I create 4 lineups.

> ℹ The system adjusts for anything you want to change like the grid or adding many lineups.



4 lineups data in the prototype

So, I create a scriptable object have a list of `vector2` to store x and y for each unit then in baker script I store the x , y in `DynamicBuffer` as `float2` to use it because ECS not support `List` , and `DynamicBuffer` works instead

UnitsCoordinatesConfiguration.cs

```cs
[CreateAssetMenu(fileName = "Lineup",
 menuName = "Scriptable Object/Units Coordinates",
 order = 2)]
public class UnitsCoordinatesConfiguration : ScriptableObject
{
```

```
    public List<Vector2> Coordinates = new List<Vector2>();
}
```

CoordinatesGridBuffer.cs

```
[InternalBufferCapacity(5)]
public struct CoordinatesGridBuffer : IBufferElementData
{
    public float2 axis;
}
```

LineupsMono.cs

```
class LineupsMono : MonoBehaviour
{
    public CoordinatesGridConfigration coordinatesGridConfigration;
}

class LineupsBaker : Baker<LineupsMono>
{
    public override void Bake(LineupsMono authoring)
    {
        var entity = GetEntity(TransformUsageFlags.Dynamic);


        DynamicBuffer<CoordinatesGridBuffer>
        pathBuffer = AddBuffer<CoordinatesGridBuffer>(entity);
        foreach (float2 axis in authoring.coordinatesGridConfigration.Coordinates)
        {
            pathBuffer.Add(new CoordinatesGridBuffer() { axis = axis });
        }

        //add Lineup tag for the entity
        AddComponent(entity, new Lineup());

        DependsOn(authoring.coordinatesGridConfigration);
    }
}
```

For the grid data, I created `IComponentData` containing grid and game data after that I bake the data, and then I read about `IAspect` and how it provides a streamlined approach to accessing and modifying data, resulting in improved performance and faster execution times. so I try to use it to get the grid and game data from it.

GameProperties.cs

```
public struct GameProperties : IComponentData
{

    public int xGridCount;
    public int zGridCount;
    public float baseOffset;
    public float xPadding;
    public float zPadding;
    public Entity gridPrefab;

```

```
    //-----------------

    public float3 teamBlueUnitPostion;
    public float3 teamRedUnitPostion;
    public Entity teamBlueUnitPrefab;
    public Entity teamRedUnitPrefab;

}
```

GameDataMono.cs

```
public class GameDataMono : MonoBehaviour
{
    public float3 teamBlueUnitPostion;
    public float3 teamRedUnitPostion;
    public GameObject teamBlueUnitPrefab;
    public GameObject teamRedUnitPrefab;

    //---------------------------------

    public int xGridCount;
    public int zGridCount;
    public float baseOffset;
    public float xPadding;
    public float zPadding;
    public GameObject gridPrefab;
}

public class GameDataBaker : Baker<GameDataMono>
{
    public override void Bake(GameDataMono authoring)
    {
        var e = GetEntity(authoring, TransformUsageFlags.NonUniformScale | TransformUsageFlags.Dynamic);


        AddComponent(e, new GameProperties
        {
            xGridCount = authoring.xGridCount,
            zGridCount = authoring.zGridCount,
            baseOffset = authoring.baseOffset,
            xPadding = authoring.xPadding,
            zPadding = authoring.zPadding,
            gridPrefab = GetEntity(authoring.gridPrefab, TransformUsageFlags.Dynamic),


            teamBlueUnitPostion = authoring.teamBlueUnitPostion,
            teamRedUnitPostion = authoring.teamRedUnitPostion,
            teamBlueUnitPrefab = GetEntity(authoring.teamBlueUnitPrefab, TransformUsageFlags.Dynamic),
            teamRedUnitPrefab = GetEntity(authoring.teamRedUnitPrefab, TransformUsageFlags.Dynamic)
        });
    }
}
```

GameDataAspect.cs

```
public readonly partial struct GameDataAspect : IAspect
{
    private readonly RefRW<GameProperties> _gridProperties;

