CRITICAL
Master the Atom

Complete Manual Collection

Nuclear Reactor Simulator Project

February 2026

Table of Contents

PART I

Unity Implementation

Unity Implementation Manual v1.0.0.0

February 2026

CRITICAL: Master the Atom

Unity Implementation Manual

Reactor Operator GUI

A step-by-step guide for building the Reactor Operator Screen

in Unity using uGUI, C# scripts, and the CRITICAL physics engine

Version 1.0.0.0 - February 2026

Assumes: Unity 2022.3+ LTS, no prior Unity UI experience

PART A

Unity Fundamentals You Need

Chapter 1: How Unity UI Works

1.1 The Canvas

Everything visible in Unity's UI system lives inside a Canvas. Think of the Canvas as an invisible rectangle that represents your game window. Every button, gauge, image, and text element must be a child (or grandchild) of a Canvas to appear on screen.

Your project already has Canvas creation in MosaicBoardBuilder.cs. The Reactor Operator Screen will use its own Canvas (or the same one).

Canvas Render Modes:

Screen Space - Overlay: The UI draws on top of everything. This is what you want for the reactor control screen.

Screen Space - Camera: UI rendered by a specific camera. Not needed here.

World Space: UI exists in the 3D world. Not needed here.

1.2 RectTransform: Anchors and Offsets

Every UI element has a RectTransform that controls position and size. The two most important concepts are anchors and offsets.

Anchors define where an element sits relative to its parent, using values from 0 to 1:

What You Want

Anchor Values

Meaning

Fill entire parent

min(0,0) max(1,1)

Element stretches to match parent exactly.

Left 15% of parent

min(0,0) max(0.15,1)

Fills left 15%. Stretches vertically with parent.

Bottom 25% of parent

min(0,0) max(1,0.25)

Fills bottom quarter. Stretches horizontally.

Center, fixed size

min(0.5,0.5) max(0.5,0.5)

Stays centered. Does not resize with parent.

IMPORTANT: Y=0 is the BOTTOM, Y=1 is the TOP. This is opposite to many graphics systems where Y goes downward.

Offsets add or subtract pixels after anchors define the region:

offsetMin = (left margin, bottom margin) in pixels. offsetMax = (right margin, top margin) - use negative values to shrink inward. Example: offsetMin=(4,4), offsetMax=(-4,-4) adds a 4px margin on all sides.

1.3 How to Set Anchors in the Inspector

Method 1: Anchor Presets (quick, limited):

Click the small square icon at the top-left of the RectTransform in the Inspector

Hold Alt+Shift and click a preset to set anchors, position, and pivot simultaneously

Good for common layouts (fill, center, corners) but cannot set values like 0.15

Method 2: Type Values Directly (precise):

Expand the RectTransform section to see Anchor Min X/Y and Anchor Max X/Y fields

Type your values: e.g., Min X=0.15, Min Y=0.26, Max X=0.65, Max Y=1.0

TIP: If you see Left/Right/Top/Bottom instead of Anchors, your element is in stretch mode. Those fields ARE the offsets. Click the anchor preset icon to view the actual anchor values.

Chapter 2: GameObjects, Components, and the Inspector

2.1 GameObjects and the Hierarchy

Everything in a Unity scene is a GameObject - an empty container with a name and a position. You add Components to give it behavior. GameObjects form a tree called the Hierarchy. Children inherit their parent's position and visibility.

Your Reactor Operator Screen hierarchy (simplified):

ReactorOperatorCanvas Canvas (Screen Space Overlay)

ReactorOperatorScreen Master controller + dark background

LeftGaugePanel 9 gauges stacked vertically

CoreMapPanel 193-cell interactive core map

RightGaugePanel 8 gauges stacked vertically

DetailPanel Assembly detail (hidden until clicked)

BottomPanel Controls + alarms

2.2 Key Components You Will Use

Component

What It Does

Image

Draws a colored rectangle or sprite. Every visible panel/background uses this.

Text (Legacy)

Renders text on screen. Used for gauge readouts, labels, alarm names.

Button

Makes a GameObject clickable. Fires an OnClick event connected to your code.

Slider

A draggable control. Used for time compression.

CanvasScaler

Controls how UI scales at different resolutions. Set to 1920x1080 reference.

GraphicRaycaster

Enables mouse clicks on UI. Required on Canvas for any interactivity.

HorizontalLayoutGroup

Auto-arranges children left-to-right. Used for button rows and alarm strips.

VerticalLayoutGroup

Auto-arranges children top-to-bottom. Used for gauge columns.

2.3 Wiring References in the Inspector

Components often need references to other objects. For example, MosaicGauge has a "Value Text" field. You connect them by dragging:

Select the gauge GameObject in Hierarchy

In the Inspector, find the MosaicGauge component's "Value Text" slot (empty box)

Drag the child Text GameObject from the Hierarchy into this slot

The gauge now knows which Text to update with numbers

IMPORTANT: This drag-and-drop wiring is exactly what the Builder scripts do in code. When code says gauge.ValueText = valueText, it does the same thing as dragging in the Inspector. The Builder just automates it.

Chapter 3: How Your Existing Code Fits Together

3.1 The Three-Layer Architecture

CRITICAL has a clean separation. Understanding this is essential:

LAYER 1: Physics (Pure C#, no Unity dependency)

PlantConstants, FuelAssembly, ControlRodBank, ReactorCore...

GOLD STANDARD - never modify these.

LAYER 2: Controller (Unity MonoBehaviour)

ReactorController.cs

Owns physics objects, runs the simulation loop.

Exposes properties: .NeutronPower, .Tavg, .Boron_ppm...

Accepts commands: .WithdrawRods(), .Trip(), .SetBoron()...

LAYER 3: GUI (Unity UI) THIS IS WHAT WE BUILD

ReactorOperatorScreen, CoreMosaicMap, MosaicGauge...

READS data from ReactorController properties.

SENDS commands via ReactorController methods.

Never touches physics directly.

The key rule: the GUI never does physics. It reads values and displays them. When the player clicks a button, the GUI calls a ReactorController method which passes the command to physics.

3.2 Data Flow Trace: How Tavg Reaches a Gauge

ReactorCore (physics) calculates Tavg from heat balance equations every timestep

ReactorController.Update() calls _core.Update(dt) each frame

ReactorController exposes: public float Tavg => _core?.Tavg

MosaicBoard.Update() runs at 10 Hz, calls UpdateData() on registered components

MosaicBoard.GetValue(GaugeType.Tavg) reads ReactorController.Tavg

MosaicGauge.UpdateData() calls _board.GetValue(Type) and stores the result

MosaicGauge updates: ValueText.text = "588.5"

Unity renders the Text. Player sees 588.5 on screen.

This entire chain runs automatically once everything is wired. Your job is to create the visual structure (GameObjects, anchors, layout) and wire the references.

3.3 Existing UI Components to Reuse

Script

Role

MosaicBoard.cs

Central hub. Holds ReactorController reference. Provides data to gauges, manages alarm colors, value smoothing. ADD THIS to your screen panel.

MosaicGauge.cs

Displays one parameter. Set Type dropdown, wire ValueText and LabelText references. Auto-registers with MosaicBoard.

MosaicRodDisplay.cs

Draws 8 bank position bars. Auto-reads from ReactorController.

MosaicControlPanel.cs

Buttons for rod control, trip, boron, time. Sends commands to ReactorController.

MosaicAlarmPanel.cs

Alarm tile management. Monitors conditions, controls flashing.

MosaicBoardSetup.cs

Runtime initializer. Auto-creates ReactorController, sets initial state. ADD THIS to your screen panel.

PART B

Building the Reactor Operator Screen

Chapter 4: Project Setup

4.1 Open the Project

Open Unity Hub, open the CRITICAL project

Wait for compilation to finish (progress bar at bottom disappears)

Navigate to Assets/Scenes in the Project window

Right-click > Create > Scene. Name it "ReactorOperatorScreen"

Double-click to open the new scene

4.2 Set Game Window Resolution

Click the Game tab at top of the viewport

Click the resolution dropdown (may say "Free Aspect")

Click + to add custom: Label = "1920x1080", Type = Fixed Resolution, W=1920, H=1080

Select your new resolution

4.3 Set Build Resolution

Edit > Project Settings > Player

Default Screen Width = 1920, Default Screen Height = 1080

Uncheck Default Is Fullscreen for windowed mode

Chapter 5: Creating the Canvas and Master Layout

5.1 Create the Canvas

Right-click in Hierarchy > UI > Canvas

Rename to "ReactorOperatorCanvas"

In Inspector, Canvas component: Render Mode = "Screen Space - Overlay"

Canvas Scaler: UI Scale Mode = "Scale With Screen Size"

Reference Resolution: X=1920, Y=1080

Screen Match Mode: "Match Width Or Height"

Match slider: 0.5

TIP: "Scale With Screen Size" at 1920x1080 means you design at those coordinates and Unity auto-scales to other resolutions.

5.2 Create the Master Screen Panel

Right-click Canvas > UI > Image. Rename to "ReactorOperatorScreen"

Set RectTransform to stretch-fill: hold Alt+Shift, click bottom-right anchor preset

Set Image color: R=26, G=26, B=31 (hex #1A1A1F) - dark background

Add Component: MosaicBoard (search for it)

Add Component: MosaicBoardSetup

5.3 Create the Five Zone Panels

Create five child Images under ReactorOperatorScreen. For each, right-click ReactorOperatorScreen > UI > Image, rename, and set anchors:

Panel Name

AnchorMin

AnchorMax

Purpose

LeftGaugePanel

(0, 0.26)

(0.15, 1)

Left column: 9 nuclear instrument gauges

CoreMapPanel

(0.15, 0.26)

(0.65, 1)

Center: 193-assembly core mosaic map

RightGaugePanel

(0.65, 0.26)

(0.80, 1)

Right column: 8 thermal-hydraulic gauges

DetailPanel

(0.80, 0.26)

(1, 1)

Assembly detail (uncheck Active checkbox to hide initially)

BottomPanel

(0, 0)

(1, 0.26)

Controls, bank bars, trip, time, alarms

Set all panel colors to #1E1E28. Add 4px offset margins: offsetMin=(4,4), offsetMax=(-4,-4).

How to type anchor values:

Select the panel in Hierarchy

In Inspector, click the triangle next to the RectTransform's anchor preset icon to expand raw values

Type: Anchor Min X, Anchor Min Y, Anchor Max X, Anchor Max Y

Type: Left, Bottom (= offsetMin.x, offsetMin.y), Right, Top (use negative for offsetMax)

Chapter 6: Building the Core Mosaic Map

6.1 Concept

The core map displays 193 fuel assemblies as colored square cells in a 15x15 grid with corners removed. Each cell changes color based on reactor conditions and responds to mouse interaction.

The CoreMosaicMap script creates cells at runtime. Here is the process conceptually:

6.2 Setting Up the Map Container

Under CoreMapPanel, create child Image called "MapContainer"

Anchor it to the center area of CoreMapPanel (leave room for buttons above and below):

AnchorMin = (0.05, 0.08), AnchorMax = (0.95, 0.92)

Set Image color to #12121A (darker than panel, makes cells pop)

Add Component: CoreMosaicMap (a new script you will create)

6.3 How CoreMosaicMap Creates 193 Cells

In the CoreMosaicMap.Start() method, the script loops through the 15x15 grid and creates cells:

```
void Start() {
float containerW = GetComponent<RectTransform>().rect.width;
float cellSize = containerW / 15f - gap;
for (int row = 0; row < 15; row++) {
for (int col = 0; col < 15; col++) {
int gridVal = CoreMapData.CORE_GRID[row, col];
if (gridVal == -1) continue;  // skip empty corners
var cellGO = new GameObject($"Cell_{row}_{col}");
cellGO.transform.SetParent(transform, false);
var rect = cellGO.AddComponent<RectTransform>();
float x = col * (cellSize + gap);
float y = -(row * (cellSize + gap));
rect.anchoredPosition = new Vector2(x, y);
rect.sizeDelta = new Vector2(cellSize, cellSize);
var img = cellGO.AddComponent<Image>();
_cellImages[row, col] = img;  // store for color updates
// Make it interactive
var ac = cellGO.AddComponent<AssemblyCell>();
ac.Row = row; ac.Col = col;
ac.ParentMap = this;
}
}
}
```

TIP: The cells are created ONCE at startup and reused. Color updates just change img.color on existing objects - never create/destroy cells each frame.

