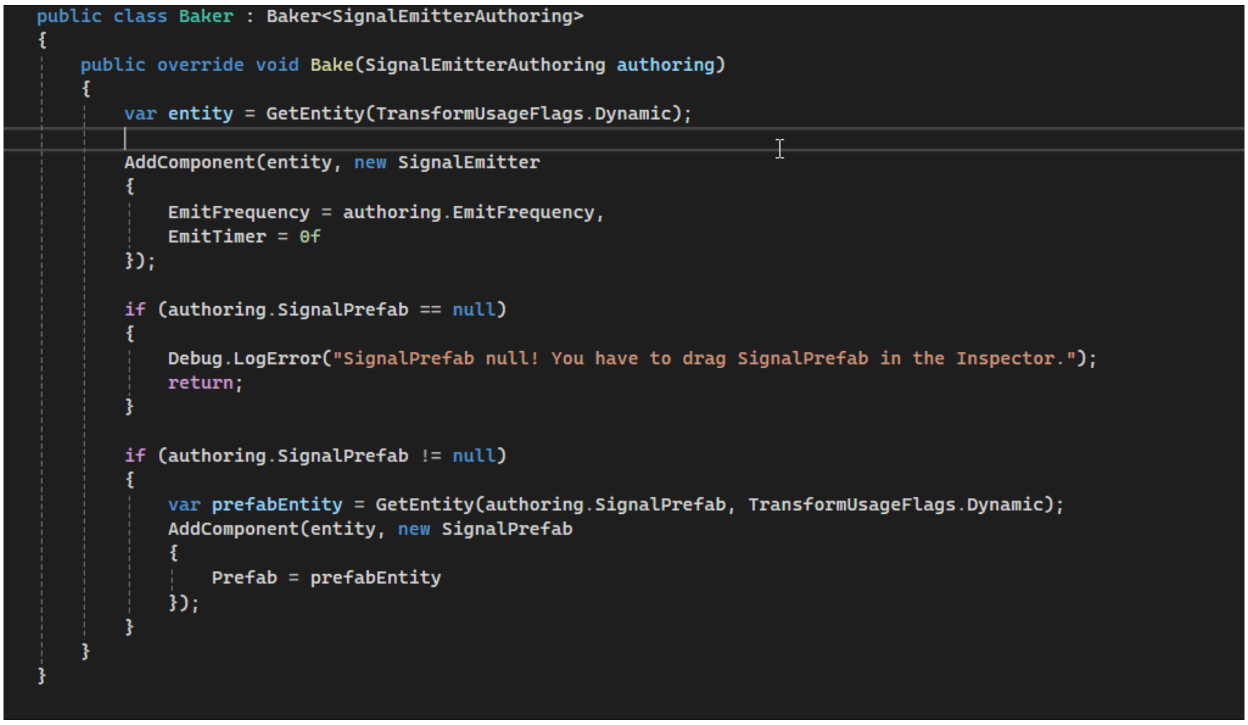
Signal Puzzle Game Documentation

This documentation outlines the implementation of a signal puzzle game using Unity DOTS/ECS framework. The game involves a tower emitting signals, interactable objects (absorbers, reflectors, and state changers), and a victory condition based on maintaining certain states for a defined period.

Core Components

- **Purpose:** Converts a MonoBehaviour to ECS-compatible data for signal emitters.
- **Key Fields:**
  - `EmitFrequency` - Frequency of signal emission.
  - `SignalPrefab` - Prefab for the emitted signal.
- 

