FireFly

Purpose

Represents a catchable firefly object in the game. Supports floating movement and net-based catching logic. Implements `Icatchable`.

Core Features

- Fires a static `OnCaught` event when successfully caught (can be used by other systems like `WaveManager`).
- Movement (currently commented out) is based on Perlin noise and sine waves to simulate natural flight.
- Only specific net types can catch this firefly (`allowedNetTypes` list).

Key Variables

- `allowedNetTypes`: defines which net types can catch this firefly.

Catch Logic

```
public void Catch(FireFlyNet.NetType netType)
```

- If `netType` is allowed:
    - Triggers `OnCaught` event.
    - Plays catch sound.
    - Destroys itself.
- If not: logs warning.

FirefliesSpawner

Purpose

Handles spawning and cleanup of firefly GameObjects in the scene. Supports spawning by specific type or randomly from a list of prefabs.

Core Components

- `fireFly`: list of firefly prefabs (by type).
- `spawnPoints`: locations where fireflies can be instantiated.
- `spawnedFireflies`: tracks all currently active instances for cleanup.

Spawning Methods

SpawnFireFly(int amount, int type)

- Spawns `amount` of fireflies using the prefab at index `type`.
- Randomly picks spawn points.
- If the firefly has a `RandomFlightWithinRadius` script, assigns its center point.

SpawnRandomFireFly(int amount)

- Spawns `amount` of fireflies using random prefabs and random spawn points.

Cleanup

```
public void ClearAllFireflies()
```

- Destroys all currently spawned fireflies and clears the list.

Editor Gizmos

Draws yellow wire spheres at spawn points in the editor using `gizmoRadius`.

Notes

- Logs error in `Awake()` if no spawn points are assigned.
- Spawner can be reused during gameplay for dynamic wave-based behavior.

RandomFlightWithinRadius – Technical Transfer Summary

Purpose

Simulates natural, forward-facing movement of an object (e.g. a firefly) within a defined 3D radius around a center point.

Core Behavior

- Picks a random point inside a radius around `centerPoint`.
- Rotates smoothly toward that target.
- Moves forward only when roughly facing the target.
- Picks a new target once close enough.

Key Fields

- `radius`: movement boundary.
- `moveSpeed`: forward movement speed.

- `rotationSpeed`: how fast the object rotates toward the target.
- `stoppingDistance`: distance threshold before selecting a new target.
- `centerTransform`: optional Transform to define movement center in-editor.

Functions

```
SetCenterPoint(Transform newCenter)
```

- Sets the movement center dynamically (used by `FirefliesSpawner`).

```
PickNewTargetPosition()
```

- Selects a random point within the radius from the center.

```
OnDrawGizmosSelected()
```

- Draws a yellow sphere around the center in the editor for visualization.

Notes

- Only moves when the object is mostly facing the target (`< 10°` angle).
- Used to give fireflies believable, organic floating behavior.

FireFlyWaveManager

Purpose

`FireFlyWaveManager` controls the full wave-based progression system for the Firefly minigame. It handles spawning, difficulty adjustments, session flow, scoring, and backend submission.

It acts as the **main controller** for the Firefly gameplay loop.

Main Responsibilities

- Manage the number of fireflies per wave/session.
- Determine firefly type based on difficulty.
- Track player progress and scoring.
- Handle breaks between sessions.
- Send gameplay data to the backend server.

Structure Overview

Session Flow

Each gameplay session consists of:

- A fixed number of **waves** (`wavesPerSession`)
- Followed by a **break** (`breakDuration`)
- Repeats up to **2 sessions**, then ends

```
StartWaves()
```

- Entry point for launching the full gameplay loop.
- Enables the nets and starts the first session.

```
StartSession()
```

- Starts a new session (up to 2 total).
- Sets wave count to 1 and begins the wave loop.

```
StartWave(int waveNumber)
```

- Calculates how many fireflies to spawn.
- Behavior is **difficulty-dependent**:
    - **Easy**: spawns only one type of firefly.
    - **Medium**: spawns random fireflies.
    - **Hard**: spawns fireflies + butterflies.
- Fireflies are spawned using `firefliesSpawner`, and optionally `butterfliesSpawner`.

```
FireFly_OnCaught()
```

- Called every time a firefly is caught (via `FireFly.OnCaught` event).
- Decreases the remaining count, updates score.
- Starts next wave after delay if all are caught.

