Battle Simulator
Unity ECS

This is a detailed documentation that showcases the installation guide for the Battle Simulator game made with unity.The unity game is made with ECS to accommodate a data oriented approach and thus is created using Unity DOTS and ECS.

Installation GUIDE

- Install Unity 2023.3.0f1 LTS in your machine.
- Download the unity project from Unity_Project Folder.
- From the Unity Hub locate the game files and add that to the unity hub.
- Launch the project.
- Make sure the project runs in the URP.
- Open scene MainGame by navigating inside the scene folder in unity project tab.
- There are two packages added to the game. These are necessary packages and require additional steps to add them to the game.
    a. Go to the package manager by navigation to Window-> Package Manager.
    b. From the top left click on add icon "+" and press add package by name.
    c. Write com.unity.Entites to add all the entities and necessary packages.
    d. Now the project will work but when we play the game we are unable to see any entities on the scene. To render this we need to add the Entites Grahphics package.
    e. Following step a and b write com.unity.entities.graphics to add the entities graphics package.
    f. Now we are able to see the entities from the subscene properly.
- The playable APK is located in the binary folder.
- Add that to your phone for installation of the .apk file.

Technical Documentation

Five System classes are created and inherited from the SystemBase class in unity. These classes are responsible for handling most of the crucial mechanisms.

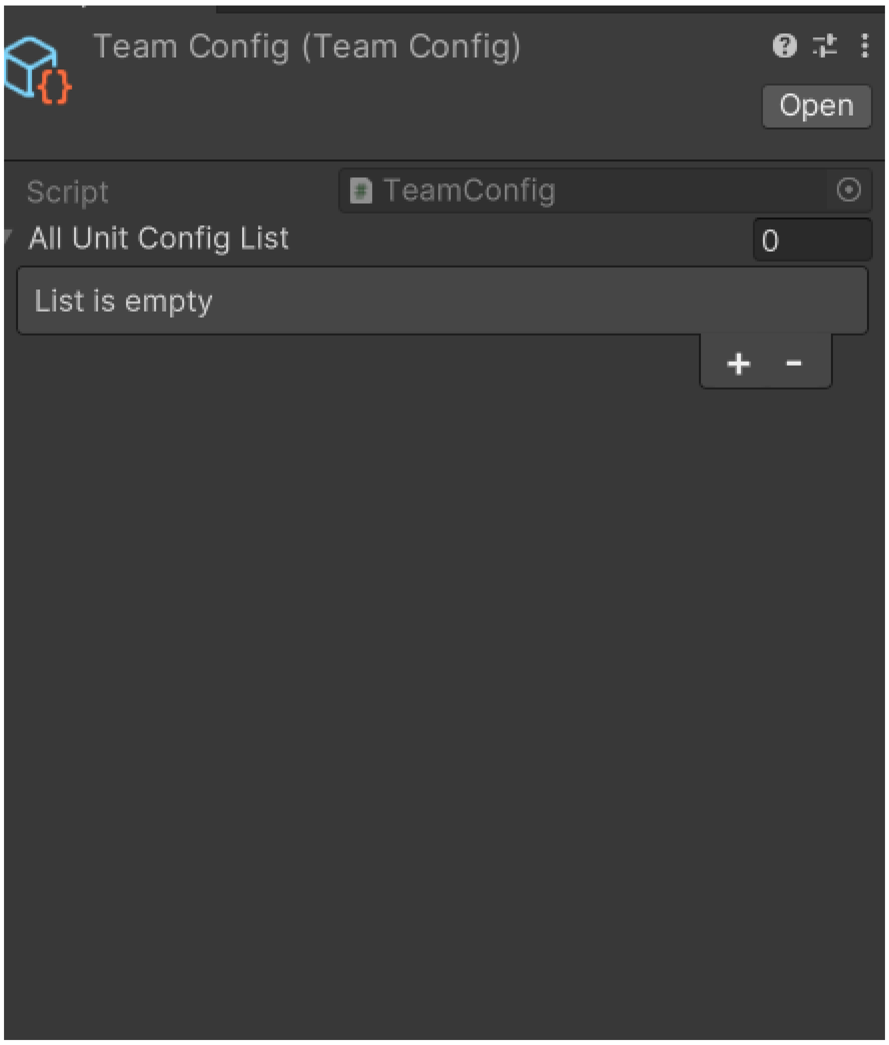
System class

- Enemy Finder System
    - This system goes through all the units within the game and determines if a unit has a target or not. Based on this information it adds a target from the available targets. Likewise when a target is killed this system removes that as target and assigns a new target to the unit.
- Game System
    - This system is a core handler in the game and binds directly to the HUDView monobehaviour and gets updated based on the button press event. Thus this system handles most of the game logic as starting the game. A managed component class derived from IComponentData is used and is available across all the systems. Thus when this data is updated it is reflected across all the systems allowing each system to update based on this class data.
- Player Movement System
    - This system handles all the movement of the units based on the targets of the unit. Thus this class determines the target location and moves the player towards the target. This system also is responsible for destroying any unit having health less or equal to zero.
- Spawn Unit System
    - This system is responsible for spawning units within the game. When an enemy team is changed this class despawn all the units and spawns new unit with required changes,
- Unit Attack System
    - This system is responsible for attacking the units. Each unit while it is alive will have a target. Once it reaches a position from where the unit can attack the target it starts attacking and deducing health points on the target until the target is killed or the unit itself dies. In case it kills the target the Enemy finder system will find a new target to the unit.