6.4 Making Cells Interactive

Each cell has an AssemblyCell component implementing Unity's pointer interfaces:

```
using UnityEngine.EventSystems;
public class AssemblyCell : MonoBehaviour,
IPointerEnterHandler, IPointerExitHandler, IPointerClickHandler
{
public int Row, Col;
public CoreMosaicMap ParentMap;
public void OnPointerEnter(PointerEventData d)
```

=> ParentMap.OnCellHover(this);

public void OnPointerExit(PointerEventData d)

=> ParentMap.OnCellHoverExit(this);

public void OnPointerClick(PointerEventData d)

=> ParentMap.OnCellClick(this);

}

These interfaces work automatically when the Canvas has a GraphicRaycaster and the cell Image has Raycast Target enabled (the default). No extra configuration needed.

6.5 Display Mode Buttons (Above Map)

Under CoreMapPanel, create "DisplayModeButtons" with AnchorMin=(0,0.93), AnchorMax=(1,1)

Add HorizontalLayoutGroup: Spacing=4, Child Force Expand Width=true

Create 4 child Buttons: POWER, FUEL TEMP, COOLANT TEMP, BANK MAP

Each button calls CoreMosaicMap.SetDisplayMode(mode) on click

6.6 Bank Filter Buttons (Below Map)

Under CoreMapPanel, create "BankFilterButtons" with AnchorMin=(0,0), AnchorMax=(1,0.07)

Add HorizontalLayoutGroup: Spacing=3, Child Force Expand Width=true

Create 11 buttons: ALL, SD (all shutdown), CTRL (all control), SA, SB, SC, SD, D, C, B, A

Each bank button toggles: CoreMosaicMap.ToggleBankFilter(bankIndex)

ALL/SD/CTRL are quick-select shortcuts

Chapter 7: Building the Gauge Columns

7.1 Using a VerticalLayoutGroup

Rather than positioning each gauge manually, the gauge columns use Unity's VerticalLayoutGroup to stack children automatically:

Select LeftGaugePanel in Hierarchy

Add Component > Vertical Layout Group

Settings: Spacing=3, Padding=4 all sides, Child Force Expand Width=true, Child Force Expand Height=true

Now every child added is automatically stacked vertically with equal spacing

With 9 children and ~785px height, each gauge gets ~85px. With 8 children (right column), each gets ~96px.

7.2 Creating a Single Gauge (Step by Step)

Right-click LeftGaugePanel > UI > Image. Name: "NeutronPowerGauge"

Set Image color: #14141A (dark gauge background)

Add Component: MosaicGauge

In MosaicGauge component, set Type = NeutronPower

Right-click NeutronPowerGauge > UI > Legacy > Text. Name: "Value"

Font Size=20, Alignment=Center, Color=#00FF88, Anchor=fill upper 60%

Drag Value text into MosaicGauge's "Value Text" slot in Inspector

Create another Text child: "Label", Font Size=10, Color=#8090A0, text="NEUTRON POWER"

Drag Label into "Label Text" slot

Repeat for all 9 gauges in the left column

7.3 Left Column: 9 Gauges (Top to Bottom)

#

Name

GaugeType

Range / Notes

1

NeutronPowerGauge

NeutronPower

0-120%

2

ThermalPowerGauge

ThermalPower

0-3800 MWt

3

StartupRateGauge

StartupRate

-1 to +1 DPM

4

PeriodGauge

ReactorPeriod

-999 to +999 sec

5

ReactivityGauge

TotalReactivity

-10000 to +1000 pcm

6

KeffGauge

(custom)

0.900-1.100

7

BoronGauge

Boron

0-2500 ppm

8

XenonGauge

Xenon

0-4000 pcm

9

FlowGauge

FlowFraction

0-120%

7.4 Right Column: 8 Gauges

#

Name

GaugeType

Range / Notes

1

TavgGauge

Tavg

500-650 degF

2

ThotGauge

Thot

500-650 degF

3

TcoldGauge

Tcold

500-650 degF

4

DeltaTGauge

DeltaT

0-80 degF

5

FuelCenterlineGauge

FuelCenterline

500-4800 degF

6

HotChannelGauge

(custom)

Hot channel fuel centerline

7

PressureGauge

(custom)

0-2500 psig (needs PZR model)

8

PZRLevelGauge

(custom)

0-100% (needs PZR model)

Chapter 8: Building the Bottom Control Panel

8.1 Subdivide into Groups

The BottomPanel contains 5 control groups arranged horizontally plus an alarm strip. Use anchor ranges:

Group

AnchorMin X

AnchorMax X

Contains

Rod Control

0

0.20

WITHDRAW, INSERT, STOP, Bank Select, AUTO/MAN

Bank Position

0.20

0.55

MosaicRodDisplay (8 vertical bars)

Boron Control

0.55

0.70

BORATE, DILUTE, ppm readout

Trip Controls

0.70

0.85

TRIP (with cover), RESET, status light

Time Controls

0.85

1.0

Compression slider, PAUSE, sim clock

For each group, use Y anchors from 0.15 to 1.0 (leaving bottom 15% for the alarm strip).

8.2 Creating Buttons

Right-click parent group > UI > Button (Legacy)

Rename (e.g., "WithdrawBtn"), set Image color (#2A3A2A for green-tinted dark)

Select child Text, set label ("WITHDRAW"), Font Size=12, Bold, Color=#00FF88

Connecting to code:

In Button component, find On Click () section at bottom

Click + to add event

Drag the MosaicControlPanel GameObject into the Object slot

Select function: MosaicControlPanel > WithdrawRods

Now clicking this button calls WithdrawRods() on MosaicControlPanel, which calls ReactorController, which tells physics to start withdrawing.

8.3 The Bank Position Display

In the Bank Position group, create a child Image that fills the area

Add Component: MosaicRodDisplay

MosaicRodDisplay reads bank positions from MosaicBoard.Instance automatically

8.4 Trip Button with Safety Cover

Two overlapping buttons - cover on top, actual trip button beneath:

Create "TripCover" button: semi-transparent red, text="LIFT TO ARM"

Create "TripActual" button behind it: bright red, text="TRIP". Start with Active=unchecked

TripCover onClick: hide itself, show TripActual

TripActual onClick: call ReactorController.Trip()

After 5 seconds idle or after trip, cover reactivates

8.5 Time Compression

Add a Slider component to a child object

Set Min Value=0, Max Value=10 (10 discrete notches)

In the OnValueChanged event, map the slider value to time compression:

float[] tcValues = {0, 1, 2, 5, 10, 50, 100, 500, 1000, 5000, 10000};

Add a Text showing current rate ("100x") and a sim clock ("02:15:30")

Chapter 9: Alarm Annunciator Strip

9.1 Layout

Under BottomPanel, create "AlarmStrip" with AnchorMin=(0,0), AnchorMax=(1,0.15)

Add HorizontalLayoutGroup: Spacing=3, Child Force Expand=true

Create 10 child Image+Text pairs (alarm tiles)

9.2 Alarm Tile States

Inactive: dark grey #2A2A2A, grey text - normal condition

Flashing: alternates alarm color / dark every 500ms - new unacknowledged alarm

Steady: constant alarm color - acknowledged but condition still active

MosaicAlarmPanel manages this state machine. New alarm starts flashing, ACK button sets to steady, condition clear extinguishes tile.

Chapter 10: Assembly Detail Panel

10.1 Visibility

The DetailPanel starts with its Active checkbox unchecked (hidden). When the player clicks an assembly cell, CoreMosaicMap shows it:

DetailPanel.gameObject.SetActive(true);

Clicking the same cell again or clicking empty space hides it.

10.2 Content Layout

Add VerticalLayoutGroup to DetailPanel

Child elements, top to bottom:

Header Text: grid position ("H-8, Assembly #97")

Type label: "FUEL" or "RCCA - Bank D"

Divider: thin Image, height=2, color=#444444

Power readout: "Relative Power: 1.05"

Fuel CL temp: "Fuel CL: 2850 degF"

Coolant outlet temp: "Coolant Out: 615 degF"

If RCCA: Rod position, bank worth, motion status

PART C

Wiring It All Together

Chapter 11: Connecting GUI to Physics

11.1 The Connection Hub: MosaicBoard

MosaicBoard is the central hub connecting UI to ReactorController. Add it to the ReactorOperatorScreen panel. All MosaicGauge, MosaicIndicator, and MosaicAlarmPanel components auto-register with MosaicBoard.Instance.

11.2 Wiring Checklist

MosaicBoard.Reactor must reference the ReactorController (auto-wired by MosaicBoardSetup)

Each MosaicGauge: ValueText and LabelText slots connected, Type dropdown set

MosaicRodDisplay: just needs to be a child/descendant of MosaicBoard's object

MosaicControlPanel: same auto-registration via MosaicBoard.Instance

MosaicAlarmPanel: same auto-registration

CoreMosaicMap: reads ReactorController directly for 193-cell updates (new wiring)

11.3 MosaicBoardSetup Does the Heavy Lifting

Add MosaicBoardSetup alongside MosaicBoard. In the Inspector it shows:

Start State dropdown: Cold Shutdown, Hot Zero Power, or Power Operation

Initial Power slider (for Power Operation)

Initial Boron ppm

Auto Start checkbox (starts simulation immediately)

At runtime, it automatically creates ReactorController and ReactorSimEngine if they do not exist in the scene, then initializes the reactor to your chosen state.

Chapter 12: Input Handling

12.1 Screen Toggle (Key 1)

In your ReactorOperatorScreen.Update() method:

```
void Update() {
if (Input.GetKeyDown(KeyCode.Alpha1)) {
screenPanel.SetActive(!screenPanel.activeSelf);
}
}
```

GetKeyDown fires once on keypress. SetActive toggles visibility of the panel and all children.

12.2 Mouse Tooltips

Create a small panel that follows the mouse cursor when hovering over assembly cells:

```
void UpdateTooltip() {
if (hoveredCell != null) {
tooltip.SetActive(true);
tooltip.transform.position = Input.mousePosition + new Vector3(15,-15,0);
tooltipText.text = GetAssemblyInfo(hoveredCell);
} else {
tooltip.SetActive(false);
}
}
```

Chapter 13: The Builder Script

13.1 Pattern

OperatorScreenBuilder follows the same pattern as your existing MosaicBoardBuilder:

```
#if UNITY_EDITOR
[MenuItem("Critical/Create Operator Screen")]
public static void Create() {
// 1. Create Canvas (or find existing)
// 2. Create master panel with MosaicBoard + MosaicBoardSetup
// 3. Create left gauge column (VerticalLayoutGroup + 9 gauges)
// 4. Create core map area + CoreMosaicMap component
// 5. Create right gauge column (8 gauges)
// 6. Create detail panel (hidden)
```

// 7. Create bottom panel (5 control groups + alarm strip)

// 8. Wire all references

}

#endif

13.2 Running It

Save all scripts (Ctrl+S)

Return to Unity, wait for compilation

Menu bar: Critical > Create Operator Screen

Entire hierarchy appears. Press Play to test.

PART D

Testing and Troubleshooting

Chapter 14: Running and Verifying

14.1 Verification Steps

Press Play. Reactor initializes (check Console for setup messages).

Gauges: Neutron Power ~0%, Tavg ~557 degF, Boron ~1500 ppm at HZP.

Core map: 193 cells in cross-pattern (corners empty, NOT a full square).

Hover cells: tooltip shows assembly data.

Click cell: detail panel appears with assembly info.

Bank filter buttons: highlight correct RCCA positions.

WITHDRAW: rod bars move up, reactivity changes.

TRIP (lift cover, press): all bars drop to zero, REACTOR TRIP alarm flashes.

Time compression: simulation speeds up, xenon builds after trip.

Key 1: screen toggles without affecting simulation.

14.2 Console Messages

"Created ReactorController" - normal, auto-created

"Initialized to HotZeroPower" - normal startup

NullReferenceException - a reference not wired. Check script name and line number.

MissingComponentException - component not on the GameObject you expected

Chapter 15: Common Problems and Fixes

Problem

Fix

UI elements invisible

Check: Canvas render mode set? Element Active? Image alpha > 0? Not behind another element? Inside the Canvas?

Wrong position

Y=0 is bottom, Y=1 is top. AnchorMin is bottom-left. Check offsets are not pushing off-screen.

Text not showing

Check: color alpha > 0? Font assigned? Font size not too large? Overflow set to Overflow not Truncate?

Buttons not clickable

Canvas needs GraphicRaycaster? Scene has EventSystem? Image Raycast Target = true? Nothing blocking on top?

Gauge shows 0

MosaicBoard.Reactor assigned? ReactorController initialized? Gauge Type set? ValueText wired?