Session Control

`BreakSession()`

- Clears butterflies from the previous wave.
- If waves remain in session → spawns next.
- Else → starts break timer.

```
BreakSession()
```

- Coroutine that waits for `breakDuration` before starting next session.

```
EndSession()
```

- Disables nets, resets internal counters.
- Sends performance data to backend using `ExerciseService`.

Scoring

- Score is awarded per firefly caught (`+100 pts`).
- ScoreManager handles point tracking.
- No current handling for wrong catches—hardcoded placeholder (`caughtWrongFirefliesCount = 100`).

Backend Integration

```
SendFireflyData()
```

- Packages session results into `CompletedFireflyExerciseDTO`
- Sends data via coroutine to backend (`excerciseSerice.SaveExercise(...)`)

UI/Gameplay Integration

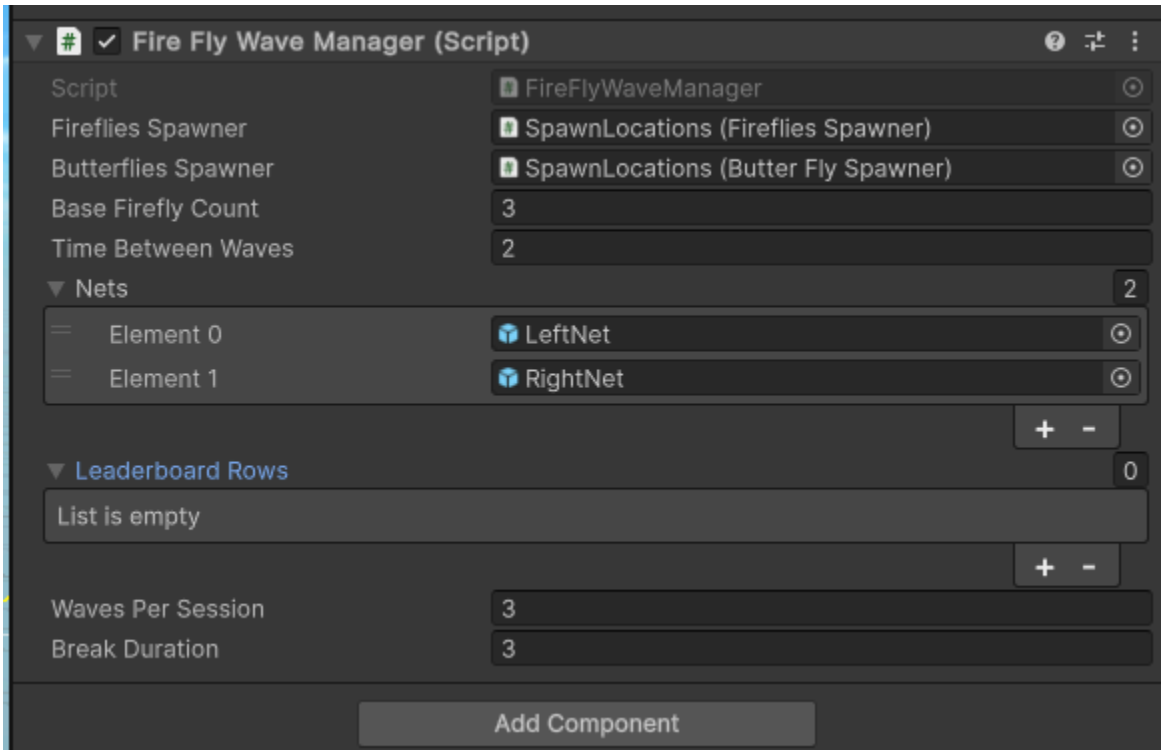
- **Nets** are GameObjects enabled/disabled per session.
- **LeaderboardRows**: presumably used for end-of-session UI (not implemented here).
- Difficulty is pulled from `DifficultyManager.Instance.SelectedDifficulty`.