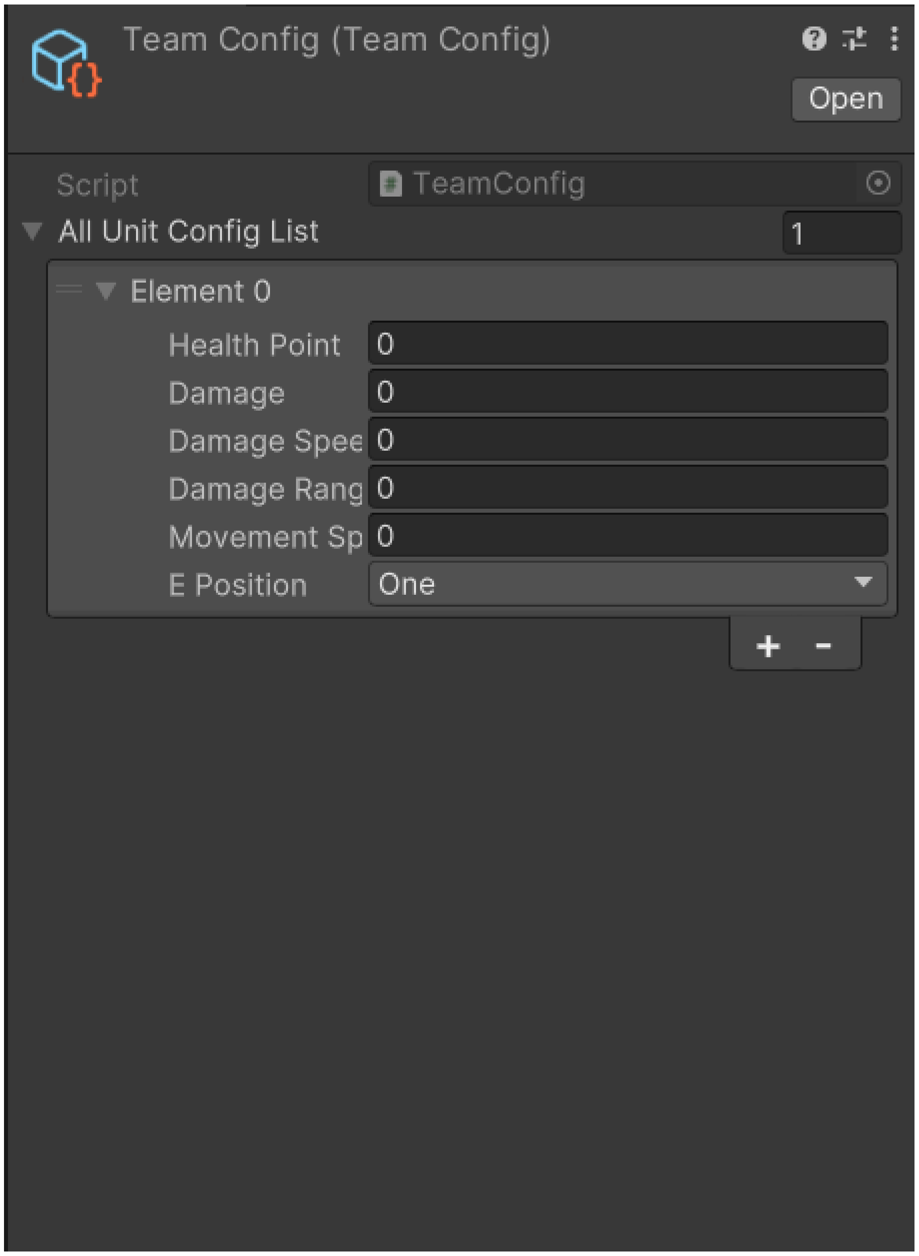
Each of this system uses collection of Component data to handle all the related data of each unit. This includes some managed Data to handle TextMeshPro as it is still not suitable in an ECS environment and some unmanaged Data use for basic data of unit such as health points.

Configuration

The game consists of two teams. Each team can have up to max 9 players. The configuration for this can be done using a scriptable config Object. To generate a scriptable config object right click on the project tab and navigate to Create -> Config -> TeamConfig.



Upon creating a config file, and clicking it, you will see the above scriptable object with an exposed list in the inspector. The number of lists will determine the number of players. Upon adding one object to the list we can see the below configuration file where we can modify each unit value. The E position define the actual position of the player to spawn. Make sure that this is different for each unit.