Layout Group children wrong size

Check Child Force Expand and Control Child Size. Check LayoutElement on children if present.

Core map is full square (not cross)

Corner cells (CORE_GRID == -1) not being skipped. Check rendering loop.

NullReferenceException

Read error for script + line number. That variable is null - a reference was not dragged in Inspector.

Low FPS

Core map updates at 2 Hz max. Gauge updates at 10 Hz. Throttle with Time.time checks.

APPENDICES

Appendix A: Complete GameObject Hierarchy

ReactorOperatorCanvas [Canvas, CanvasScaler, GraphicRaycaster]

EventSystem [EventSystem, StandaloneInputModule]

ReactorController [ReactorController]

ReactorSimEngine [ReactorSimEngine]

ReactorOperatorScreen [Image, MosaicBoard, MosaicBoardSetup]

LeftGaugePanel [Image, VerticalLayoutGroup]

NeutronPowerGauge [Image, MosaicGauge] > Value [Text], Label [Text]

ThermalPowerGauge, StartupRateGauge, PeriodGauge,

ReactivityGauge, KeffGauge, BoronGauge, XenonGauge, FlowGauge

CoreMapPanel [Image]

DisplayModeButtons [HLayoutGroup] > PowerBtn, FuelTempBtn...

MapContainer [Image, CoreMosaicMap] > Cell_R_C [Image, AssemblyCell] x193

BankFilterButtons [HLayoutGroup] > AllBtn, SA..A Btns

RightGaugePanel [Image, VerticalLayoutGroup]

TavgGauge, ThotGauge, TcoldGauge, DeltaTGauge,

FuelCenterlineGauge, HotChannelGauge, PressureGauge, PZRLevelGauge

DetailPanel [Image, AssemblyDetailPanel] (inactive)

HeaderText, TypeLabel, Divider, Readouts...

BottomPanel [Image]

RodControlGroup > WithdrawBtn, InsertBtn, StopBtn, BankSelect

BankPositionDisplay [MosaicRodDisplay]

BoronControlGroup > BorateBtn, DiluteBtn, BoronReadout

TripControlGroup > TripCover, TripActual, ResetBtn, StatusLight

TimeControlGroup > TimeSlider, PauseBtn, SimClock, RateDisplay

AlarmStrip [HLayoutGroup] > 10 alarm tiles

Appendix B: Anchor Quick Reference

Goal

AnchorMin

AnchorMax

Offsets

Fill entire parent

(0, 0)

(1, 1)

(0,0) (0,0)

Left 15%

(0, 0)

(0.15, 1)

(4,4) (-2,-4)

Center 50%

(0.15, 0)

(0.65, 1)

(2,4) (-2,-4)

Right 20%

(0.80, 0)

(1, 1)

(2,4) (-4,-4)

Bottom 25%

(0, 0)

(1, 0.25)

(4,4) (-4,-2)

Top 15% strip

(0, 0.85)

(1, 1)

(0,0) (0,0)

Fixed 200x100 at center

(0.5, 0.5)

(0.5, 0.5)

sizeDelta=(200,100)

Appendix C: Color Palette Reference

Element

Hex

Usage

Screen background

#1A1A1F

Darkest layer. Main screen fill.

Panel background

#1E1E28

Zone panels (gauge columns, bottom panel).

Gauge background

#14141A

Individual gauge dark wells.

Map background

#12121A

Core map container behind cells.

Panel border

#2A2A35

Subtle edge delineation.

Normal text

#C8D0D8

Primary text color.

Label text

#8090A0

Secondary labels, units.

Green accent

#00FF88

Normal values, healthy status.

Amber accent

#FFB830

Warnings, approaching limits.

Red accent

#FF3344

Alarms, trip state.

Cyan accent

#00CCFF

Selection highlight, interactive focus.

Core cold (blue)

#0044AA

Low relative power on core map.

Core hot (red)

#FF2200

High relative power on core map.

Alarm tile inactive

#2A2A2A

Dark grey when no alarm.

- End of Implementation Manual -

Reactor Operator GUI Design v1.0.0.0

February 2026

CRITICAL: Master the Atom

Reactor Operator GUI

Design Specification Document

Version 1.0.0.0

February 2026

Target: 1920 x 1080 Windowed | Unity uGUI

Westinghouse 4-Loop PWR (3411 MWt) Reference Plant

1. Design Overview

1.1 Purpose

The Reactor Operator GUI is the primary gameplay interface for CRITICAL: Master the Atom. It provides the player with a realistic control room experience modeled on authentic Westinghouse 4-Loop PWR operator interfaces. The screen is bound to keyboard key '1' and presents a unified view of reactor status, core conditions, and operator controls.

1.2 Design Philosophy

The GUI design draws from two complementary control room traditions:

Westinghouse PWR Control Board: The horseshoe-shaped instrumentation layout with annunciator panels, vertical rod position indicators, and analog/digital parameter displays. This provides the functional foundation.

RBMK Mosaic Mimic Board: The iconic illuminated core map with colored cells representing individual fuel channels. This provides the visual centerpiece - adapted from the RBMK channel grid to the Westinghouse 193-assembly cross-pattern.

The result is a hybrid aesthetic: the visual drama of the RBMK mosaic board with the engineering accuracy of a Westinghouse PWR control room.

1.3 Resolution and Layout Strategy

The target resolution is 1920 x 1080 pixels in a windowed display. The layout divides the screen into four functional zones:

Zone

Position

Content

Left Gauges

Left 15%

Nuclear instrumentation: power, reactivity, period, startup rate, boron, xenon

Core Mosaic Map

Center 50%

Interactive 193-assembly core visualization with RCCA overlays, color-coded power/temperature

Right Gauges

Right 15%

Thermal-hydraulic parameters: temperatures, pressure, flow, fuel temps

Bottom Panel

Bottom 25%

Control panel (rod controls, bank position bars, boron controls, trip, time compression) and alarm annunciator strip

1.4 Color Theme

Dark industrial theme reflecting actual control room ambient lighting:

Element

Color

Rationale

Background

#1A1A1F

Near-black with blue undertone. Reduces eye strain during extended play.

Panel borders

#2A2A35

Subtle panel delineation without harsh contrast.

Text (normal)

#C8D0D8

Slightly cool white. Readable without glare.

Text (labels)

#8090A0

Muted blue-grey for secondary labels and units.

Accent (green)

#00FF88

Primary healthy/normal indicator. Nuclear green.

Accent (amber)

#FFB830

Warning state. Approaching limits.

Accent (red)

#FF3344

Alarm/trip state. Immediate attention required.

Accent (cyan)

#00CCFF

Selected/highlighted items. Interactive focus.

Core cold (blue)

#0044AA

Low relative power/temperature on core map.

Core hot (red)

#FF2200

High relative power/temperature on core map.

2. Central Core Mosaic Map

The core mosaic map is the visual centerpiece of the GUI and the primary innovation beyond standard PWR control room displays. It renders all 193 fuel assemblies as interactive colored cells in the authentic Westinghouse cross-pattern, with RCCA positions overlaid as bank-colored markers.

2.1 Core Map: 193 Fuel Assembly Layout

The Westinghouse 4-Loop PWR contains 193 fuel assemblies arranged in a roughly circular cross-pattern within a 15x15 grid. The corners of the grid are unoccupied, yielding the characteristic octagonal core cross-section. Each assembly is a 17x17 lattice of 264 fuel rods, 24 guide thimbles, and 1 instrument tube.

Of the 193 assemblies, 53 contain Rod Cluster Control Assemblies (RCCAs) distributed across 8 banks. The remaining 140 assemblies contain either burnable poison rod assemblies (BPRAs), thimble plugs, or secondary neutron source assemblies.

2.1.1 Core Cross-Pattern (15x15 Grid Occupancy)

The following table shows the 15x15 grid occupancy. Each cell shows the assembly type: F = fuel-only, bank letter = RCCA location (SA, SB, SC, SD, D, C, B, A), dash = unoccupied corner position. Grid coordinates use Row (1-15, top to bottom) and Column (A-P, excluding I, left to right).

This layout is based on the standard Westinghouse 4-Loop core loading arrangement from NRC FSAR documentation. The RCCA bank assignments follow the typical pattern where shutdown banks (SA-SD) occupy peripheral and semi-peripheral positions, while control banks (D, C, B, A) occupy positions nearer the core center for effective power shaping.

The core occupancy follows quarter-core symmetry (octant symmetry for RCCA placement). The 15x15 grid has corner positions removed:

Row 1: -- -- -- -- F F F F F F F -- -- -- --

Row 2: -- -- F F F F F F F F F F F -- --

Row 3: -- F F F F F F F F F F F F F --

Row 4: -- F F F F F F F F F F F F F --

Row 5: F F F F F F F F F F F F F F F

Row 6: F F F F F F F F F F F F F F F

Row 7: F F F F F F F F F F F F F F F

Row 8: F F F F F F F F F F F F F F F

Row 9: F F F F F F F F F F F F F F F

Row 10: F F F F F F F F F F F F F F F

Row 11: F F F F F F F F F F F F F F F

Row 12: -- F F F F F F F F F F F F F --

Row 13: -- F F F F F F F F F F F F F --

Row 14: -- -- F F F F F F F F F F F -- --

Row 15: -- -- -- -- F F F F F F F -- -- -- --

Total occupied positions: 193. Corner exclusions: 4 groups of 8 = 32 removed from 225 (15x15).

2.1.2 RCCA Bank Assignments (53 locations)

The 53 RCCA locations are distributed across 8 banks following standard Westinghouse practice. Bank assignments follow octant symmetry where possible:

Bank	Type	RCCAs	Function
SA	Shutdown	8	First withdrawn during startup. Provides shutdown margin.
SB	Shutdown	8	Second withdrawn. Peripheral core positions.
SC	Shutdown	4	Third withdrawn. Semi-peripheral positions.
SD	Shutdown	4	Last shutdown bank withdrawn. Provides scram reactivity.
D	Control	9	Lead control bank. Primary regulating bank for load follow and temperature control.
C	Control	9	Overlap bank. Supplements Bank D reactivity range.
B	Control	8	Deep power reduction bank. Moves only at lower power levels.
A	Control	4	Lead bank for power reduction. First control bank inserted on power decrease.

Total: 53 RCCAs (24 shutdown + 29 control). Some references cite 53 total for 4-Loop, others cite 57. Our simulation uses 53 per the standard Westinghouse Technology Manual.

Note: The remaining 140 fuel assembly positions contain no movable control elements and are occupied by fuel-only assemblies, BPRAs, thimble plugs, or neutron source assemblies.

2.2 Visual Rendering

2.2.1 Assembly Cell Rendering

Each fuel assembly is rendered as a square cell approximately 28x28 pixels (fitting the 15x15 grid into ~450x450 pixel area, with 2px gaps). The cell color encodes the primary display parameter selected by the operator:

Display Mode

Color Mapping

Data Source

Relative Power

Blue (#0044AA) to Red (#FF2200) continuous gradient

Assembly power fraction from ReactorCore (power distribution model)

Fuel Temperature

Blue to Yellow to Red (3-stop gradient)

FuelAssembly.CenterlineTemp_F mapped to range

Coolant Temperature

Cyan to Orange gradient

Assembly outlet temperature mapped to range

Rod Bank Overlay

Unique color per bank (see below), unoccupied = grey

ControlRodBank assignment map + position data

Default display mode is Relative Power, which provides the most operationally relevant information at a glance.

2.2.2 RCCA Overlay Rendering

The 53 RCCA locations are overlaid on the core map with bank-specific visual indicators. Each RCCA cell shows:

Bank letter label (SA, SB, SC, SD, D, C, B, A) in the cell corner, using bank color

Insertion indicator - a vertical fill bar within the cell showing rod position (0 = full bar from top = fully inserted, 228 = empty = fully withdrawn)

Motion indicator - subtle pulse animation when bank is actively moving (withdrawing or inserting)

Bank color assignments:

Bank

Hex Color

Description

Visual Notes

SA

#009688

Teal

Shutdown banks use cool tones

SB

#26A69A

Light Teal

Distinguishable from SA

SC

#66BB6A

Green

Fewer assemblies (4), needs visibility

SD

#43A047

Dark Green

Last shutdown bank, transition tone

D

#42A5F5

Blue

Lead control bank. Most visible - largest bank (9).

C

#7E57C2

Purple

Overlap bank, 9 assemblies

B

#EC407A

Pink

Deep reduction bank, 8 assemblies

A

#FFA726

Orange

Lead reduction bank, 4 assemblies. Warm tone for regulating bank.