```

```
    public float3 teamBlueUnitPostion => _gridProperties.ValueRO.teamBlueUnitPostion;
    public float3 teamRedUnitPostion => _gridProperties.ValueRO.teamRedUnitPostion;

    public Entity teamBlueUnitPrefab => _gridProperties.ValueRO.teamBlueUnitPrefab;
    public Entity teamRedUnitPrefab => _gridProperties.ValueRO.teamRedUnitPrefab;



    public int xGridCount => _gridProperties.ValueRO.xGridCount;
    public int zGridCount => _gridProperties.ValueRO.zGridCount;
    public float xPadding => _gridProperties.ValueRO.xPadding;
    public float zPadding => _gridProperties.ValueRO.zPadding;
    public float baseOffset => _gridProperties.ValueRO.baseOffset;
    public Entity gridPrefab => _gridProperties.ValueRO.gridPrefab;


    public LocalTransform GetteamBlueSpawnPoint()
    {
        var position = _gridProperties.ValueRW.teamBlueUnitPostion;
        return new LocalTransform
        {
            Position = position,
            Rotation = quaternion.identity,
            Scale = 1f
        };
    }
    public LocalTransform GetteamRedSpawnPoint()
    {
        var position = _gridProperties.ValueRW.teamRedUnitPostion;
        return new LocalTransform
        {
            Position = position,
            Rotation = quaternion.identity,
            Scale = 1f
        };
    }
}
```

And the final script is the system that contains all previous data together and uses it as spawning units.

SpawnSystem.cs

```
public partial class SpawnSystem : SystemBase
{
    private EntityManager entityManager;
    private Entity unitEntity;
    private Entity EntityCorrdinates;
    public bool isChange = false;
    public int lineupIndex = 0;
    public List<Entity> Lineups = new List<Entity>();

    [BurstCompile]
    protected override void OnUpdate()
    {

        if (LevelManager.Instance.StartTheLevel)
```

```
        {
            entityManager = World.DefaultGameObjectInjectionWorld.EntityManager;
            Lineups = new List<Entity>();
            lineupIndex = 0;

            Entities
            .WithAll<GameProperties>()
            .ForEach(
            (Entity entity) =>
            {
                unitEntity = entity;
            }
            )
            .WithoutBurst()
            .Run();

            var unit = SystemAPI.GetAspect<GameDataAspect>(unitEntity);

            //add lineups to the list
            Entities
            .WithAll<Lineup>()
            .ForEach(
            (Entity entity) =>
            {
                Lineups.Add(entity);
            }
            )
            .WithoutBurst()
            .Run();

            isChange = true;

            //spawn red team on game start
            SpawnRedTeam();
            LevelManager.Instance.StartTheLevel = false;
        }

        if (isChange)
        {
            EntityCorrdinates = Lineups[lineupIndex];
            var ecb = new EntityCommandBuffer(Allocator.Temp);
            var unit = SystemAPI.GetAspect<GameDataAspect>(unitEntity);

            //when choose another lineup previous units will destroy
            Entities
            .WithAll<DestroyTag>()
            .ForEach(
            (Entity entity) =>
            {
                ecb.DestroyEntity(entity);

            }
            )
            .WithoutBurst()
            .Run();
            ecb.Playback(entityManager);

            //spawn blue team
```

```
            SpawnBlueTeam();

            isChange = false;
        }
    }
    public void SpawnRedTeam()
    {

        var ecb = new EntityCommandBuffer(Allocator.Temp);
        var unit = SystemAPI.GetAspect<GameDataAspect>(unitEntity);

        //random lineup for red team
        int randomNum;
        var random = new Random((uint)UnityEngine.Random.Range(0, 100000));
        randomNum = random.NextInt(0, Lineups.Count);

        var gridType = entityManager.GetBuffer<CoordinatesGridBuffer>(Lineups[randomNum]);


        // Instantiate red team ------------------

        for (int k = 0; k < unit.xGridCount; k++)
        {
            for (int j = 0; j < unit.zGridCount; j++)
            {
                var grid = ecb.Instantiate(unit.gridPrefab);
                float3 GridPostion = new float3(k * unit.xPadding, unit.baseOffset, j * unit.zPadding);
                ecb.SetComponent(grid, new LocalTransform { Position = GridPostion, Scale = 0.1f });
                ecb.SetName(grid, "[" + k + "," + j + "]");
                for (int i = 0; i < gridType.Length; i++)
                {
                    if (k == gridType[i].axis.x && j == gridType[i].axis.y)
                    {
                        var newUnit = ecb.Instantiate(unit.teamRedUnitPrefab);
                        ecb.AddComponent<Target>(newUnit);
                        ecb.AddComponent<RedTeam>(newUnit);
                        float3 UnitPostion = new float3(k * unit.xPadding, unit.baseOffset + 1, j * unit.zPa
                        ecb.SetComponent(newUnit, new LocalTransform { Position = UnitPostion, Scale = 1f }
                        ecb.SetName(newUnit, "Red Unit [" + k + "," + j + "]");
                    }
                }
            }
        }
        ecb.Playback(entityManager);


    }
    public void SpawnBlueTeam()
    {

        var unit = SystemAPI.GetAspect<GameDataAspect>(unitEntity);
        var ecb = new EntityCommandBuffer(Allocator.Temp);
        var gridType = entityManager.GetBuffer<CoordinatesGridBuffer>(EntityCorrdinates);

        // Instantiate blue team ------------------

        for (int k = 0; k < unit.xGridCount; k++)
        {
```

```
            for (int j = 0; j < unit.zGridCount; j++)
            {
                var grid = ecb.Instantiate(unit.gridPrefab);
                ecb.AddComponent<DestroyTag>(grid);
                float3 GridPostion = new float3(k * unit.xPadding, unit.baseOffset, j * unit.zPadding);
                ecb.SetComponent(grid, new LocalTransform { Position = GridPostion + unit.GetteamBlueSpawnP
                ecb.SetName(grid, "[" + k + "," + j + "]");
                for (int i = 0; i < gridType.Length; i++)
                {
                    if (k == gridType[i].axis.x && j == gridType[i].axis.y)
                    {
                        var newUnit = ecb.Instantiate(unit.teamBlueUnitPrefab);
                        ecb.AddComponent<Target>(newUnit);
                        ecb.AddComponent<BlueTeam>(newUnit);
                        ecb.AddComponent<DestroyTag>(newUnit);
                        float3 UnitPostion = new float3(k * unit.xPadding, unit.baseOffset + 1, j * unit.zPa
                        ecb.SetComponent(newUnit, new LocalTransform { Position = UnitPostion + unit.Gettear
                        ecb.SetName(newUnit, "Blue Unit [" + k + "," + j + "]");
                    }
                }
            }
        }
        ecb.Playback(entityManager);
    }
}
```