Team Config (Team Config)

Open

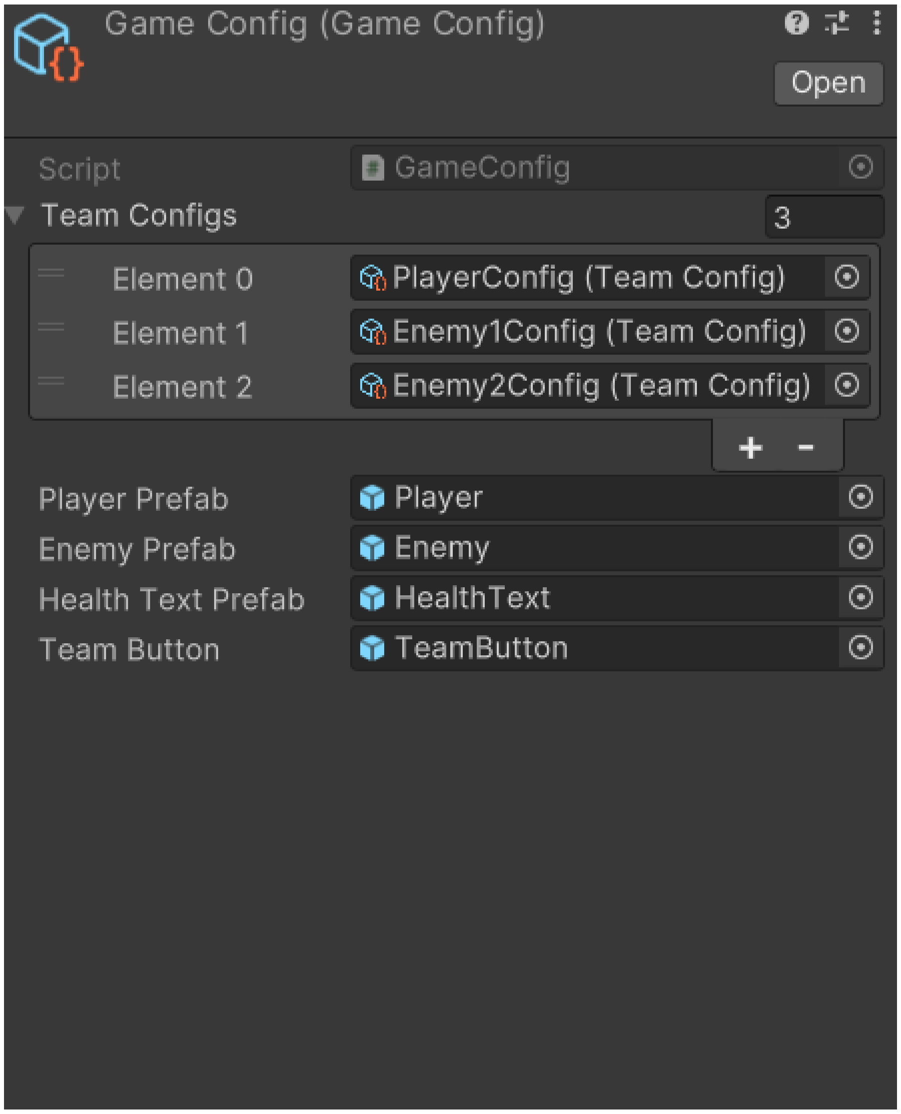
Script: TeamConfig

All Unit Config List: 1

Element 0

Health Point	0
Damage	0
Damage Spee	0
Damage Rang	0
Movement Sp	0
E Position	One

In this way we can create a config file for each team. It is necessary to add these to the GameConfig scriptableobject to so that they are reflected on the game.


Game Config (Game Config)