2.2.3 Interactive Behavior

The core map supports the following interaction modes, designed to maximize information density within the limited 1920x1080 space:

Hover (Mouse Over Assembly):

Tooltip displays: Assembly grid position (e.g., H-8), relative power, fuel centerline temperature, coolant outlet temperature, and RCCA bank/position if applicable

Highlighted assembly cell gets a bright border outline (cyan)

Click (Select Assembly):

Locks the detail panel (see 2.3) to show expanded data for this assembly

If the assembly has an RCCA, highlights ALL assemblies in the same bank with a bank-colored border glow

Click again or click empty space to deselect

Bank Filter Buttons (Below Core Map):

Row of 8 bank toggle buttons: SA | SB | SC | SD | D | C | B | A

Clicking a bank button highlights all RCCA locations for that bank on the core map with bright bank-colored outlines

Non-selected bank assemblies dim slightly (60% opacity) to provide visual contrast

Multiple banks can be selected simultaneously (toggle behavior)

'ALL' button selects/deselects all banks. 'CTRL' button selects only control banks. 'SD' button selects only shutdown banks.

Display Mode Selector (Above Core Map):

Toggle buttons for display mode: POWER | FUEL TEMP | COOLANT TEMP | BANK MAP

Changes the color-coding basis for all 193 cells

BANK MAP mode shows assemblies colored by bank assignment (using bank colors above), fuel-only assemblies in neutral grey

2.3 Assembly Detail Panel

When an assembly is selected (clicked), a detail panel appears adjacent to the core map showing comprehensive data for that assembly. This panel replaces the need for a separate drill-down screen.

Panel contents:

Grid position and assembly index (e.g., H-8, Assembly #97)

Assembly type: Fuel, RCCA (with bank ID), BPRA, Source

Relative power factor

Fuel centerline temperature (degF)

Coolant outlet temperature (degF)

If RCCA: Rod position (steps), bank total worth (pcm), bank insertion reactivity (pcm), motion status

Mini radial temperature profile graphic: fuel centerline -> pellet surface -> gap -> clad -> coolant

3. Instrument Gauge Columns

3.1 Left Column - Nuclear Instrumentation

The left gauge column presents all nuclear/reactivity parameters. These are the parameters a reactor operator monitors to understand the neutronic state of the core. Arranged top-to-bottom by operational priority:

Parameter

Unit

Range

Source Property

Neutron Power

%

0 - 120%

ReactorController.NeutronPower * 100

Thermal Power

MWt

0 - 3800

ReactorController.ThermalPower_MWt

Startup Rate

DPM

-1 to +1

ReactorController.StartupRate_DPM

Reactor Period

sec

-999 to +999

ReactorController.ReactorPeriod (clamped)

Total Reactivity

pcm

-10000 to +1000

ReactorController.TotalReactivity

keff

0.900 - 1.100

ReactorController.Keff

Boron Concentration

ppm

0 - 2500

ReactorController.Boron_ppm

Xenon Worth

pcm

0 - 4000

ReactorController.Xenon_pcm (abs value)

RCS Flow

%

0 - 120%

ReactorController.FlowFraction * 100

Each gauge displays as a combined analog arc (upper 70% of gauge area) with a digital readout (lower 30%). The analog arc provides quick visual assessment while the digital readout provides precision. Gauge borders change color

based on alarm state: green (normal), amber (warning), red (alarm).

3.2 Right Column - Thermal-Hydraulic Instrumentation

The right gauge column presents thermal-hydraulic parameters reflecting the heat removal and coolant state:

Parameter

Unit

Range

Source Property

Tavg

degF

500 - 650

ReactorController.Tavg

Thot

degF

500 - 650

ReactorController.Thot

Tcold

degF

500 - 650

ReactorController.Tcold

T (Core)

degF

0 - 80

ReactorController.DeltaT

Fuel Centerline

degF

500 - 4800

ReactorController.FuelCenterline

Hot Ch. Centerline

degF

500 - 4800

ReactorController.HotChannelCenterline

RCS Pressure

psig

0 - 2500

PZR pressure minus 14.7 (when PZR model available)

PZR Level

%

0 - 100

PZR level percentage (when PZR model available)

4. Bottom Control Panel

The bottom 25% of the screen houses all operator controls and the alarm annunciator strip. It is subdivided into five functional groups arranged left to right:

4.1 Rod Control Group

WITHDRAW button - Starts sequential rod withdrawal (SA->A with overlap)

INSERT button - Starts sequential rod insertion (A->SA)

STOP button - Halts all rod motion immediately

BANK SELECT dropdown - Selects individual bank for manual single-bank control

AUTO/MANUAL toggle - Switches between automatic rod control (Tavg program) and manual mode

4.2 Bank Position Display

Eight vertical bar indicators showing positions of all rod banks (SA through A). Each bar:

Scale: 0 (bottom, fully inserted) to 228 (top, fully withdrawn)

Colored fill using bank color scheme from Section 2.2.2

Digital step count displayed below each bar

Bank label (SA, SB, SC, SD, D, C, B, A) above each bar

Motion indicator arrow (up/down) when bank is moving

Insertion limit line (dashed) at step 30 for Bank D (BANK_D_INSERTION_LIMIT)

4.3 Boron/Chemistry Control

BORATE button - Increases boron concentration (positive reactivity control)

DILUTE button - Decreases boron concentration (negative reactivity control)

Boron ppm readout - Current boron concentration with trend arrow

Rate selector - Adjusts boration/dilution rate

4.4 Trip and Safety Controls

TRIP button - Red, with safety cover toggle. Drops all rods. Requires cover lift before press (two-action safety).

RESET TRIP button - Resets trip condition after all rods confirmed at bottom. Grayed out unless trip conditions cleared.

TRIP STATUS indicator - Backlit panel: NORMAL (green) or TRIPPED (flashing red)

4.5 Time Compression Controls

Time compression slider - Logarithmic scale: 1x, 2x, 5x, 10x, 50x, 100x, 500x, 1000x, 5000x, 10000x

PAUSE button - Freezes simulation time (TimeCompression = 0)

Simulation clock display - Shows elapsed simulation time in HH:MM:SS format

Current compression readout - Shows active time compression factor (e.g., "100x")

4.6 Alarm Annunciator Strip

A horizontal strip of backlit alarm tiles spanning the full width of the bottom panel. Modeled after the Westinghouse

annunciator window design:

Alarm Tile

Color (Active)

Trigger Condition

REACTOR TRIP

Magenta flash

ReactorController.IsTripped == true

OVERPOWER

Red flash

ThermalPower > 1.04 (104%)

HI NEUTRON FLUX

Red flash

NeutronPower > 1.09 (109% trip setpoint)

LO RCS FLOW

Red flash

FlowFraction < 0.87 (87% low flow trip)

TAVG HI

Amber flash

Tavg > 590 degF (above HFP setpoint + deadband)

TAVG LO

Amber flash

Tavg < 555 degF (below HZP setpoint)

ROD BOTTOM

Amber flash

Any control bank at step 0 (ControlRodBank.RodBottomAlarm)

ROD DEVIATION

Amber

Bank sequence violation detected

HI STARTUP RATE

Amber flash

StartupRate_DPM > 0.5 DPM

SHORT PERIOD

Red flash

ReactorPeriod > 0 AND < 30 seconds

Alarm behavior follows the standard Westinghouse annunciator sequence: new alarm flashes rapidly until acknowledged (ACK button), then burns steady until condition clears, then extinguishes. The ACK button clears the flash state but not

the alarm itself.

5. Data Architecture and Module Integration

5.1 Data Flow

The GUI reads data exclusively from ReactorController (MonoBehaviour) which in turn reads from the physics modules. No physics calculations occur in the GUI layer. The data flow is strictly one-directional for display, with operator commands flowing back through ReactorController methods:

ReactorCore (physics) -> ReactorController (coordinator) -> GUI (display)

GUI (operator input) -> ReactorController (commands) -> ReactorCore (physics)

5.2 New Components Required

Component

Type

Purpose

ReactorOperatorScreen.cs

MonoBehaviour

Master screen controller. Manages toggle (key 1), layout, component wiring, update loop.

CoreMosaicMap.cs

MonoBehaviour

193-assembly interactive core map. Handles rendering, selection, hover, bank filtering.

CoreMapData.cs

Static Data

Assembly grid positions, RCCA bank assignments, coordinate lookup tables. Pure data, no MonoBehaviour.

AssemblyDetailPanel.cs

MonoBehaviour

Expandable detail panel for selected assembly. Shows radial temp profile, rod data, etc.

OperatorScreenBuilder.cs

Editor Tool

Menu tool: Critical > Create Operator Screen. Generates complete UI hierarchy.

Existing components to be reused as-is (these are GOLD STANDARD validated modules and must not be modified):

MosaicGauge.cs - Individual gauge rendering (analog arc + digital readout)

MosaicIndicator.cs - Binary status indicators with flash capability

MosaicRodDisplay.cs - Rod position visualization (vertical bars)

MosaicControlPanel.cs - Operator controls (rod, boron, trip, time)

MosaicAlarmPanel.cs - Alarm annunciator with scrolling list

MosaicTypes.cs - Shared enums (GaugeType, AlarmState, IndicatorCondition)

5.3 CoreMapData Static Layout

The CoreMapData class encodes the authentic Westinghouse 4-Loop core layout as static arrays. This data is derived from NRC FSAR documentation and the Westinghouse Technology Systems Manual.

Key data structures:

// 15x15 grid: -1 = empty (corner), 0 = fuel-only, 1-8 = RCCA bank index

public static readonly int[,] CORE_GRID = new int[15, 15] { ... };

// Assembly index (0-192) to grid position mapping

public static readonly (int row, int col)[] ASSEMBLY_POSITIONS = new (int,int)[193];

// RCCA bank to assembly indices mapping

public static readonly int[][] BANK_ASSEMBLIES = new int[8][];

This data must be validated against the Westinghouse core loading arrangement (NRC FSAR Figure 3.1-5 or equivalent). The exact RCCA positions within the 193-assembly grid must match the standard 4-Loop reference plant.

6. Implementation Plan

6.1 Implementation Phases

Step

Component

Deliverables

1

CoreMapData.cs

Static data class with authenticated 193-assembly grid layout, RCCA bank assignments, coordinate mappings. Full validation method.

2

CoreMosaicMap.cs

Core map MonoBehaviour: renders 193 cells, applies color-coding, handles hover/click selection, bank filtering. Uses Unity Image components in a Grid Layout.

3

AssemblyDetailPanel.cs

Floating detail panel that appears when an assembly is selected. Reads assembly-specific data from ReactorController/ReactorCore.

4

ReactorOperatorScreen.cs

Master screen controller: key 1 toggle, layout management, component orchestration, gauge data binding, alarm wiring.

5

OperatorScreenBuilder.cs

Editor tool to generate complete UI hierarchy. Menu: Critical > Create Operator Screen. Wires all references.

6

Integration + Test

Connect to ReactorController, verify data flow at HZP/50%/100% power states, validate alarm triggers, test interaction (hover, click, bank filter).

6.2 Technical Constraints

GOLD standard modules (ReactorCore, ControlRodBank, FuelAssembly, PlantConstants, all physics) must NOT be modified

GUI update rate: 10 Hz (100ms) for gauges, 2 Hz (500ms) for core map color updates (performance)

All rendering via Unity uGUI (Canvas, Image, Text, Button) - no custom shaders required for Phase 1

Core map uses 193 pre-instantiated Image GameObjects (not dynamically created each frame)

Tooltip and detail panel use object pooling (show/hide, not create/destroy)

All RCCA bank assignments and core geometry must be validated against Westinghouse 4-Loop PWR technical specifications

6.3 Acceptance Criteria

All 193 assembly cells render in correct Westinghouse cross-pattern with no gaps or misalignment

53 RCCA locations correctly identified and bank-assigned per reference plant data

Core map color-coding updates in real-time reflecting ReactorCore physics state

All 17 gauges (9 left + 8 right) display correct data from ReactorController with proper units and ranges

Rod control commands (withdraw, insert, stop, trip, reset) function correctly through GUI buttons

Bank position bars accurately track all 8 banks with correct bank colors and step counts

Alarm annunciator tiles trigger and flash correctly for all defined alarm conditions

Interactive core map: hover shows tooltip, click selects assembly, bank filter highlights correct positions

Screen toggle (key 1) shows/hides entire GUI without affecting simulation state

Time compression controls function correctly across full range (1x to 10000x)

7. Pixel Layout Specification (1920 x 1080)

Exact pixel regions for the four layout zones, accounting for 4px panel margins:

Zone

X

Y

Width

Height

Notes

Left Gauges

4

4

280

788

9 gauges @ ~85px each

Core Map Area

288

4

960

788

Map + mode buttons + bank buttons

Right Gauges

1252

4

280

788

8 gauges @ ~96px each

Detail Panel

1536

4

380

788

Assembly detail (when selected)

Bottom Panel

4

796

1912

280

Controls + alarms

The core map rendering area within the Core Map Zone is approximately 480 x 480 pixels (32px per cell x 15 cells), leaving room for mode selector buttons above and bank filter buttons below.

