```
///<Summary>
/// Handles rotation of the clock arcs continuously
///<Summary>
private void RotateArcsContinuous()
{
    _rotate1.Rotate(Vector3.forward, 10f * _speedRotate * Time.deltaTime);
    _rotate2.Rotate(Vector3.forward, 6f * _speedRotate * Time.deltaTime);
}

///<Summary>
/// Handles tick-tock style rotation of the clock arcs
///<Summary>
private void RotateArcsTickTock()
{
    if (_rotateArcs)
    {
        _rotate1.Rotate(Vector3.forward, 50f * _speedRotate * Time.deltaTime);
        _rotate2.Rotate(Vector3.forward, 25f * _speedRotate * Time.deltaTime);
    }
    if (_coroutineDelayArcs == null)
    {
        _coroutineDelayArcs = StartCoroutine(DelayArcs());
    }
}

///<Summary>
/// Coroutine that controls the tick-tock delay for the arcs
///<Summary>
private IEnumerator DelayArcs()
{
    yield return new WaitForSeconds(1f);
    _rotateArcs = true;
    yield return new WaitForSeconds(0.15f);
    _rotateArcs = false;
    _coroutineDelayArcs = null;
}

///<Summary>
/// Handles player interaction with the clock
///<Summary>
public void Interact()
{
    if (Cooldown) return;
    Cooldown = true;

    _canSwitchSoundEffect = true;
    _cooldownPhase = true;
    _turnOn = !_parallelWorld.activeSelf;

    SwitchContainers();
    SwitchMannequin();
}

///<Summary>
/// Activates or deactivates the On/Off containers based on state
///<Summary>
private void SwitchContainers()
{
    _onContainer.SetActive(_turnOn);
    _offContainer.SetActive(!_turnOn);
}

///<Summary>
/// Switches the mannequin model if assigned
///<Summary>
private void SwitchMannequin()
{
    if (_mannequinController) _mannequinController.SwitchModel();
}

///<Summary>
/// Handles world switching logic including volume and rotation speed
///<Summary>
private void SwitchWorld()
{
    if (_canSwitchSoundEffect)
    {
        SetSoundEffect(_reversClockSoundEffectClip, false, 50);
        _canSwitchSoundEffect = false;
    }

    if (_turnOn) ActivateWorld();
    else DeactivateWorld();
}

///<Summary>
/// Activates the parallel world and increases visual effect
///<Summary>
private void ActivateWorld()
{
    _parallelWorld.SetActive(true);
    if (_volume.weight < 1f)
    {
        _volume.weight += _speedEffect * Time.deltaTime;
        _speedRotate += _multiplierSpeedRotate * Time.deltaTime;
    }
    else _cooldownPhase = false;
}

///<Summary>
/// Deactivates the parallel world and decreases visual effect
///<Summary>
private void DeactivateWorld()
{
    if (_volume.weight > 0f)
    {
        _volume.weight -= _speedEffect * Time.deltaTime;
        _speedRotate += _multiplierSpeedRotate * Time.deltaTime;
    }
    else
    {
        _parallelWorld.SetActive(false);
        _cooldownPhase = false;
    }
}

///<Summary>
/// Sets and plays a clock sound effect
///<Summary>
private void SetSoundEffect(AudioClip clip, bool loop, float distance = 10)
{
    _clockAudioSource.maxDistance = distance;
    _clockAudioSource.clip = clip;
    _clockAudioSource.loop = loop;
    _clockAudioSource.Play();
}

///<Summary>
/// Handles cooldown logic and rotation speed limits in Update
///<Summary>
private void HandleCooldownAndSpeed()
{
    if (_speedRotate > _maxSpeedRotate)
    {
        _speedRotate -= Time.deltaTime * _multiplierSpeedRotate * 1.25f;
        _canSwitchSoundEffect = true;
    }
    else
    {
        Cooldown = false;
        if (_canSwitchSoundEffect)
        {
            SetSoundEffect(_clockSoundEffectClip, true);
            _canSwitchSoundEffect = false;
        }
    }
}
```