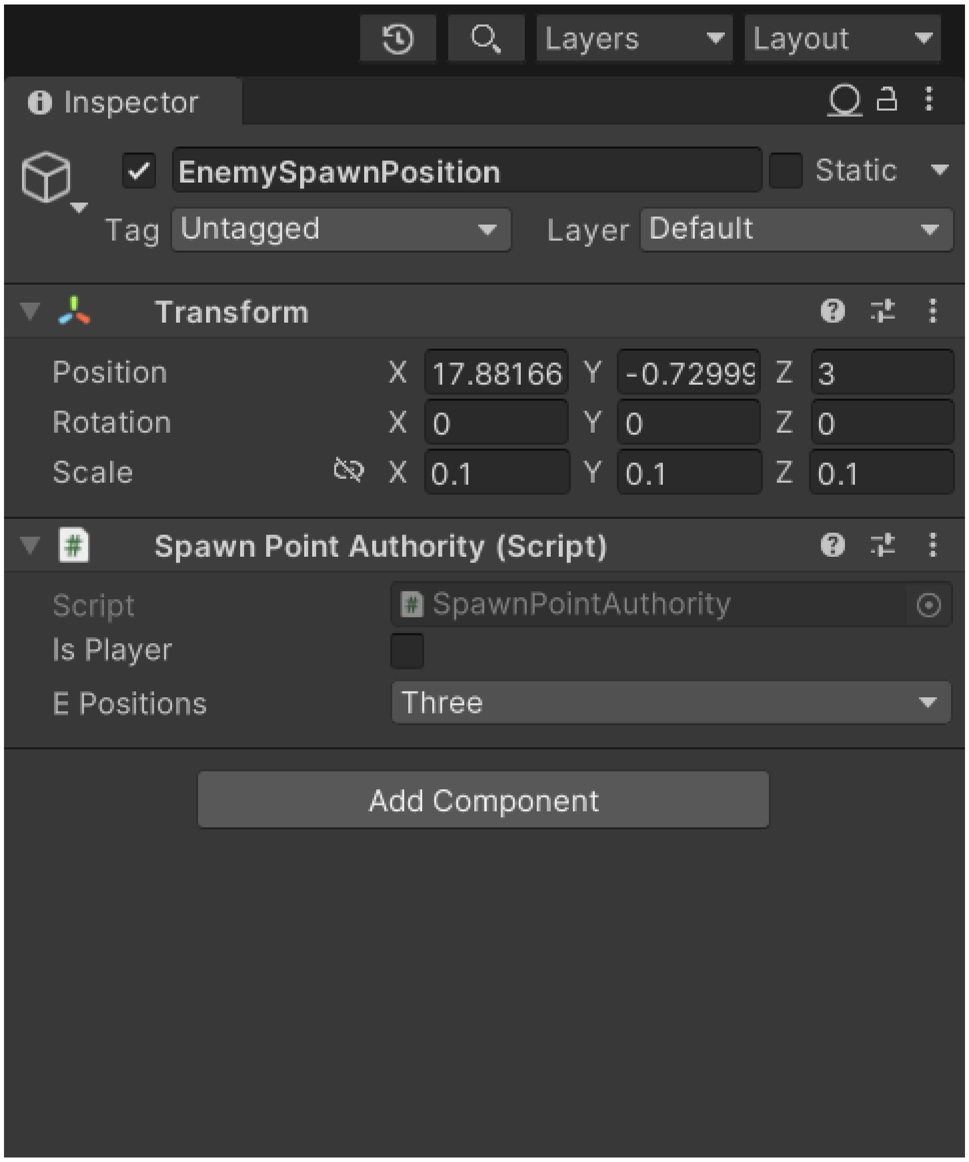
Script	GameConfig
Team Configs	3
Element 0	PlayerConfig (Team Config)
Element 1	Enemy1Config (Team Config)
Element 2	Enemy2Config (Team Config)
Player Prefab	Player
Enemy Prefab	Enemy
Health Text Prefab	HealthText
Team Button	TeamButton

In the team Configs file the first element in list represent player while all the other elements in the list are enemy

Technical Decision:

Baker

Baker is used within the game to convert a GameObject Data into an Entity Component. This allows easy addition of data through unity gameobject and modifies these values through unity inspector.



EnemySpawnPosition
Tag Untagged
Layer Default

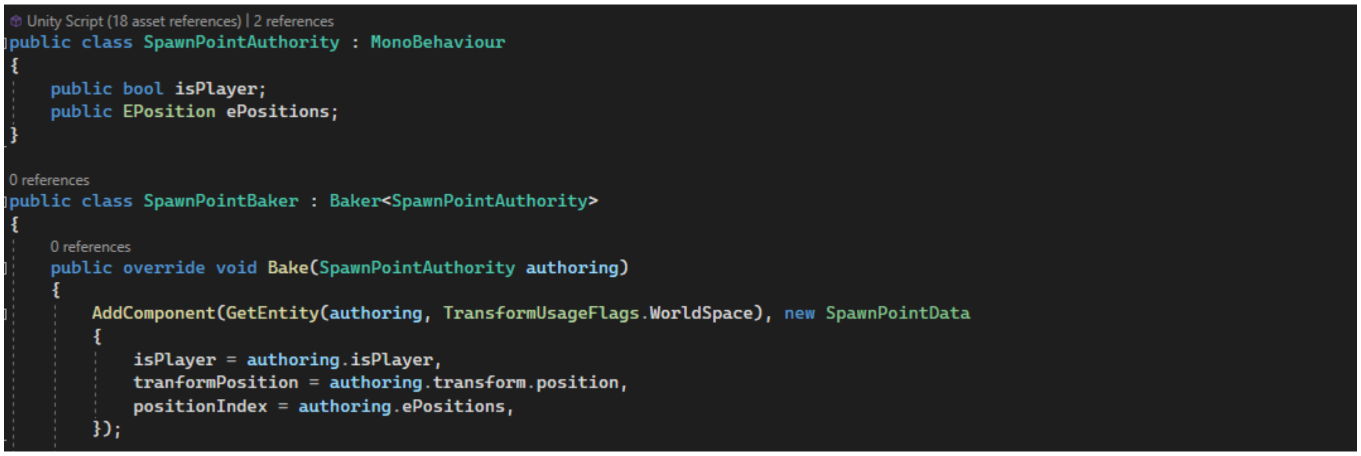
Transform

Position X 17.88166 Y -0.72999 Z 3
Rotation X 0 Y 0 Z 0
Scale X 0.1 Y 0.1 Z 0.1

Spawn Point Authority (Script)