Dependencies

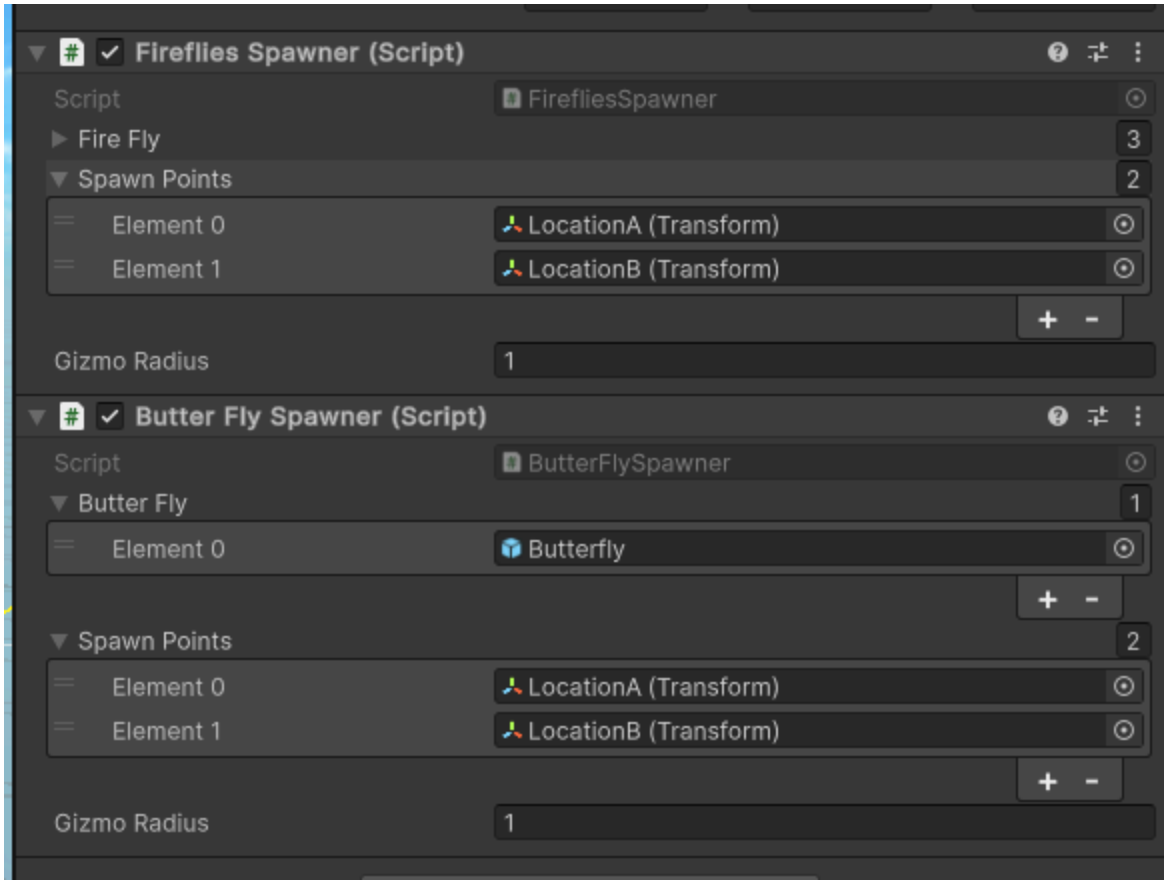
- Requires:
    - `FirefliesSpawner`
    - `ButterFlySpawner`
    - `FireFly` with `OnCaught` event
    - `ScoreManager`, `DifficultyManager`, `ExerciseService`, `StatisticsService`

Notes

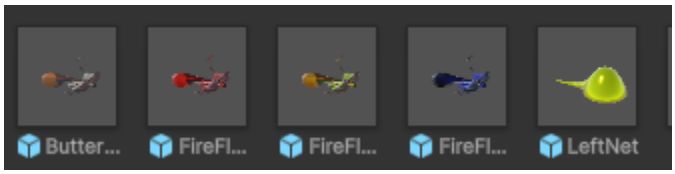
- Singleton: accessible via `FireFlyWaveManager.FireFlyInstance`
- The script is modular and can be extended to support:
    - More complex difficulty scaling
    - Adaptive wave generation
    - Custom scoring or penalties for incorrect actions



Assigned references for the wave manager



Assigned references for the spawners



Firefly game prefab