Refactoring the Example

What is happening?

The code indeed looks very confusing. As I understand it, the main task is to set a certain target. First, there is a check to see if _lockedCandidateTarget and _lockedTarget can be targets. If not, they are set to null. If the target is valid, nothing happens except for unnecessary computations. If the target is outdated, a new target is searched for by checking _lockedTarget and _activeTarget. If there is still no target, _targetInRangeContainer is used. Finally, regardless of whether a return occurs or not, the code in the finally block is executed, where, if a new target is found, the moment of target discovery is recorded through _previousTargetSetTime.

Suggestions for Improvement

The logic is convoluted due to many flags and nested conditions. Using early returns can simplify the process.

The main method `CleanupTest` looks very cumbersome as it performs many different actions. Breaking it into smaller methods will improve readability.

The names of some fields are not very clear. It's not entirely obvious what the difference is between `_target` and `_activeTarget`. Also, there is no need to store `_previousTarget` and `_target` simultaneously, as you can manage with just `_currentTarget`.

There is a `try-finally` block without `catch`, which suggests error checking. However, in this example, it does not occur, and the `finally` block is used for other purposes—namely, final processing after finding a new target by checking `_lockedTarget` and `_activeTarget`.

The code also frequently uses implicit `null` checks. I suggest using explicit `null` checks to make the code more understandable.

My Version

The source code can be found here:
https://github.com/Igor1818/ExamplesForRefactoring/blob/main/Example5/Program_Refactored.cs

Let's separate out the checking and clearing of invalid targets into a separate ClearInvalidLockedTargets method:

```
private void ClearInvalidLockedTargets()
{
    if (_lockedCandidateTarget != null && !_lockedCandidateTarget.CanBeTarget())
    {
        _lockedCandidateTarget = null;
    }

    if (_lockedTarget != null && !_lockedTarget.CanBeTarget())
    {
        _lockedTarget = null;
    }
}
```

Let's separate the check of the current target into separate methods. This will reduce repetitive code and make it clearer through clear names.

```
private bool IsCurrentTargetValid() => IsTargetValid(_currentTarget) && Time.time - _currentTargetSetTime < TargetChangeTime;

private static bool IsTargetValid(object? target) => target != null && target.CanBeTarget();
```

We will also put the definition of new goals into separate methods and give them appropriate names.

```
private object? GetLockedTarget() => IsTargetValid(_lockedTarget) ? _lockedTarget : null;

private object? GetActiveTarget() => IsTargetValid(_activeTarget) ? _activeTarget : null;
```

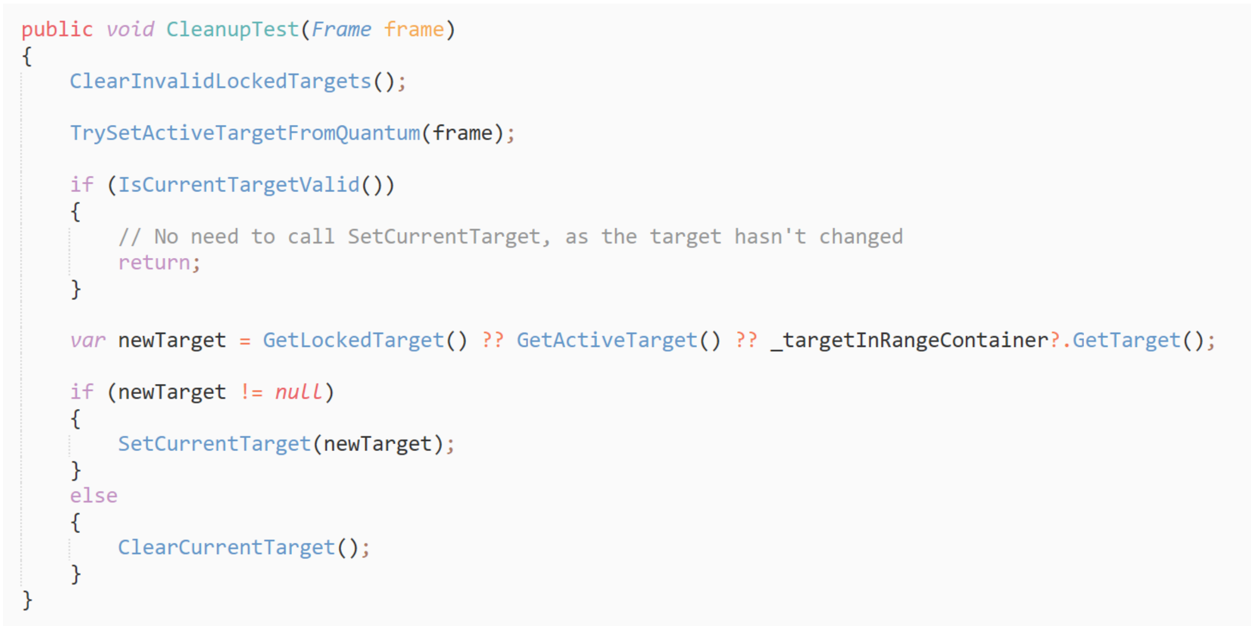
Instead of using the finally block, I suggest using SetCurrentTarget and ClearTarget. I replaced _previousTarget with _currentTarget and _target with the local variable newTarget.

```
private void SetCurrentTarget(object newTarget)
{
    if (_currentTarget == newTarget)
    {
        return;
    }

    _currentTarget = newTarget;
    _currentTargetSetTime = Time.time;
    TargetableEntity.Selected = newTarget;
}

private void ClearTarget()
{
    _currentTarget = null;
    TargetableEntity.Selected = null;
}
```

The main method I ended up with looks much more readable. First, invalid targets are cleared by calling the `ClearInvalidLockedTargets(frame)` method. Then the current target is checked with `IsCurrentTargetValid()`. If it is valid, no further action is required. If not, we proceed to find a new target by querying other possible targets. If a target is found, it is set using the `SetCurrentTarget()` method. Otherwise, the target is cleared.



```
public void CleanupTest(Frame frame)
{
    ClearInvalidLockedTargets();

    TrySetActiveTargetFromQuantum(frame);

    if (IsCurrentTargetValid())
    {
        // No need to call SetCurrentTarget, as the target hasn't changed
        return;
    }

    var newTarget = GetLockedTarget() ?? GetActiveTarget() ?? _targetInRangeContainer?.GetTarget();

    if (newTarget != null)
    {
        SetCurrentTarget(newTarget);
    }
    else
    {
        ClearCurrentTarget();
    }
}
```

This version of the code is an optimized and more readable version of the original, with all the basic functionality retained.