Script SpawnPointAuthority
Is Player
E Positions Three

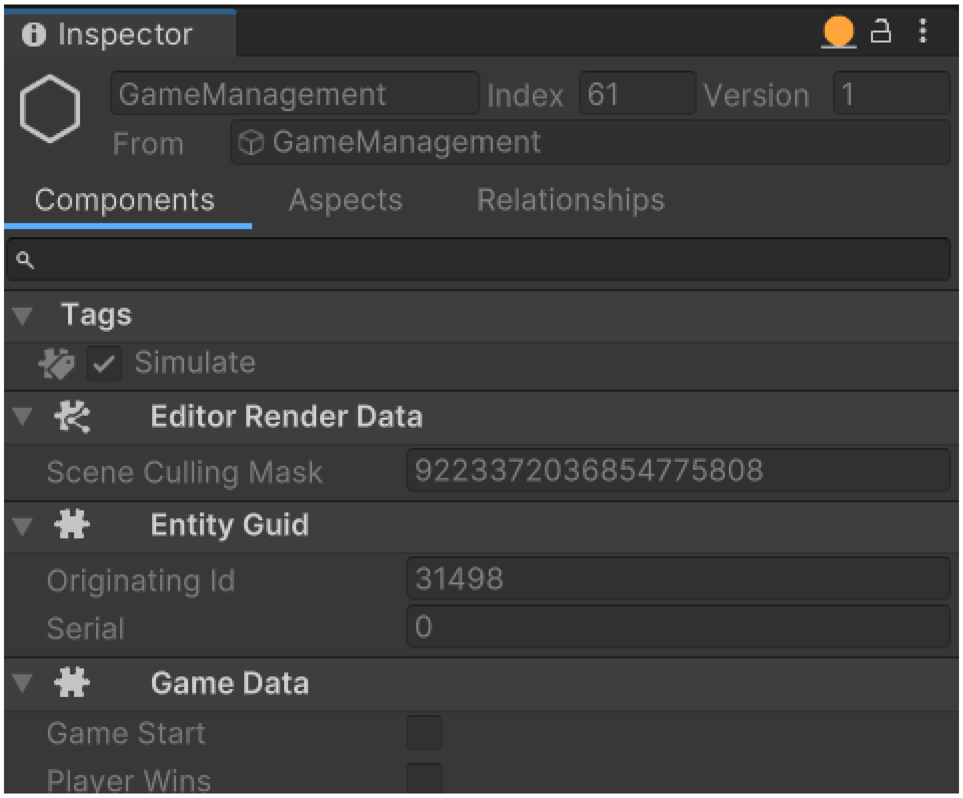
The class spawn point authority is successfully backed into an entity using a baker as shown below. At run time the below SpawnpointBaker class bakes the Spawn Point Authority class data into an entity.



```
Unity Script (18 asset references) | 2 references
public class SpawnPointAuthority : MonoBehaviour
{
    public bool isPlayer;
    public EPosition ePositions;
}

0 references
public class SpawnPointBaker : Baker<SpawnPointAuthority>
{
    0 references
    public override void Bake(SpawnPointAuthority authoring)
    {
        AddComponent(GetEntity(authoring, TransformUsageFlags.WorldSpace), new SpawnPointData
        {
            isPlayer = authoring.isPlayer,
            tranformPosition = authoring.transform.position,
            positionIndex = authoring.ePositions,
        });
```

The gameobject at run time thus is converted to entity as shown below.