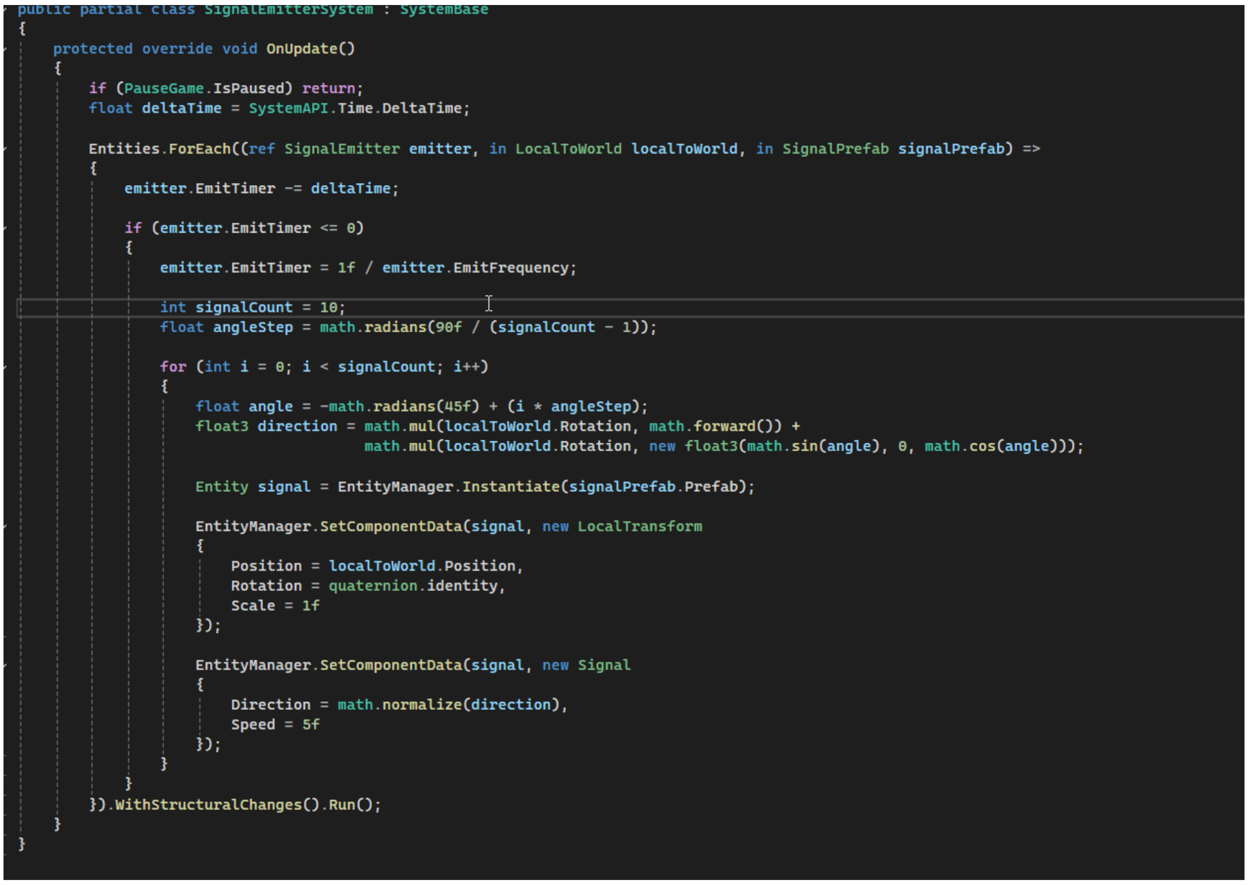
```csharp
public class Baker : Baker<SignalEmitterAuthoring>
{
    public override void Bake(SignalEmitterAuthoring authoring)
    {
        var entity = GetEntity(TransformUsageFlags.Dynamic);

        AddComponent(entity, new SignalEmitter
        {
            EmitFrequency = authoring.EmitFrequency,
            EmitTimer = 0f
        });

        if (authoring.SignalPrefab == null)
        {
            Debug.LogError("SignalPrefab null! You have to drag SignalPrefab in the Inspector.");
            return;
        }

        if (authoring.SignalPrefab != null)
        {
            var prefabEntity = GetEntity(authoring.SignalPrefab, TransformUsageFlags.Dynamic);
            AddComponent(entity, new SignalPrefab
            {
                Prefab = prefabEntity
            });
        }
    }
}
```

- The SignalEmitterSystem controls the behavior of signal emitters in the game:
- Rotates the emitter and emits signals at specified intervals based on its `EmitFrequency`.
- Signals are emitted in an arc, evenly distributed within a 90-degree range.
- Each signal is assigned a direction and speed, determined by its position in the arc.

Key functionality:

1. Decreases the `EmitTimer` for each signal emitter by the time elapsed since the last update.
2. Emits multiple signals in a 90-degree arc when the timer reaches zero, resetting the timer.

3. Creates and initializes new signal entities, setting their position, rotation, scale, direction, and speed.



```
public partial class SignalEmitterSystem : SystemBase
{
    protected override void OnUpdate()
    {
        if (PauseGame.IsPaused) return;
        float deltaTime = SystemAPI.Time.DeltaTime;

        Entities.ForEach((ref SignalEmitter emitter, in LocalToWorld localToWorld, in SignalPrefab signalPrefab) =>
        {
            emitter.EmitTimer -= deltaTime;

            if (emitter.EmitTimer <= 0)
            {
                emitter.EmitTimer = 1f / emitter.EmitFrequency;

                int signalCount = 10;
                float angleStep = math.radians(90f / (signalCount - 1));

                for (int i = 0; i < signalCount; i++)
                {
                    float angle = -math.radians(45f) + (i * angleStep);
                    float3 direction = math.mul(localToWorld.Rotation, math.forward()) +
                                       math.mul(localToWorld.Rotation, new float3(math.sin(angle), 0, math.cos(angle)));

                    Entity signal = EntityManager.Instantiate(signalPrefab.Prefab);

                    EntityManager.SetComponentData(signal, new LocalTransform
                    {
                        Position = localToWorld.Position,
                        Rotation = quaternion.identity,
                        Scale = 1f
                    });

                    EntityManager.SetComponentData(signal, new Signal
                    {
                        Direction = math.normalize(direction),
                        Speed = 5f
                    });
                }
            }
        }).WithStructuralChanges().Run();
    }
}
```

SignalInteractionSystem

-Manages interactions between signals and different game objects:

- Absorbers destroy signals within a defined radius.
- Reflectors bounce signals, altering their direction while keeping them in the x-z plane.
- State changers toggle their state and update their visual color when hit by a signal.