Definition of Each Team: The team list is populated by including all units and assigning them the appropriate team tag. This ensures that every unit is associated with either the BlueTeam or RedTeam in the team list. when the buttle start I run this code one time.

AssignTargetSystem.cs

```
// add each teams to list ------
            if (StartButtel)
            {

                Entities
                .WithAll<RedTeam>()
                .ForEach(
                (Entity entity, ref LocalTransform localTransform) =>
                {
                    redTeam.Add(entity);
                    OnSpawn?.Invoke(localTransform.Position, entity);

                }
                )
                .WithoutBurst()
                .Run();


                Entities
                .WithAll<BlueTeam>()
                .ForEach(
                    (Entity entity, ref LocalTransform localTransform) =>
                    {
                        blueTeam.Add(entity);
```

```
                        OnSpawn?.Invoke(localTransform.Position, entity);
                    }
                )
                .WithoutBurst()
                .Run();

                StartButtel = false;
            }
            //------------
```

Find Target

After spawning, each unit searches for a target by adding an entity value. If the target is destroyed, it will be randomly replaced with a new entity. also when the target die will enter this code aging and find another target.

AssignTargetSystem.cs

```
//check if unit found targert foreach teams -----

            Entities
            .WithAll<RedTeam>()
            .ForEach((Entity entity, ref LocalTransform localTransform, ref Target Target) =>
            {
                if (blueTeam.Count > 0 && Target.targetEntity == Entity.Null && !entityManager.HasComponent<
                {
                    var random = new Unity.Mathematics.Random((uint)UnityEngine.Random.Range(0, 100000));
                    int rand = random.NextInt(0, blueTeam.Count);
                    if (!entityManager.HasComponent<EntityDestroy>(blueTeam[rand]))
                    {
                        Target.targetEntity = blueTeam[rand];
                        ecb.SetComponent(entity, new Target { targetEntity = blueTeam[rand] });
                    }
                }
            }
            )
            .WithoutBurst()
            .Run();


            Entities
            .WithAll<BlueTeam>()
            .ForEach((Entity entity, ref LocalTransform localTransform, ref Target target) =>
                {

                    if (!entityManager.HasComponent<EntityDestroy>(target.targetEntity) && redTeam.Count > 0
                    {
                        var random = new Unity.Mathematics.Random((uint)UnityEngine.Random.Range(0, 100000));
                        int rand = random.NextInt(0, redTeam.Count);
                        if (!entityManager.HasComponent<EntityDestroy>(redTeam[rand]))
                        {
                            target.targetEntity = redTeam[rand];
                            ecb.SetComponent(entity, new Target { targetEntity = redTeam[rand] });
                        }
                    }

                }
```

```
            )
            .WithoutBurst()
            .Run();

            //----------

            ecb.Playback(entityManager);
```

Teams Configuration File

For each team, a configuration file containing their properties.