The detail panel on the far right is only visible when an assembly is selected. When no assembly is selected, the right gauge column extends to fill the available space, or the core map area widens slightly.

8. Technical References

NRC ML11223A212 - Westinghouse Technology Systems Manual, Section 3.1: Reactor Vessel and Internals (core loading arrangement, RCCA design parameters)

NRC ML11223A342 - Westinghouse Technology Systems Manual, Section 19.0: Plant Operations (startup/shutdown procedures, control room operations)

Westinghouse 4-Loop FSAR Chapter 4 - Reactor Design (fuel assembly specifications, RCCA bank assignments, core geometry)

NRC NUREG/CR-6042 - Control Room Design Review Guidelines (human factors engineering for nuclear plant control rooms)

MIT OCW 22.06 - Engineering of Nuclear Systems, PWR Description (Buongiorno): fuel assembly geometry, RCCA configuration, core parameters

Existing CRITICAL Codebase - PlantConstants.cs, ControlRodBank.cs, FuelAssembly.cs, ReactorController.cs, MosaicBoard.cs and related UI components

- End of Design Document -

PART II

Blender to Unity Construction Guides

Section 0: Title and Overview

February 2026

WESTINGHOUSE 4-LOOP PWR

CONTROL ROOM INDICATOR TYPES

Blender 5.0 -> Unity 6.3 Construction Manual

Companion Volume to the Analog Round Gauge Manual

Vertical Edgewise Meters Vertical Bar Graphs Strip Chart Recorders

Annunciator Windows Indicator Lamps Digital Numeric Readouts

Nuclear Reactor Simulator Project - February 2026

Table of Contents

Update this Table of Contents in Word/LibreOffice by right-clicking and selecting Update Field.

Part 1: Real-World PWR Control Room Indicator Types

A Westinghouse 4-Loop PWR main control room employs a diverse array of instrumentation displays across its vertical panels and benchboards. The control room is arranged in a horseshoe or wrap-around configuration, divided into functional sections: the reactor/RCS panel, steam generator and feedwater panel, turbine-generator panel, electrical distribution panel, and engineered safety features panel. Each section uses a specific mix of indicator types optimized for the information being conveyed.

The companion Analog Round Gauge Manual covered the classic Bourdon-tube style circular dial gauge in complete detail. This manual covers the six remaining major indicator types found on Westinghouse PWR control panels, each serving a distinct operational purpose within the control room human-system interface.

1.1 Complete Indicator Type Inventory

The following table catalogues every major indicator type found on a real Westinghouse 4-Loop PWR main control room panel, with typical parameters and panel locations.

Indicator Type

Typical Use

Panel Location

Analog Round Gauge

Temperatures, pressures, levels

Vertical panel faces, eye level

Vertical Edgewise Meter

Loop temps, SG levels, motor currents, flows

Dense vertical arrays on panel faces

Vertical Bar Graph Display

PZR level, SG levels, flow rates

Modernized panel sections

Strip Chart Recorder

Temperature/pressure/level/power trends

Benchboards and vertical faces

Annunciator Window Tile

All alarm conditions

Grid arrays at TOP of all panels

Indicator Lamp (Status Light)

Pump/valve/breaker component status

Embedded in mimic bus diagrams

Digital Numeric Readout

Rod position, power %, precise readings

Where high precision is needed

1.2 Design Philosophy

Each indicator type is modeled as a 2.5D asset in Blender 5.0, exported as FBX, imported into Unity 6.3, and rendered to a RenderTexture displayed on the GUI panel via a RawImage component. This provides photorealistic fidelity while maintaining excellent runtime performance. All dimensions, color schemes, scale markings, and behavioral characteristics are based on real Westinghouse 4-Loop PWR instrumentation as documented in NRC NUREG-0700, Weschler Instruments nuclear-grade specifications, and standard nuclear industry human factors engineering practice.

Section 1: Vertical Edgewise Meter

February 2026

Part 2: Vertical Edgewise Meter

2.1 Real-World Reference

The vertical edgewise meter is the single most common indicator on a Westinghouse PWR control panel. The industry standard is the Weschler VX-252 and VC-252 series, designed and seismically qualified specifically for the nuclear power industry per IEEE Standard 344-1987 and IEEE Standard 420-1973. These meters occupy a narrow vertical profile approximately 2.5 inches wide by 6 inches tall, allowing operators to compare many parameters at a glance in dense side-by-side arrays.

A thin knife-edge pointer moves vertically along a printed linear scale on the left or right side of the face. The faceplate is white or off-white with black scale markings and a black pointer. On a Westinghouse 4-Loop PWR, edgewise meters monitor: RCS loop temperatures (T-hot and T-cold for all 4 loops), SG narrow-range levels, RCP motor currents, charging and letdown flow, main steam pressures, and containment sump levels.

2.2 Specifications

Property

Specification

Overall Dimensions

2.5" W x 6.0" H x 2.5" D (front face visible)

Faceplate

White/off-white background, matte finish

Scale

Vertical, linear, printed on left or right side

Major Graduations

Every 50 or 100 units, with numeric labels

Minor Graduations

Every 10 or 25 units

Pointer

Black knife-edge, translates vertically

Pointer Travel

Approximately 4.5" of vertical travel

Bezel

Black plastic rectangular frame

Accuracy

2% per ANSI C39.1, calibratable to 1%

Seismic Qualification

IEEE 344-1987

2.3 Parameter Ranges

Parameter

Edgewise Meter Range

T-hot / T-cold (per loop)

100-650 degF

SG Narrow Range Level

0-100%

RCP Motor Current

0-500 A

Charging Flow

0-150 gpm

Letdown Flow

0-120 gpm

Main Steam Pressure

0-1200 psig

2.4 Blender 5.0 Modeling

Scene Setup

Open Blender 5.0, delete default cube. Set units: Metric, Unit Scale 0.01 (1 BU = 1 cm).

Set viewport background to neutral dark gray (#1A1A1A).

Meter Body

Add Cube. Dimensions: X=6.35cm, Y=0.5cm, Z=15.24cm. Name "EdgewiseBody".

Edit Mode: select front (+Y) face, Inset (I) by 0.2cm for bezel border, extrude inner face back -0.1cm for recess.

Apply black plastic material: Base Color #1A1A1A, Roughness 0.7, Metallic 0.0.

Faceplate

Add Plane, rotate R X 90. Dimensions: X=5.95cm, Z=14.64cm. Position Y=0.21cm. Name "EdgewiseFace".

UV Unwrap (U -> Unwrap). Apply material with Image Texture for the scale.

TIP: Create scale texture in GIMP/Photoshop: 256x512px, white background, vertical scale line on left with major ticks every ~50px, minor ticks every ~10px. Black numerals in small sans-serif font. Export as PNG.

Pointer

Add Plane. Reshape in Edit Mode to a thin horizontal knife: ~3.0cm wide (X) x 0.15cm tall (Z). Merge two left vertices to form a pointed tip.

Position at Y=0.25cm (in front of faceplate). Set origin to bottom-center of the pointer (this is the slide axis anchor).

Name "EdgewisePointer". Apply solid black material (#0A0A0A, Roughness 0.3).

WARNING: Unlike the round gauge needle which rotates, the edgewise pointer translates vertically. The C# script moves this object along local Z. Ensure the origin is correct.

Camera and Lighting

Add Camera: Orthographic, position (0, 15, 7.5), rotation (90, 0, 0), Ortho Scale 18.

Add Area Light above-front: position (0, 10, 20), 50W, 5000K. Optional fill light from below at 10W.

2.5 FBX Export

Select all objects. File -> Export -> FBX: Scale=0.01, Apply Scalings=FBX All, Forward=-Z, Up=Y, Apply Modifiers=checked. Save as EdgewiseMeter.fbx.

2.6 Unity 6.3 Setup

Import FBX to Assets/Models/Gauges/Edgewise/. Scale Factor=1.0. Import textures to Assets/Textures/Gauges/.

Create layer "GaugeEdgewise". Assign meter prefab to this layer.

Create camera "EdgewiseCam": Orthographic, Culling Mask=GaugeEdgewise only.

Create RenderTexture 256x512px. Assign to EdgewiseCam Target Texture.

Add RawImage on GUI Canvas, assign the RenderTexture.

2.7 C# Driver Script

Attach to EdgewisePointer:

```
using UnityEngine;
public class EdgewiseMeterDriver : MonoBehaviour
{
[Header("Scale Configuration")]
public float scaleMin = 100f;
public float scaleMax = 650f;
[Header("Pointer Travel (local Z positions)")]
public float pointerZMin = 0.015f;  // Bottom of scale
public float pointerZMax = 0.130f;  // Top of scale
[Header("Dynamics")]
public float damping = 8.0f;
private float _targetZ;
public void SetValue(float value)
{
float v = Mathf.Clamp(value, scaleMin, scaleMax);
float t = Mathf.InverseLerp(scaleMin, scaleMax, v);
_targetZ = Mathf.Lerp(pointerZMin, pointerZMax, t);
}
void Update()
{
Vector3 pos = transform.localPosition;
pos.z = Mathf.Lerp(pos.z, _targetZ, Time.deltaTime * damping);
transform.localPosition = pos;
}
}
```

Section 2: Vertical Bar Graph

February 2026

Part 3: Vertical Bar Graph Display

3.1 Real-World Reference

Vertical bar graph displays (Weschler BG-252 series) combine a 101-segment vertical LED bar with a digital numeric readout. Green segments indicate normal range, amber for caution, red for alarm. Programmable setpoint markers (triangular LEDs) show Hi/Lo thresholds. Trend arrows indicate rising/falling values.

On Westinghouse 4-Loop PWR panels, bar graphs monitor: pressurizer level (0-100%, setpoints at 17% Lo and 92% Hi), SG narrow-range levels, RCS flow rates, CVCS charging and letdown flows, and boric acid concentrations.

3.2 Specifications

Property

Specification

Overall Dimensions

2.5" W x 6.0" H (same panel cutout as edgewise)

Bar Column

101 LED segments, green/amber/red zones

Digital Display

3.5 or 4.5 digit LED below bar column

Setpoint Markers

Triangular LED indicators at Hi/Lo thresholds

Trend Arrows

LEDs above digital display

Housing

Black plastic, flush panel mount

Update Rate

4 Hz (250 ms refresh)

3.3 Blender 5.0 Modeling

Housing and Bar Column

Add Cube: X=6.35cm, Y=0.5cm, Z=15.24cm. Edit front face: inset 0.25cm, extrude back -0.15cm. Name "BarGraphBody". Black plastic material.

Add Plane for bar column background: 1.2cm x 10.5cm, positioned inside recess. Name "BarColumnBG". Very dark material (#0D0D0D).

Duplicate as "BarFill": position slightly forward (Y+0.01). Set origin to bottom edge (pivot for vertical scaling). Apply emissive green material: Base Color #00CC44, Emission #00CC44, Strength 3.0.

TIP: For segmented LED appearance: create a 32x512px PNG with thin dark horizontal lines every 5px. Apply as alpha-cutout texture over the emissive material.

Setpoints and Digital Area

Create two small triangles (3-vertex circles, ~0.3cm) on bar edge at Hi/Lo positions. Emissive amber material (#FFAA00, Strength 2.0). Name "SetpointHi"/"SetpointLo".

Add small Plane below bar column for digital readout: 4.0cm x 2.0cm. Dark background (#0A0A0A). Name

"DigitalReadout". This displays the numeric value via TextMeshPro in Unity.

3.4 Export and Unity Setup

Export as BarGraphDisplay.fbx with standard settings. In Unity, use the same layer/camera/RenderTexture pattern (256x512px). The BarFill object is scaled on Z by the driver script. TextMeshPro overlays the digital value.