Inspector

	GameManagement	Index	61	Version	1
From	GameManagement				

Components | Aspects | Relationships

Tags
- ✓ Simulate

Editor Render Data

Scene Culling Mask	9223372036854775808

Entity Guid

Originating Id	31498
Serial	0

Game Data

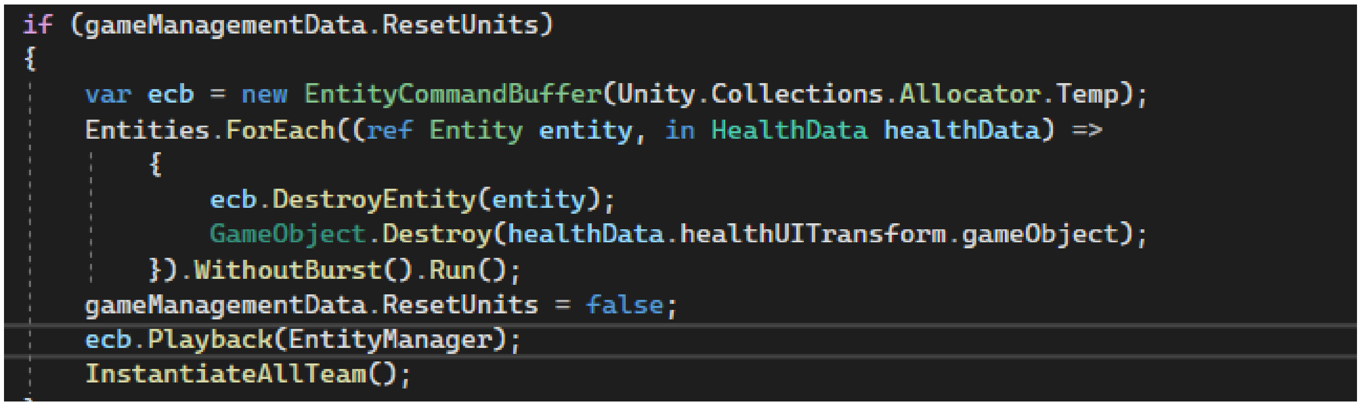
Game Start	☐
Player Wins	☐

Entity Component Buffer

Despite running the task of Instantiating the entity in the main thread, the system doesn't allow automatic structural changes. This could be for several reason including

- Lookup for some component can be invalid once the structural change due to deallocated lookup.

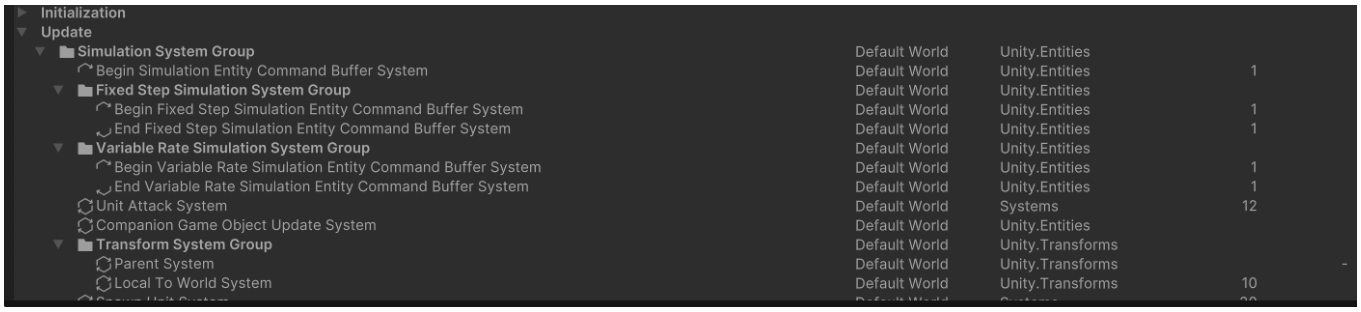
Thus we use Entity Component Buffer and initialize it before the loop and add any structural change on the buffer and playback after the loop.



```
if (gameManagementData.ResetUnits)
{
    var ecb = new EntityCommandBuffer(Unity.Collections.Allocator.Temp);
    Entities.ForEach((ref Entity entity, in HealthData healthData) =>
        {
            ecb.DestroyEntity(entity);
            GameObject.Destroy(healthData.healthUITransform.gameObject);
        }).WithoutBurst().Run();
    gameManagementData.ResetUnits = false;
    ecb.Playback(EntityManager);
    InstantiateAllTeam();
```

Profiling Entity World:

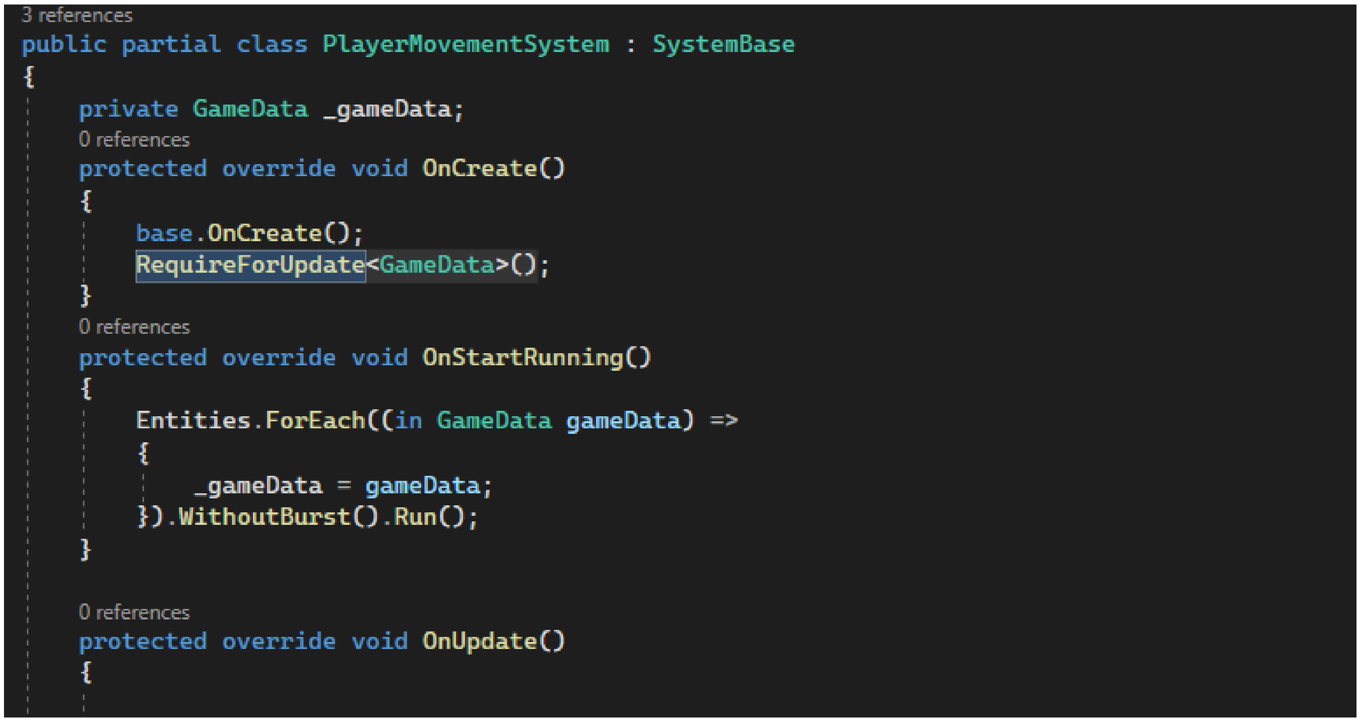
With the use of System Profiling provided by Enitites package. We can simply go the the system profiler tab through Windows->Entities->Profiler and validate all the operation being carried out in Initialized, Update and Pre late Update Loop in ECS system.



```
▶ Initialization
▼ Update
  ▼ Simulation System Group                                Default World   Unity.Entities
      Begin Simulation Entity Command Buffer System        Default World   Unity.Entities      1
    ▼ Fixed Step Simulation System Group                   Default World   Unity.Entities
        Begin Fixed Step Simulation Entity Command Buffer System   Default World   Unity.Entities   1
        End Fixed Step Simulation Entity Command Buffer System     Default World   Unity.Entities   1
    ▼ Variable Rate Simulation System Group                Default World   Unity.Entities
        Begin Variable Rate Simulation Entity Command Buffer System   Default World   Unity.Entities   1
        End Variable Rate Simulation Entity Command Buffer System     Default World   Unity.Entities   1
      Unit Attack System                                   Default World   Systems             12
      Companion Game Object Update System                  Default World   Unity.Entities
    ▼ Transform System Group                               Default World   Unity.Transforms
        Parent System                                      Default World   Unity.Transforms         -
        Local To World System                              Default World   Unity.Transforms    10
```

SystemBase

A managed Class GameData is used to handle the game logic that includes starting the game,restarting the game. Thus we need a managed system to handle the system logic to handle game mechanics such as starting the game and running system updates. Thus all the game systems used a managed Data to determine if the game has started and if so starts the game. Due to this reason all the systems implement SystemBase abstract class. Since there are not many entities in our game there is not much performance difference for not using burst compilers available in ISystem.



```
3 references
public partial class PlayerMovementSystem : SystemBase
{
    private GameData _gameData;
    0 references
    protected override void OnCreate()
    {
        base.OnCreate();
        RequireForUpdate<GameData>();
    }
    0 references
    protected override void OnStartRunning()
    {
        Entities.ForEach((in GameData gameData) =>
        {
            _gameData = gameData;
        }).WithoutBurst().Run();
    }

    0 references
    protected override void OnUpdate()
    {
```

RequireForUpdate<T>() Method

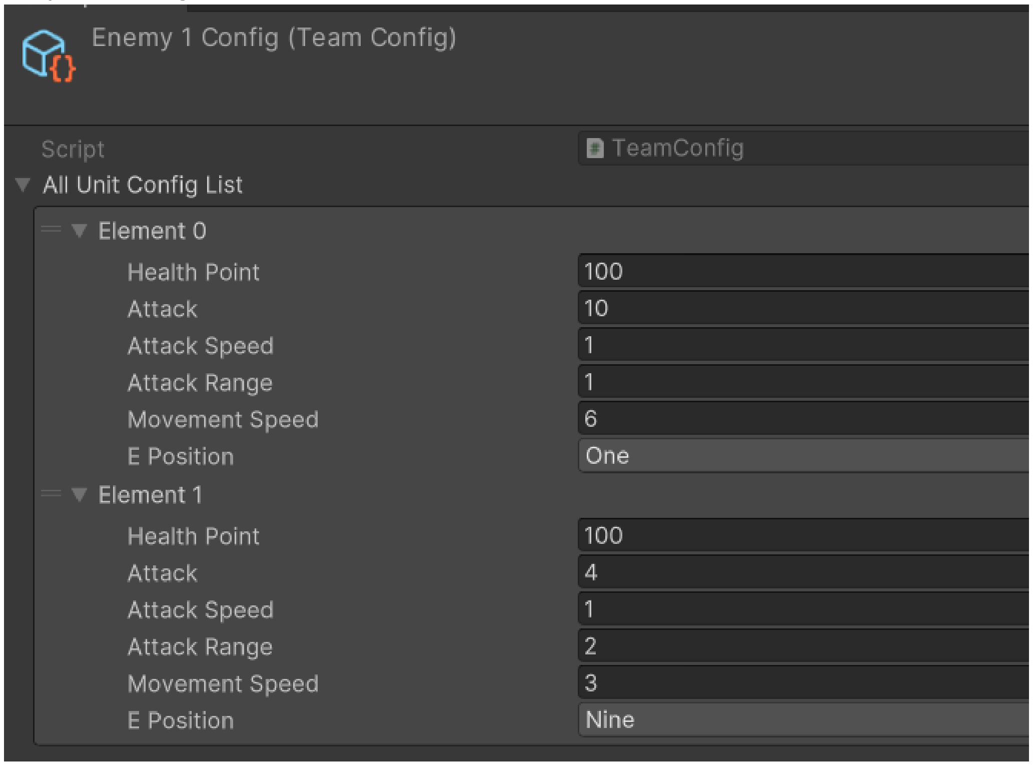
It is necessary to have an entity with a game data component for all systems to work properly. Thus a required update with game data is added to the start of each system to make sure the Game data is properly loaded into its entity before the game starts. This was done as the game was unable to run on some of the build of the game.