UnitConfiguration.cs

```
[CreateAssetMenu(fileName = "TeamUnitsData",
 menuName = "Scriptable Object/Team Units Data",
  order = 0)]

public class UnitConfiguration : ScriptableObject
{

    public float HP;
    public float AttackDamage;
    public float InitialAttackSpeed;
    public float AttackRange;
    public float MovementSpeed;

    // This is a util method use for baking process.
    public void ConvertToUnManaged(ref UnitProperties data, ref Health hp)
    {
        hp.HP = HP;
        data.AttackDamage = AttackDamage;
        data.InitialAttackSpeed = InitialAttackSpeed;
        data.AttackRange = AttackRange;
        data.MovementSpeed = MovementSpeed;
    }
}

```

UnitProperties.cs

```
 public struct UnitProperties : IComponentData
{
    public float AttackDamage;
    public float InitialAttackSpeed;
    public float UpdatedAttackSpeed;

    public float AttackRange;
    public float MovementSpeed;

}

public struct Health : IComponentData
{
    public float HP;

```

```
}
```

UnitConfigAuthoring.cs

```
public class UnitConfigAuthoring : MonoBehaviour
{
    [SerializeField]
    private UnitConfiguration unitConfiguration;

    private class UnitConfigBaker : Baker<UnitConfigAuthoring>
    {
        public override void Bake(UnitConfigAuthoring authoring)
        {
            var entity = GetEntity(TransformUsageFlags.None);
            var playerConfiguration = new UnitProperties();
            var playerConfigurationHP = new Health();


            // we will use the method that we already create before to parsing data into the component
            authoring.unitConfiguration.ConvertToUnManaged(ref playerConfiguration, ref playerConfigurationHP


            // Add configuration to entity

            UnitProperties unitProperties = default;
            Health helth = default;

            helth.HP = authoring.unitConfiguration.HP;

            unitProperties.AttackDamage = authoring.unitConfiguration.AttackDamage;
            unitProperties.InitialAttackSpeed = authoring.unitConfiguration.InitialAttackSpeed;
            unitProperties.AttackRange = authoring.unitConfiguration.AttackRange;
            unitProperties.MovementSpeed = authoring.unitConfiguration.MovementSpeed;
            AddComponent(entity, unitProperties);
            AddComponent(entity, helth);

            DependsOn(authoring.unitConfiguration);

        }
    }
}
```

Movement and Attack

If a unit has a target, it will move towards the target until it reaches the specified range. the attack will commence, resulting in a reduction of the target's HP by the unit's attack damage. The attack speed operates similarly to a damage delay loop.

If the target's HP reaches zero, several actions will be performed before destroying the entity. Firstly, the target entity will be saved, and then it will be set to `Entity.Null` Subsequently, the tag `EntityDestroy` will be added.