3.5 C# Driver Script

```
using UnityEngine;
using TMPro;
public class BarGraphDriver : MonoBehaviour
{
[Header("Scale")]
public float scaleMin = 0f, scaleMax = 100f;
public string units = "%";
[Header("Setpoints")]
public float setpointLo = 17f, setpointHi = 92f;
[Header("References")]
public Transform barFill;
public Renderer barFillRenderer;
public TextMeshPro digitalReadout;
[Header("Colors")]
public Color normalColor = new Color(0f, 0.8f, 0.27f);
public Color cautionColor = new Color(1f, 0.67f, 0f);
public Color alarmColor = new Color(1f, 0.15f, 0.15f);
public float damping = 10f;
private float _targetScale, _currentValue;
private MaterialPropertyBlock _mpb;
void Awake() { _mpb = new MaterialPropertyBlock(); }
public void SetValue(float value)
{
_currentValue = Mathf.Clamp(value, scaleMin, scaleMax);
_targetScale = Mathf.InverseLerp(scaleMin, scaleMax, _currentValue);
}
void Update()
{
Vector3 s = barFill.localScale;
s.z = Mathf.Lerp(s.z, _targetScale, Time.deltaTime * damping);
barFill.localScale = s;
```

```
Color c = (_currentValue <= setpointLo || _currentValue >= setpointHi)
? alarmColor
: (_currentValue <= setpointLo + 5f || _currentValue >= setpointHi - 5f)
? cautionColor : normalColor;
barFillRenderer.GetPropertyBlock(_mpb);
_mpb.SetColor("_EmissionColor", c * 3f);
_mpb.SetColor("_BaseColor", c);
barFillRenderer.SetPropertyBlock(_mpb);
if (digitalReadout)
digitalReadout.text = _currentValue.ToString("F1") + " " + units;
}
}
```

Section 3: Strip Chart Recorder

February 2026

Part 4: Strip Chart Recorder

4.1 Real-World Reference

Strip chart recorders provide continuous time-history traces of key parameters, enabling operators to detect slow drifts, monitor transients, and verify post-trip behavior. A typical Westinghouse 4-Loop PWR contains 15-25 strip chart recorders (Yokogawa, Honeywell, or ABB models). Multi-pen (2-4 pen) continuous paper recorders advance chart paper at 1-6 inches per hour (selectable), drawing colored traces on pre-printed grid paper.

Common applications: NR-45 (2-pen selectable NIS recorder), T-hot/T-cold trend recorders per loop, pressurizer pressure and level recorder, SG level trend recorders, containment pressure/temperature recorder, and steam/feedwater flow recorders. The visible chart window shows approximately 2-4 hours of recent data.

4.2 Specifications

Property

Specification

Visible Window

~6-8" W x 6-8" H (model dependent)

Chart Paper

Pre-printed grid, white, green/blue gridlines

Grid

10 major divisions, 5 minor subdivisions each

Pen Colors

Pen 1: Red, Pen 2: Blue, Pen 3: Green, Pen 4: Purple

Chart Speed

Selectable: 1, 2, 4, 6 in/hr (normal), up to 60 in/hr (fast)

Housing

Metal case, glass window, flush panel mount

4.3 Blender 5.0 Modeling

Add Cube for housing: X=20cm, Y=1cm, Z=20cm. Inset front face 0.8cm for frame, then 0.3cm for window bezel, extrude back -0.15cm. Dark gray metallic material (#2A2A2A, Roughness 0.5, Metallic 0.3). Name "RecorderBody".

Add Plane for chart paper: 17cm x 17cm inside window. White material (#FAFAF5, Roughness 0.95). Name "ChartPaper". UV Unwrap.

Create tileable grid texture: 512x512px, white background, light green (#C8E6C9) major gridlines every 51px, lighter green (#E8F5E9) minor lines every 10px.

Add label planes above (parameter name/legend) and on left/right edges (scale markings).

Pen traces are NOT modeled in Blender - they are generated dynamically in Unity (see C# script).

Export as StripChartRecorder.fbx. In Unity: dedicated layer, camera, 512x512px RenderTexture.

4.4 C# Driver Script

Creates a live scrolling strip chart by writing pen traces into a dynamic Texture2D:

using UnityEngine;

public class StripChartDriver : MonoBehaviour

```
{
[Header("Chart Config")]
public int texWidth = 512, texHeight = 512;
public float scrollSpeed = 2f;  // pixels/sec
public float sampleInterval = 0.5f;
[Header("Pen 1")]
public float pen1Min = 100f, pen1Max = 650f;
public Color pen1Color = Color.red;
[Header("Pen 2")]
public float pen2Min = 100f, pen2Max = 650f;
public Color pen2Color = Color.blue;
public Renderer chartRenderer;
public string texProperty = "_DetailAlbedoMap";
private Texture2D _tex;
private Color32[] _px;
private float _scrollAccum, _sampleTimer;
private float _v1, _v2;
public void SetPen1(float v) { _v1 = Mathf.Clamp(v, pen1Min, pen1Max); }
public void SetPen2(float v) { _v2 = Mathf.Clamp(v, pen2Min, pen2Max); }
void Start()
{
_tex = new Texture2D(texWidth, texHeight, TextureFormat.RGBA32, false);
_tex.filterMode = FilterMode.Point;
_px = new Color32[texWidth * texHeight];
var clr = new Color32(0,0,0,0);
for (int i = 0; i < _px.Length; i++) _px[i] = clr;
_tex.SetPixels32(_px); _tex.Apply();
chartRenderer.material.SetTexture(texProperty, _tex);
}
void Update()
{
_sampleTimer += Time.deltaTime;
_scrollAccum += scrollSpeed * Time.deltaTime;
while (_scrollAccum >= 1f) { _scrollAccum--; ScrollLeft(); }
if (_sampleTimer >= sampleInterval)
{
```

```
_sampleTimer = 0f;
DrawPen(texWidth-1, _v1, pen1Min, pen1Max, pen1Color);
DrawPen(texWidth-1, _v2, pen2Min, pen2Max, pen2Color);
_tex.SetPixels32(_px); _tex.Apply();
}
}
void ScrollLeft()
{
for (int y = 0; y < texHeight; y++)
for (int x = 0; x < texWidth-1; x++)
_px[ytexWidth+x] = _px[ytexWidth+x+1];
var c = new Color32(0,0,0,0);
for (int y = 0; y < texHeight; y++)
_px[y*texWidth+texWidth-1] = c;
}
void DrawPen(int col, float val, float mn, float mx, Color clr)
{
int row = Mathf.RoundToInt(Mathf.InverseLerp(mn, mx, val) * (texHeight-1));
row = Mathf.Clamp(row, 0, texHeight-1);
Color32 c = clr;
for (int d = -1; d <= 1; d++)
_px[Mathf.Clamp(row+d,0,texHeight-1)*texWidth+col] = c;
}
}
```

Section 4: Annunciator Window Tile

February 2026

Part 5: Annunciator Window Tile

5.1 Real-World Reference

Annunciator windows are rectangular backlit alarm tiles in grid arrays across the top of every control panel section. They are the primary alarm notification system. When an alarm triggers, the window flashes rapidly with an audible horn. The operator acknowledges (ACK button) to stop flashing (steady-on, horn silenced). When the condition clears, the window slow-flashes until the operator resets it. A typical 4-Loop PWR has 500-1000 individual annunciator windows.

Each tile is a translucent plastic faceplate engraved with 2-3 lines of uppercase text (e.g., "PZR PRESS HI", "SG 1 LEVEL LO-LO"). The backlight is white or amber. The alarm sequence follows NUREG-0700 and ANSI/ISA-18.1: INACTIVE (dark) -> ALERTING (fast flash + horn) -> ACKNOWLEDGED (steady-on) -> CLEARING (slow flash) -> RESET (dark).

5.2 Specifications

Property

Specification

Tile Size

~2.75" W x 1.75" H per window

Grid Spacing

~0.125" gaps between tiles

Faceplate

Translucent white/amber plastic, engraved black text

Alert Flash Rate

2-4 Hz (fast flash)

Clear Flash Rate

0.5-1 Hz (slow flash)

Acknowledged

Steady-on (no flash)

Horn

Audible buzzer during ALERTING, silenced by ACK

5.3 Typical Annunciator Labels

Panel Section

Example Labels

Reactor / RCS

PZR PRESS HI, PZR LEVEL LO, T-AVG HI, RCS FLOW LO

Reactor Protection

REACTOR TRIP, ROD BOTTOM, NIS POWER RANGE HI

Steam Generators

SG 1 LEVEL HI, SG PRESS LO (SAFETY INJ)

CVCS

VCT LEVEL LO, CHARGING PUMP TRIP, LETDOWN ISOL

Containment

CNTMT PRESS HI-HI, CNTMT SUMP LEVEL HI

ESF

SAFETY INJECTION ACTUATED, CNTMT SPRAY ACTUATED

Electrical

4KV BUS UNDERVOLTAGE, DIESEL GEN START

5.4 Blender 5.0 Modeling

Model a single tile as a reusable prefab; instanced in Unity to build grid arrays.

Add Cube: X=7.0cm, Y=0.8cm, Z=4.45cm. Inset front face 0.15cm, extrude back -0.1cm. Dark frame material (#2A2A2A). Name "AnnunTile".

Add Plane faceplate: 6.5cm x 4.0cm, positioned at Y=0.32cm. Name "AnnunFace". UV Unwrap.

Create 256x128px alarm text texture: white background, black uppercase text, 2-3 lines. Apply to faceplate with Emission node (Strength=0.0 initial, driven by script).

Export as AnnunciatorTile.fbx.

TIP: Emission strength by state: INACTIVE=0.0, ALERTING/ACKNOWLEDGED=5.0-8.0, CLEARING oscillates 0.0-5.0.

5.5 Unity Setup

Import FBX, create prefab. Instantiate grid of tiles in a parent GameObject with 0.3cm gaps.

Each tile gets AnnunciatorTileDriver script linked to simulation alarm conditions.

Render via "AnnunCam" to 1024x256px RenderTexture (per row of ~8 tiles). Display on GUI canvas at top of panel.

5.6 C# Driver Script

```
using UnityEngine;

public enum AnnunciatorState { Inactive, Alerting, Acknowledged, Clearing }

public class AnnunciatorTileDriver : MonoBehaviour

{

public string alarmId;

public AnnunciatorState state = AnnunciatorState.Inactive;

public float alertFlashHz = 3f, clearFlashHz = 0.7f;

public float maxEmission = 6f;

public Color backlightColor = Color.white;

public Renderer faceplateRenderer;

public AudioSource hornAudio;

private MaterialPropertyBlock _mpb;

private float _t;

void Awake() { _mpb = new MaterialPropertyBlock(); }

public void TriggerAlarm()

{
```

```
if (state == AnnunciatorState.Inactive || state == AnnunciatorState.Clearing)
{ state = AnnunciatorState.Alerting; if (hornAudio) hornAudio.Play(); }
}
public void Acknowledge()
{
if (state == AnnunciatorState.Alerting)
{ state = AnnunciatorState.Acknowledged; if (hornAudio) hornAudio.Stop(); }
}
public void ConditionCleared()
{ if (state == AnnunciatorState.Acknowledged) state = AnnunciatorState.Clearing; }
public void Reset()
{ if (state == AnnunciatorState.Clearing) state = AnnunciatorState.Inactive; }
void Update()
{
_t += Time.deltaTime;
float em = 0f;
switch (state)
{
case AnnunciatorState.Inactive: em = 0f; break;
case AnnunciatorState.Alerting:
em = Mathf.Sin(_t  alertFlashHz  2f * Mathf.PI) > 0 ? maxEmission : 0; break;
case AnnunciatorState.Acknowledged: em = maxEmission; break;
case AnnunciatorState.Clearing:
em = Mathf.Sin(_t  clearFlashHz  2f * Mathf.PI) > 0 ? maxEmission : 0; break;
}
faceplateRenderer.GetPropertyBlock(_mpb);
_mpb.SetColor("_EmissionColor", backlightColor * em);
faceplateRenderer.SetPropertyBlock(_mpb);
}
}
```

Section 5: Indicator Lamp

February 2026

Part 6: Indicator Lamp (Status Light)

6.1 Real-World Reference

Indicator lamps are the small round status lights embedded throughout every Westinghouse PWR control panel. The industry standard is the GE ET-16 indicating lamp, approximately 1.0-1.125 inches diameter, with a colored translucent dome lens (red, green, amber, white, or blue). Press-to-test capability verifies lamp function. Available in incandescent and LED retrofit models.

Lamps show binary component status: pump running (green) vs. stopped (red), valve open (green) vs. closed (red), breaker status, diesel generator status, ESF actuation. They are embedded in mimic bus diagrams - panel artwork showing simplified system schematics with lamps at component locations.