- EntityCommandBuffer:
  - A special object that collects commands to modify entities or components in the ECS world.
    -Structural changes (e.g., adding/removing components, destroying entities) are not allowed directly during the execution of a system's main logic because they can disrupt the ongoing queries. The ECB defers these changes safely.
- SystemAPI.Query<RefRW, RefRO>()
  Uses Unity DOTS/ECS to iterate over a specific set of entities in the ECS world.
  - SystemAPI.Query:
  - Defines a query to retrieve specific components from entities.
  - This query is used to find entities that have the specified components attached.
  - RefRW:
  - Fetches the `Signal` component of the entity with both read and write access, meaning the system can both read from and modify the `Signal` component's

data.

- RefRO:
  - Stands for **Reference Read-Only**.
- Fetches the `LocalTransform` component of the entity with read-only access, meaning the system can read the position/rotation/scale but cannot modify them.
  - Combined Query:
    This query will match entities that have both `Signal` (modifiable) and `LocalTransform` (read-only) components attached.
- Entity Access - .WithEntityAccess()
  - Enables the retrieval of the **Entity** itself, in addition to the components queried.
- The `entity` variable represents the specific entity being processed in the loop, which is useful for operations like adding or removing components, or destroying the entity.
- math.reflect(signalData.Direction, normal):
- The `math.reflect()` function in Unity's **Mathematics** library computes the reflection vector of an incoming vector off a surface defined by its normal vector.
- Parameters
- `direction`: The incoming vector (typically representing motion or direction).
- `normal`: The normal vector of the surface where the reflection occurs. This vector must be normalized (unit length) for accurate results.
- math.distance
  - is a function in Unity's **Mathematics** library that calculates the Euclidean distance between two points in space.
- ecb.Playback(EntityManager) and ecb.Dispose():
  - These lines are part of Unity's **EntityCommandBuffer (ECB)** system, which is used to schedule and perform structural changes in the Entity Component System (ECS). Structural changes include creating, destroying, or adding/removing components from entities.

class SignalMovementSystem:

```
    - Moves signal based on direction and speed.

-This code is a part of Unity's ECS system and defines a **job** that
updates the positions of entities with a `LocalTransform` and `Signal`
component based on the signal's direction and speed. It uses **parallel
scheduling** to execute the job across multiple threads, optimizing
performance.
- This system ensures proper placement of objects and manages game state
```

```
transitions based on player interaction with state changers.
- It integrates with the game UI to trigger victory conditions.
```

- Key Components:
  - `Entities.ForEach`:
    - Iterates over all entities in the ECS world that have the specified components (`LocalTransform` and `Signal`).
    - `ref LocalTransform transform`: This allows modifying the entity's position data.
    - `in Signal signal`: The `in` keyword ensures `Signal` is read-only, meaning its data cannot be modified during the job.
- class ObjectSpawnerSystem:
  - Handles the spawning of game objects (absorbers, reflectors, and state changers) in the scene and monitors victory conditions.
  - Key Features:

1. Object Spawning
   - Spawns objects based on specified percentages (absorber, reflector, state changer).
   - Ensures objects are placed on valid ground positions, maintaining required distances:
     - **From the tower** (`minDistanceFromTower`)
     - **Between objects** (`minDistanceBetweenObjects`)
2. **Victory Condition Monitoring**
   - Tracks the number of `state changers` in the "On" state.
   - Triggers a victory event if the "On" state is maintained for a specified duration (`requiredOnTime`).
3. Pause Integration
   - The game pauses when the victory condition is met.
   -Main Components:
4. `OnStartRunning()`
   - Spawns objects using data from the `ObjectSpawnerComponent`.
   - Uses a raycast to ensure valid ground placement and avoids overlap.
5. `OnUpdate()`
   - Checks if the game is paused.
   - Counts active `state changers` (`IsOn == true`).
   - Tracks how long the required count is maintained to trigger victory.
   - Signals victory using `UIManager.onVictory.Invoke(true)`.
6. `GenerateValidPosition()`
   - Validates positions for spawning using:
     - Raycasting to find ground.

- Distance checks to avoid overlapping objects and the tower.