MoveAndDamageSystem.cs

```
public partial class MoveAndDamageSystem : SystemBase
{
    //for UI display
    public Action<float, float3, Entity> OnDealDamage;
    public Action<float3, Entity> OnMove;
```

```
6
7       [BurstCompile]
8       protected override void OnUpdate()
9       {
10
11          if (UI.Instance.startGame)
12          {
13
14              EntityCommandBuffer ecb = new EntityCommandBuffer(Allocator.Temp);
15
16
17              //blue team -------------
18
19              Entities
20              .WithAll<BlueTeam>()
21              .WithNone<EntityDestroy>()
22              .ForEach((Entity entity, ref Target Target, ref LocalTransform localTransform, ref UnitPropertie
23                  {
24
25                      //movement function -------
26
27                      if (Target.targetEntity != Entity.Null)
28                      {
29                          LocalTransform trgetTransform = World.DefaultGameObjectInjectionWorld.EntityManager
30                          var dist = math.distance(trgetTransform.Position, localTransform.Position);
31
32                          if (dist > blueTeamUnitProperties.AttackRange)
33                          {
34                              if (trgetTransform.Position.x > localTransform.Position.x)
35                              {
36                                  localTransform.Position.x += blueTeamUnitProperties.MovementSpeed * SystemAP
37                              }
38                              else
39                              {
40                                  localTransform.Position.x -= blueTeamUnitProperties.MovementSpeed * SystemAP
41                              }
42
43                              if (trgetTransform.Position.z > localTransform.Position.z)
44                              {
45                                  localTransform.Position.z += blueTeamUnitProperties.MovementSpeed * SystemAP
46                              }
47                              else
48                              {
49                                  localTransform.Position.z -= blueTeamUnitProperties.MovementSpeed * SystemAP
50                              }
51
52                              OnMove?.Invoke(localTransform.Position, entity);
53
54                          }
55                          else
56                          {
57                              //Attack function -----
58
59                              var tagetHp = World.DefaultGameObjectInjectionWorld.EntityManager.GetComponentDa
60                              var tagetPostion = World.DefaultGameObjectInjectionWorld.EntityManager.GetCompon
61
62                              if (tagetHp.HP > 0)
63                              {
```

```
                                if (blueTeamUnitProperties.UpdatedAttackSpeed >= 0)
                                {
                                    blueTeamUnitProperties.UpdatedAttackSpeed -= SystemAPI.Time.DeltaTime;
                                }
                                else
                                {
                                    float hp = tagetHp.HP - blueTeamUnitProperties.AttackDamage;
                                    Debug.Log(Target.targetEntity + " have health: " + hp);
                                    ecb.SetComponent(Target.targetEntity, new Health { HP = hp });
                                    blueTeamUnitProperties.UpdatedAttackSpeed = blueTeamUnitProperties.Initi
                                    OnDealDamage?.Invoke(hp, tagetPostion.Position, Target.targetEntity);

                                }
                            }
                            else
                            {
                                //before add EntityDestroy tag find to the entity new target ----

                                if (Target.targetEntity != Entity.Null)
                                {
                                    var tt = Target.targetEntity;
                                    Target.targetEntity = Entity.Null;
                                    ecb.AddComponent<EntityDestroy>(tt);
                                }
                            }

                        }
                    }
                })
                .WithoutBurst()
                .Run();



            //red team ------------

            Entities
            .WithAll<RedTeam>()
            .WithNone<EntityDestroy>()
            .ForEach((Entity entity, ref Target Target, ref LocalTransform localTransform, ref UnitPropertie
               {
                   if (Target.targetEntity != Entity.Null)
                   {
                       LocalTransform trgetTransform = World.DefaultGameObjectInjectionWorld.EntityManager.G

                       var dist = math.distance(trgetTransform.Position, localTransform.Position); // distan
                       if (dist > redTeamUnitProperties.AttackRange) // moving every axis seperatly to the t
                       {
                           if (trgetTransform.Position.x > localTransform.Position.x)
                           {
                               localTransform.Position.x += redTeamUnitProperties.MovementSpeed * SystemAPI
                           }
                           else
                           {
                               localTransform.Position.x -= redTeamUnitProperties.MovementSpeed * SystemAPI
                           }

                           if (trgetTransform.Position.z > localTransform.Position.z)
```

```
                        {
                            localTransform.Position.z += redTeamUnitProperties.MovementSpeed * SystemAPI
                        }
                        else
                        {
                            localTransform.Position.z -= redTeamUnitProperties.MovementSpeed * SystemAPI
                        }

                        OnMove?.Invoke(localTransform.Position, entity);

                    }
                    else
                    {
                        var tagetHp = World.DefaultGameObjectInjectionWorld.EntityManager.GetComponentDat
                        var tagetPostion = World.DefaultGameObjectInjectionWorld.EntityManager.GetCompone


                        if (tagetHp.HP > 0)
                        {
                            if (redTeamUnitProperties.UpdatedAttackSpeed >= 0)
                            {
                                redTeamUnitProperties.UpdatedAttackSpeed -= SystemAPI.Time.DeltaTime;
                            }
                            else
                            {
                                float hp = tagetHp.HP - redTeamUnitProperties.AttackDamage;
                                OnDealDamage?.Invoke(hp, tagetPostion.Position, Target.targetEntity);


                                Debug.Log(Target.targetEntity + " have health: " + hp);
                                ecb.SetComponent(Target.targetEntity, new Health { HP = hp });
                                redTeamUnitProperties.UpdatedAttackSpeed = redTeamUnitProperties.InitialA
                            }
                        }
                        else
                        {
                            if (Target.targetEntity != Entity.Null)
                            {
                                var tt = Target.targetEntity;
                                Target.targetEntity = Entity.Null;
                                ecb.AddComponent<EntityDestroy>(tt);
                            }
                        }
                    }
                }
            }
        )
        .WithoutBurst()
        .Run();

         ecb.Playback(World.DefaultGameObjectInjectionWorld.EntityManager);
    }

}

}
```