Color conventions per NUREG-0700: RED = danger, abnormal, trip; GREEN = normal, safe, running; AMBER = caution, transition; WHITE = neutral status; BLUE = bypass, maintenance.

6.2 Specifications

Property

Specification

Diameter

1.0-1.125" (25-29mm)

Shape

Round, domed translucent lens

Colors

Red, Green, Amber, White, Blue

Lamp Type

Incandescent (original) or LED (retrofit)

Push-to-Test

Press lens cap to verify function

Mounting

Panel cutout, snap-in/threaded collar

6.3 Blender 5.0 Modeling

Add Cylinder (32 vertices): X=2.86cm, Y=1.0cm, Z=2.86cm. Inset front face 0.15cm for bezel. Chrome material (#808080, Roughness 0.25, Metallic 1.0). Name "LampHousing".

Add UV Sphere, delete back hemisphere. Scale to ~2.5cm diameter dome. Position at Y=0.3cm. Name "LampLens".

Apply translucent material per color. Example RED: Base Color #CC0000, Roughness 0.15, Alpha 0.85, Blend Mode Alpha Blend. Emission Color #FF0000, Emission Strength 0.0 (driven by script).

Export as IndicatorLamp.fbx. Create Unity prefab variants for each color.

6.4 Unity Setup

Create prefab variants: RedLamp, GreenLamp, AmberLamp, WhiteLamp, BlueLamp.

For mimic panels: create 2D system schematic background sprite, place lamp prefabs at component locations.

Each lamp gets IndicatorLampDriver. Render via dedicated camera to RenderTexture (128x128 per lamp, or 1024x512 for a full mimic panel).

6.5 C# Driver Script

```
using UnityEngine;
public class IndicatorLampDriver : MonoBehaviour
{
public enum LampMode { Off, SteadyOn, Flashing }
public LampMode mode = LampMode.Off;
public Color lensColor = Color.red;
public float onEmission = 5f, flashHz = 2f;
public Renderer lensRenderer;
private MaterialPropertyBlock _mpb;
private float _t;
void Awake() { _mpb = new MaterialPropertyBlock(); }
public void SetState(bool on)
{ mode = on ? LampMode.SteadyOn : LampMode.Off; }
void Update()
{
_t += Time.deltaTime;
float em = mode switch
{
LampMode.Off => 0f,
LampMode.SteadyOn => onEmission,
LampMode.Flashing => Mathf.Sin(_tflashHz2f*Mathf.PI) > 0 ? onEmission : 0f,
_ => 0f
};
lensRenderer.GetPropertyBlock(_mpb);
_mpb.SetColor("_EmissionColor", lensColor * em);
lensRenderer.SetPropertyBlock(_mpb);
}
}
```

Section 6: Digital Numeric Readout

February 2026

Part 7: Digital Numeric Readout

7.1 Real-World Reference

Digital numeric readouts are 7-segment LED displays presenting precise numerical values. The most prominent on a Westinghouse 4-Loop PWR are the control rod position indicators, showing steps withdrawn (0-228) for each control bank (A, B, C, D) and shutdown banks in bright red 3-digit 7-segment displays. Other readouts show reactor power (% RTP), precise pressurizer pressure, and NIS channel readings.

A typical display has 3-5 digits with red, green, or amber LED segments. The housing is a black rectangular panel with a tinted lens. Unlit segments are faintly visible as dark ghost outlines - a characteristic visual detail that adds authenticity. Displays update at 1-4 Hz.

7.2 Specifications

Property

Specification

Digit Height

0.56" (14mm) or 1.0" (25mm) per digit

Digits

3, 4, or 5 digits + optional sign/decimal

Segment Type

7-segment LED

Colors

Red (most common), Green, Amber

Background

Black, non-reflective tinted lens

Update Rate

1-4 Hz

7.3 Parameter Formats

Parameter

Format

Rod Position (steps)

3 digits: 000-228 (red)

Reactor Power (%RTP)

4 digits: 000.0-120.0 (red/green)

Pressurizer Pressure

4 digits: 0000-2500 psig (green)

T-avg

4 digits: 000.0-650.0 degF (green)

Delta-I

4 digits with sign: 00.0 (red)

7.4 Blender 5.0 Modeling

Add Cube for housing. 3-digit (rod position): X=8.0cm, Y=0.8cm, Z=4.0cm. Black matte (#111111, Roughness 0.8). Inset front face 0.3cm, extrude back -0.2cm. Name "DigitalBody".

Add Plane for digit window: X=6.8cm, Z=3.0cm, inside recess at Y=0.15cm. Dark tinted material (#0A0503, Roughness 0.3). Name "DigitWindow". UV Unwrap.

Optionally create ghost-segment texture: 256x128px showing "888" in very dark red (#1A0500) for the background layer.

Export as DigitalReadout.fbx. In Unity, use TextMeshPro with a 7-segment LED font (DSEG7-Classic recommended) for the digit display.

TIP: For maximum authenticity, use the free DSEG7-Classic font. Import as TMP font asset. Set HDR color with emission for the LED glow effect. Render ghost segments as a dim TMP text behind the active digits.

7.5 Unity Setup

Import FBX, create prefab. Add TextMeshPro 3D Text child positioned just in front of DigitWindow (+Y by 0.001).

TMP settings: Font=DSEG7-Classic, Alignment=Center/Middle, Color=Red HDR (intensity ~3.0).

Optionally add second TMP for ghost segments: text="888" (or "8888."), color = very dim red.

Render via dedicated camera to 256x128px RenderTexture.

7.6 C# Driver Script

```
using UnityEngine;
using TMPro;
public class DigitalReadoutDriver : MonoBehaviour
{
[Header("Display Config")]
public int digitCount = 3;
public int decimalPlaces = 0;
public bool leadingZeros = true;
public bool showSign = false;
[Header("Range")]
public float minValue = 0f, maxValue = 228f;
[Header("References")]
public TextMeshPro displayText;
public TextMeshPro ghostText;
[Header("Appearance")]
public Color digitColor = new Color(1f, 0.1f, 0.05f);
public float emissionIntensity = 3f;
private float _val;
void Start()
{
if (displayText) displayText.color = digitColor * emissionIntensity;
```

```
if (ghostText)
{
string g = new string('8', digitCount);
if (decimalPlaces > 0) g = g.Insert(digitCount-decimalPlaces, ".");
ghostText.text = g;
ghostText.color = digitColor * 0.08f;
}
}
public void SetValue(float value)
{
_val = Mathf.Clamp(value, minValue, maxValue);
if (!displayText) return;
string fmt = decimalPlaces > 0
? (leadingZeros ? new string('0', digitCount-decimalPlaces-1)+"0."+new string('0', decimalPlaces) : "F"+decimalPlaces)
: (leadingZeros ? new string('0', digitCount) : "F0");
string txt = Mathf.Abs(_val).ToString(fmt);
if (showSign) txt = (_val >= 0 ? "+" : "-") + txt;
displayText.text = txt;
}
}
```

Section 7: Guidelines and Quick Reference

February 2026

Part 8: Universal Integration Guidelines

8.1 Standard FBX Export Settings

Setting

Value

Scale

0.01

Apply Scalings

FBX All

Forward Axis

-Z Forward

Up Axis

Y Up

Apply Modifiers

Checked

Smoothing

Face

8.2 Unity Layer Architecture

Layer

Purpose

GaugeAnalog

Analog round gauges (companion manual)

GaugeEdgewise

Vertical edgewise meters

GaugeBarGraph

Vertical bar graph displays

GaugeStripChart

Strip chart recorders

GaugeAnnunciator

Annunciator tile arrays

GaugeStatusLamp

Indicator lamp / mimic panels

GaugeDigital

Digital numeric readouts

Each layer has a dedicated Orthographic camera rendering only that layer to its own RenderTexture. Clear Flags = Solid Color (black). Depth value must not conflict with main camera.

8.3 RenderTexture Sizing

Gauge Type

Recommended RT Size

Analog Round Gauge

512 x 512 px

Vertical Edgewise Meter

256 x 512 px

Vertical Bar Graph

256 x 512 px

Strip Chart Recorder

512 x 512 px

Annunciator Grid (per row of 8)

1024 x 128 px

Indicator Lamp (single)

128 x 128 px

Indicator Lamp (mimic panel)

1024 x 512 px

Digital Readout

256 x 128 px

8.4 Performance Optimization

Group gauges that share one RenderTexture where possible (e.g., 4 edgewise meters in a bank).

Use MaterialPropertyBlock instead of unique Material instances per gauge.

Render gauge cameras on-demand (camera.Render()) for slow-changing values; every frame for strip charts.

Use GPU instancing on annunciator tile mesh renderers.

8.5 Lighting Standard

Primary: Area Light, 50W, 4000-5000K warm white, above-front.

Fill: Point/Area Light, 10-15W, below-front to soften bezel shadows.

Ambient: Low warm gray (#2A2520) simulating control room wall reflections.

No directional light (avoid harsh shadows on panel instruments).

8.6 NUREG-0700 Color Standards

Color

Meaning

Red

Danger, abnormal, trip, alarm, safety concern

Green

Normal, safe, running, open

Amber/Yellow

Caution, warning, intermediate state

White

Neutral status, information only

Blue

Bypass, suppressed, maintenance

Part 9: Quick Reference and Checklists

9.1 Parameter-to-Gauge Selection Guide

Parameter

Recommended Indicator(s)

Temperatures (T-hot/T-cold/T-avg)

Analog round gauge OR edgewise + strip chart

Pressurizer Pressure

Analog round + digital readout + strip chart

Pressurizer Level

Bar graph + strip chart

SG Water Levels

Bar graph + edgewise backup + strip chart

Reactor Power (%RTP)

Digital readout + analog round gauge

Rod Position (steps)

Digital readout (dedicated)

Flow Rates (RCS, CVCS)

Edgewise meter OR bar graph

Component Status

Indicator lamp (binary on/off)

Alarms

Annunciator window tile

Nuclear Instrumentation

Edgewise meter (log scale) + strip chart

9.2 File Naming Convention

Asset Type

Pattern

Blender source

GaugeType_ParamName.blend

FBX export

GaugeType_ParamName.fbx

Scale texture

GaugeType_Scale_Range.png

Alarm text texture

Annun_System_AlarmText.png

Unity prefab

GaugeType_ParamName.prefab

C# driver

GaugeTypeDriver.cs

RenderTexture

RT_GaugeType_ParamName

9.3 Implementation Checklist

Identify real-world gauge type and parameter range from this manual.

Model in Blender 5.0 per type-specific instructions.

Apply materials with Principled BSDF and emission nodes where needed.

Set up orthographic camera and standard lighting in Blender.

Export FBX with standardized settings (Section 8.1).

Import into Unity 6.3 with correct settings.

Assign to dedicated layer.

Create dedicated orthographic camera for that layer.

Create RenderTexture at correct size (Section 8.3).

Assign RenderTexture to camera Target Texture.

Attach C# driver script to animated element.

Wire SetValue() to the simulation physics engine.

Add RawImage on GUI canvas, assign RenderTexture.

Test full parameter range for correct behavior.

Verify visual consistency with other panel gauges.

--- End of Manual ---

T-HOT Gauge Blender Manual

February 2026

CRITICAL: Master the Atom

Westinghouse 4-Loop PWR Simulator

2.5D Temperature Gauge

Blender 5.0 to Unity 6 Instruction Manual

T_HOT - Hot Leg Temperature Indicator

Range: 100 degF - 650 degF

Revision 1.0 - February 2026

For use with Blender 5.0.1 and Unity 6.3 (6000.1.x)

Table of Contents

Part 0 - Overview and Context

0.1 What We Are Building

This manual guides you through creating a 2.5D temperature gauge in Blender 5.0 and importing it into Unity 6 for use on the Reactor Manual Operation GUI (Key 1). The gauge replicates the T_HOT (Hot Leg Temperature) indicator found on a real Westinghouse 4-Loop PWR main control board.

The term 2.5D means the gauge has real 3D geometry with depth, bevels, and dimensionality, but is designed to be viewed from a fixed front-facing camera angle. This gives it a premium, physically realistic look compared to a flat 2D texture, while remaining lightweight enough for a UI overlay.

0.2 What the Real Gauge Looks Like

On a Westinghouse 4-Loop PWR main control board, temperature indicators for T_HOT, T_COLD, and T_AVG are typically round analog gauges, approximately 6 inches in diameter with the following features:

A dark (usually black or dark grey) face plate with white scale markings and numerals.

A chrome or brushed-metal circular bezel (the outer ring).