Testing

Test Spawning Items

- Test case id : 001

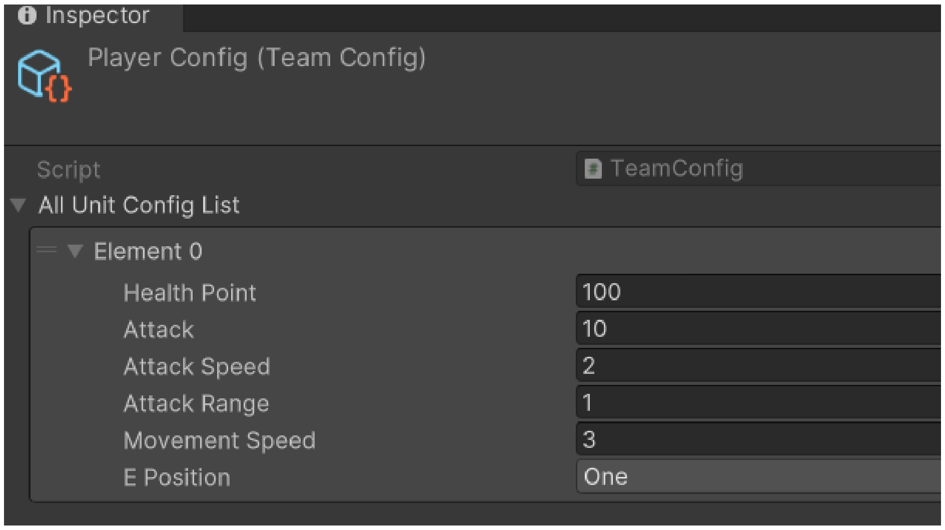
- Unit to test
    - Verify if the units are spawn properly.
- Assumptions
    - All the unit place on system will spawn
- Test data
    - Variables and their values:
        - Player Config File



Enemy 1 Config (Team Config)

Script	TeamConfig
All Unit Config List	
Element 0	
Health Point	100
Attack	10
Attack Speed	1
Attack Range	1
Movement Speed	6
E Position	One
Element 1	
Health Point	100
Attack	4
Attack Speed	1
Attack Range	2
Movement Speed	3
E Position	Nine

- Enemy Config File



Inspector

Player Config (Team Config)

Script	TeamConfig
All Unit Config List	
Element 0	
Health Point	100
Attack	10
Attack Speed	2
Attack Range	1
Movement Speed	3
E Position	One

- Steps to be executed

        - Run the program with two above file
- Expected Result
    - All the unit spawn in their respective position
- Actual result
    - All the unit spawn in their respective position
- Pass/Fail
    - Pass
- Comments
    - When team config files have different positions on each config the items are spawned properly. In case the position overlaps the system will spawn in that position only.

Test Floating Scroll Button

Test case id : 002

- Unit to test
    - Verify if the scroll view works and button will spawn when there are numerous number of enemy team config
- Assumptions
    - All the team config will have buttons
- Test data
    - Variables and their values:
        - 10 Enemy Config files
- Steps to be executed
    - Add each config files to the Game Config
- Expected Result
    - 10 buttons will be added to the scroll view button which can be use to toggle the enemies config.
- Actual result
    - 10 buttons added to the scroll view with capabilities to toggle between enemies
- Pass/Fail
    - Pass
- Comments
    - Even when the config files are the same but added multiple times to the Game Config's teamConfig list the button will be instantiated.
    -
- Note
    - The initial item in the list is player's config.

Conclusion

In Conclusion the game can be highly customized to facilitate more enemies units and can be configured easily. Due to time constraint I was unable to work on achieving multithreading, and do any performance pass or perform any performance technique.as