Destroy Target

Identify all entities that possess the `EntityDestroy` tag, will remove them from the team list, and subsequently proceed with their destruction.

AssignTargetSystem.cs

```
// destroy unit if has EntityDestroy
           Entities
           .WithAll<EntityDestroy>()
           .WithAll<BlueTeam>()
           .ForEach((Entity entity) =>
              {
                  blueTeam.Remove(entity);
                  ecb.DestroyEntity(entity);
              }
           )
           .WithoutBurst()
           .Run();

           Entities
          .WithAll<EntityDestroy>()
          .WithAll<RedTeam>()
          .ForEach((Entity entity) =>
              {
                  redTeam.Remove(entity);
                  ecb.DestroyEntity(entity);
              }
          )
          .WithoutBurst()
          .Run();

           //--------------
```

Display to UI

Health: for health, I use World Space Canvas and I use Action. first, I call to action to spawn the text in `AssignTargetSystem.cs` `public Action<float3, Entity> OnSpawn;` and I invoke it when the game starts and the player chooses their lineup. To detect the increase and decrees and position for the health text, I add `public Action<float, float3, Entity> OnDealDamage;` and `public Action<float3, Entity> OnMove;` in script `MoveAndDamageSystem.cs`

WorldSpaceUIController.cs

```
public class WorldSpaceUIController : MonoBehaviour
{
    [SerializeField] private GameObject _damageIconPrefab;
    [SerializeField] private List<TMP_Text> textList = new List<TMP_Text>();
    private Transform _mainCameraTransform;
    private EntityManager entityManager;

    private void Start()
    {
        entityManager = World.DefaultGameObjectInjectionWorld.EntityManager;
        _mainCameraTransform = Camera.main.transform;
    }
```

```
13
14      private void OnEnable()
15      {
16          var dealDamageSystem = World.DefaultGameObjectInjectionWorld.GetExistingSystemManaged<MoveAndDamageSy
17          var spawnSystem = World.DefaultGameObjectInjectionWorld.GetExistingSystemManaged<AssignTargetSystem>(
18
19          spawnSystem.OnSpawn += DisplayHP;
20
21          dealDamageSystem.OnDealDamage += DisplayDamageIcon;
22          dealDamageSystem.OnMove += Move;
23      }
24
25      private void OnDisable()
26      {
27          if (World.DefaultGameObjectInjectionWorld == null) return;
28          var dealDamageSystem = World.DefaultGameObjectInjectionWorld.GetExistingSystemManaged<MoveAndDamageSy
29          var spawnSystem = World.DefaultGameObjectInjectionWorld.GetExistingSystemManaged<AssignTargetSystem>(
30
31          spawnSystem.OnSpawn -= DisplayHP;
32
33          dealDamageSystem.OnDealDamage -= DisplayDamageIcon;
34          dealDamageSystem.OnMove -= Move;
35      }
36
37      private void DisplayDamageIcon(float damageAmount, float3 startPosition, Entity entity)
38      {
39          foreach (TMP_Text text in textList)
40          {
41              if (entityManager.GetName(entity) == text.gameObject.name)
42              {
43                  text.text = $"<color=black>{damageAmount.ToString()}</color>";
44
45                  //if the health rach 0 make the text empty
46                  if (damageAmount <= 0)
47                  {
48                      text.text = "";
49                  }
50              }
51          }
52      }
53
54      private void DisplayHP(float3 startPosition, Entity entity)
55      {
56          var directionToCamera = (Vector3)startPosition - _mainCameraTransform.position;
57          var rotationToCamera = Quaternion.LookRotation(directionToCamera, Vector3.up);
58          var newIcon = Instantiate(_damageIconPrefab, new Vector3(startPosition.x, startPosition.y + 1.5f, sta
59          var newIconText = newIcon.GetComponent<TextMeshProUGUI>();
60          textList.Add(newIconText);
61          newIconText.name = entityManager.GetName(entity);
62          newIconText.text = $"<color=black>{100}</color>";
63      }
64
65      private void Move(float3 Position, Entity entity)
66      {
67          foreach (TMP_Text text in textList)
68          {
69              if (entityManager.GetName(entity) == text.gameObject.name)
70              {
```

```
                text.transform.position = new Vector3(Position.x, Position.y + 1.5f, Position.z);
            }
        }
    }
}
```

Start Screen: for the start screen I added scroll buttons and each button has added a listener for the lineup index, to adjust many lineups I will explain this in the video.
and a start button that has a listener to start the game in this data.

UI.cs

```
public class UI : MonoBehaviour
{
    [SerializeField] Button startButton;
    public GameObject buttonPrefab;
    public GameObject ScrollContent;
    public bool startGame = false;
    public int numbersOfLineups;
    private static UI _instance;
    public static UI Instance { get { return _instance; } }
    private void Awake()
    {
        if (_instance != null && _instance != this)
        {
            Destroy(gameObject);
        }
        else
        {
            _instance = this;
        }
    }

    // Start is called before the first frame update
    void Start()
    {
        startButton.onClick.AddListener(StartGame);

        for (int i = 0; i < numbersOfLineups; i++)
        {
            GameObject buttonObject = Instantiate(buttonPrefab, new Vector3(0, 0, 0), Quaternion.identity);
            buttonObject.transform.parent = ScrollContent.transform;
            buttonObject.transform.position = new Vector3(0, 0, 0);
            buttonObject.transform.localScale = ScrollContent.transform.localScale;
            buttonObject.GetComponentInChildren<TMP_Text>().text = "Team " + (i + 1);
            int cachedIndex = i;
            Button button = buttonObject.GetComponent<Button>();

            button.onClick.AddListener(delegate { ChooseLineup(cachedIndex); });
        }

    }

    void ChooseLineup(int _index)
    {
```

```
        Debug.Log("" + _index);
        var dealDamageSystem33 = World.DefaultGameObjectInjectionWorld.GetExistingSystemManaged<SpawnSystem>(
        dealDamageSystem33.isChange = true;
        dealDamageSystem33.lineupIndex = _index;

    }
    void StartGame()
    {
        startGame = true;
        Debug.Log(startGame);
        startButton.gameObject.SetActive(false);
    }

}
```

End Screen & Reload Scene: In the end screen, I created a `LevelManager.cs` that loading a subscene when reloading.

LevelManager.cs

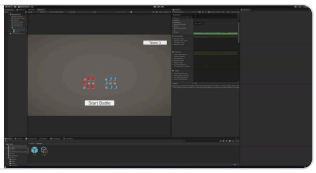
```
public class LevelManager : MonoBehaviour
{
    public SubScene subScenes;
    public bool StartTheLevel = false;
    Entity SceneEntity;
    bool endLoad;

    public static LevelManager Instance { get; private set; }
    private void Awake()
    {
        Instance = this;
        DontDestroyOnLoad(this);
    }
    void Start()
    {
        SceneEntity = SceneSystem.GetSceneEntity(World.DefaultGameObjectInjectionWorld.Unmanaged, subScenes.
        endLoad = false;
    }
    void Update()
    {
        if (SceneSystem.IsSceneLoaded(World.DefaultGameObjectInjectionWorld.Unmanaged, SceneEntity) && !endL
        {
            StartTheLevel = true;
            Debug.Log("the subscene loaded");
            endLoad = true;
        }
    }
    public void RestartGame()
    {
        var entityManager = World.DefaultGameObjectInjectionWorld.EntityManager;
        entityManager.DestroyEntity(entityManager.UniversalQuery);
        SceneManager.LoadScene(SceneManager.GetActiveScene().name, LoadSceneMode.Single);
    }
}
```

Battle Simulator - How to create Lineups

Created by Wedad Almehmadi • Updated 6 minutes ago

YouTube

Open preview



Battle Simulator - Game Overview

Created by Wedad Almehmadi • Updated 20 hours ago

YouTube

Open preview

Conclusion

I would like to conclude my completion of this project by thanking you for your guidance and time. It was a unique experience from which I learned a lot and mastered many things that will benefit me in my career and programming hobby.

I really hope to move forward with you to success.