A red or orange needle/pointer that sweeps from roughly the 7 oclock position (minimum) to the 5 oclock position (maximum), covering approximately 270 degrees of arc.

Scale range for T_HOT: 100 degF to 650 degF, with major divisions every 50 degF and minor tick marks every 10 degF.

A label reading T-HOT degF centred below the needle pivot.

A glass cover with subtle reflections.

0.3 Parts We Will Model

Part

Geometry

Material

Notes

Bezel (outer ring)

Torus / extruded circle

Brushed chrome metal

Gives the gauge its frame

Face plate

Cylinder (very thin)

Dark matte grey/black

Background behind markings

Scale markings

Thin rectangular prisms

White emissive

Major + minor tick marks

Numerals

3D text or texture

White emissive

100, 150, 200 ... 650

Needle

Thin tapered wedge

Red/orange emissive

This is the animated part

Needle hub

Small cylinder

Chrome metal

Centre cap over the needle pivot

Glass cover

Slightly convex disc

Transparent + glossy

Adds realism with reflection

Back plate

Cylinder

Dark metal

Seals the back, adds depth

0.4 Software Requirements

Software

Version

Download

Blender

5.0.1 or later

blender.org/download

Unity

6.3 (6000.1.x) or later

unity.com/download

Unity FBX Importer

Built-in (Package Manager)

Included with Unity

Part 1 - Blender 5.0 Essentials for Beginners

1.1 First Launch

Step 1: Download and install Blender 5.0.1 from blender.org/download.

Step 2: Launch Blender. You will see a splash screen. Click anywhere outside the splash to dismiss it.

Step 3: You are now in the default scene containing a cube, a camera, and a light. We will delete all three shortly.

1.2 Understanding the Interface

Blenders interface has several key areas. Here is what you see on the default screen:

Area

Location

Purpose

3D Viewport

Centre (largest area)

Where you model, rotate, and view your 3D objects

Outliner

Top-right

Lists every object in your scene (like a file explorer for 3D objects)

Properties Panel

Bottom-right

Settings for the selected object: materials, physics, render, etc.

Timeline

Bottom strip

Used for animation keyframes (we will use this later)

Header / Top Bar

Very top

File menu, workspace tabs (Layout, Modeling, Sculpting, etc.)

Toolbar

Left side of 3D Viewport

Quick-access tools (move, rotate, scale, etc.)

1.3 Essential Navigation (Mouse + Keyboard)

Blender relies heavily on keyboard shortcuts. These are the ones you absolutely must know:

Viewport Navigation

Middle Mouse Button (MMB) drag - Orbit (rotate) the view around the scene

Shift + MMB drag - Pan (slide) the view left/right/up/down

Scroll wheel - Zoom in and out

Numpad 1 - Front view (looking at the front of your object)

Numpad 3 - Right side view

Numpad 7 - Top-down view

Numpad 5 - Toggle perspective / orthographic view

Numpad 0 - Camera view (what the camera sees)

Home key - Zoom to fit everything in the viewport

TIP: If you dont have a numpad, go to Edit > Preferences > Input and check Emulate Numpad. This maps numpad keys to the number row.

Object Operations

Left click - Select an object

A - Select all / Deselect all (toggle)

X or Delete - Delete selected object (confirm in popup)

G - Grab (move) - press X, Y, or Z after to constrain to an axis

R - Rotate - press X, Y, or Z after to constrain to an axis

S - Scale - press X, Y, or Z after to constrain to an axis

Shift + A - Add menu (add new objects: mesh, curve, text, etc.)

Tab - Toggle between Object Mode and Edit Mode

Ctrl + Z - Undo (works for almost everything)

Ctrl + S - Save your file

Edit Mode Operations

1, 2, 3 (top row) - Switch between Vertex, Edge, Face select mode

E - Extrude selected faces/edges/vertices

I - Inset faces (creates a smaller face inside a selected face)

Ctrl + R - Loop cut (adds a ring of edges around geometry)

S then number - Scale by exact amount (e.g., S then 0.5 then Enter = half size)

WARNING: Always make sure you know which MODE you are in. The mode selector is in the top-left of the 3D Viewport. Object Mode is for moving whole objects. Edit Mode is for modifying the mesh geometry of a single object.

1.4 Clean Up the Default Scene

Step 1: Press A to select all objects (cube, camera, light).

Step 2: Press X and confirm Delete in the popup.

Step 3: You now have a completely empty scene. This is our starting point.

Step 4: Press Ctrl + S and save the file as T_HOT_Gauge.blend in your project folder.

TIP: Save frequently. Blender can crash, and losing work is painful. Use Ctrl+S often.

Part 2 - Modelling the Gauge in Blender

2.1 Set Up Your Workspace

Step 1: Press Numpad 1 to switch to Front View.

Step 2: Press Numpad 5 to switch to Orthographic mode (removes perspective distortion; good for precision work).

Step 3: We will model everything centred at the origin (0, 0, 0). The gauge face will be on the XY plane, with depth going along the Z axis (towards you).

2.2 Create the Back Plate

The back plate is a thin cylinder that forms the rear of the gauge housing.

Step 1: Press Shift + A to open the Add menu.

Step 2: Navigate to Mesh > Cylinder.

Step 3: IMMEDIATELY after adding the cylinder, look at the bottom-left of the 3D viewport. You will see a small floating panel labelled Add Cylinder. Click it to expand if needed.

Step 4: Set the following values in that panel:

Parameter

Value

Why

Vertices

64

Smooth circular shape (default 32 looks slightly faceted)

Radius

1.0

This defines our gauge as 2 Blender units in diameter

Depth

0.05

Very thin disc - this is the back plate

Location Z

-0.05

Pushes it slightly behind the origin

Step 5: In the Outliner (top-right), double-click on the object name Cylinder and rename it to BackPlate.

TIP: Always name your objects as you create them. With 8+ objects, unnamed Cylinder.001, Cylinder.002 etc. becomes confusing fast.

2.3 Create the Face Plate

The face plate is a slightly thinner disc that sits in front of the back plate. This is the dark surface behind the scale markings.

Step 1: Press Shift + A > Mesh > Cylinder.

Step 2: In the Add Cylinder panel, set:

Parameter

Value

Vertices

64

Radius

0.92

Depth

0.02

Location Z

0.0

Step 3: Rename this object to FacePlate.

The face plate has a smaller radius (0.92 vs 1.0) so there is a visible ring of the back plate / bezel around it.

2.4 Create the Bezel (Outer Ring)

The bezel is the chrome ring around the outside edge of the gauge. We will use a Torus for this.

Step 1: Press Shift + A > Mesh > Torus.

Step 2: In the Add Torus panel, set:

Parameter

Value

Why

Major Segments

64

Smooth circle

Minor Segments

16

Smooth cross-section of the ring tube

Major Radius

1.0

Matches the overall gauge diameter

Minor Radius

0.08

Thickness of the bezel ring tube

Step 3: The torus is created lying flat. We need it to sit at Z = 0.01 (just in front of the face plate). Press G, then Z, then type 0.01, then Enter.

Step 4: Rename this object to Bezel.

2.5 Create the Scale Markings (Tick Marks)

This is the most involved step. We need to create tick marks arranged in an arc from approximately 225 deg (7 oclock) to -45 deg (5 oclock), covering 270 deg of arc. There are 11 major marks (every 50 degF from 100 to 650) and 55 minor marks (every 10 degF).

2.5.1 Create the Major Tick Template

Step 1: Press Shift + A > Mesh > Cube.

Step 2: Press S to scale, then type 0.01 and press Enter. This makes a tiny cube.

Step 3: Now we will scale it non-uniformly. Press S, then X, then type 3, then Enter. This stretches it into a thin rectangle along the X axis.

Step 4: Press S, then Y, then type 0.3, then Enter. This flattens it in Y.

Step 5: Press S, then Z, then type 0.5, then Enter. This gives it a bit of depth.

Step 6: You should now have a small elongated rectangular shape - about 0.06 units long in X, thin in Y and Z. This is one major tick mark.

Step 7: Rename it to MajorTick.

2.5.2 Position the First Tick

The tick marks live near the rim of the face plate. The first tick (100 degF) needs to be at the 225 deg position (7 oclock). The centre of the gauge is at the origin.

Step 1: Move the tick to the rim: Press G, then X, then type 0.78, then Enter. This places it near the inner edge of the bezel.

Step 2: We now need to rotate it around the gauge centre. The 225 deg position means we rotate the tick 225 deg around the Z axis, but using the 3D cursor as pivot.

Step 3: First, make sure the 3D cursor is at the origin. Press Shift + S and choose Cursor to World Origin.

Step 4: Set the pivot point to 3D Cursor: In the header of the 3D Viewport, find the pivot point dropdown (it shows a dot icon by default, labelled Median Point). Change it to 3D Cursor.

Step 5: With MajorTick selected, press R, then Z, then type 225, then Enter. The tick mark is now at the 7 oclock position, 0.78 units from centre.

Step 6: The tick should also be oriented radially (pointing towards centre). Press R, then Z, then type 225, then Enter, but this time make sure you are in Object Mode and using Individual Origins as pivot. Actually, a simpler approach: select the tick and press Ctrl + A > Rotation to apply the rotation.

TIP: This is getting complex for a first-timer. An easier approach follows below using the Array + Empty method.

2.5.3 Easier Method: Array Modifier with Empty

Blender has a powerful trick for arranging objects in a circle: the Array modifier combined with an Empty object set to rotate. This lets you create all tick marks from a single object.

Step 1: First, undo everything from 2.5.1 and 2.5.2 (Ctrl+Z until you are back to just the FacePlate, BackPlate, and Bezel).

We will start fresh with a cleaner method:

Step 2: Create the Empty (rotation driver): Press Shift + A > Empty > Plain Axes. Rename it to TickRotator. Make sure it is at the origin (0, 0, 0).

Step 3: We need 56 tick marks total (55 minor intervals across 270 deg). Each interval is 270/55 = 4.909 deg. Set the TickRotators Z rotation: In the Properties panel on the right, expand the Object Properties tab (orange square icon). Under Transform > Rotation Z, type: 4.909

Step 4: Create the tick mark: Press Shift + A > Mesh > Cube.

Step 5: Scale it into a thin rectangle: Press S, then type 0.006, then Enter. Then press S, X, type 4, Enter. Then press S, Z, type 0.3, Enter. You now have a thin elongated bar.

Step 6: Move it to the rim: Press G, then Y, then type 0.8, then Enter. The tick is now 0.8 units above centre.

Step 7: Rename it to TickMark.

Step 8: Apply the Array modifier: With TickMark selected, go to the Properties panel > Modifiers tab (wrench icon). Click Add Modifier > Array.

Step 9: In the Array modifier settings, set:

Setting

Value

Fit Type

Fixed Count

Count

56

Relative Offset

UNCHECK this (turn it off)

Object Offset

CHECK this (turn it on)

Object

TickRotator (select from the dropdown)

Step 10: You should now see 56 tick marks arranged in a circular arc. They start from the top (12 oclock) and sweep 270 deg around.

Step 11: Rotate the whole thing so tick #1 starts at 7 oclock (225 deg): Select the TickMark object. Set the pivot point to 3D Cursor (which should be at origin). Press R, Z, type 135, Enter. This rotates the starting position to approximately the 7 oclock position.

Step 12: Apply the Array modifier: With TickMark selected, hover over the Array modifier in the Properties panel and press Ctrl + A (or click the dropdown arrow > Apply). The tick marks are now real geometry.

NOTE: After applying the Array modifier, all 56 ticks are part of one mesh. Every 5th tick is a major mark (corresponding to 50 degF intervals). We will differentiate major vs minor ticks using materials or by scaling them after the fact. For now, this gives you the correct positioning.

2.5.4 Differentiate Major and Minor Ticks

Step 1: With TickMark selected, press Tab to enter Edit Mode.

Step 2: Press 3 (top row) to switch to Face Select mode.

Step 3: You need to select every 5th tick (these are the major marks at 100, 150, 200... 650 degF). Hold Ctrl and click on the faces of every 5th tick mark (the 1st, 6th, 11th, 16th, 21st, 26th, 31st, 36th, 41st, 46th, 51st ticks - 11 total).

Step 4: Once selected, press S, then the axis perpendicular to the face length (this will be context-dependent on orientation), then type 1.8, then Enter. This scales those ticks to be ~1.8x longer than the minor ticks.

Step 5: Press Tab to return to Object Mode.

TIP: If selecting every 5th tick manually feels tedious, you can instead create two separate tick arrays: one with 11 ticks at 27 deg spacing (majors) and one with 44 ticks at the remaining positions (minors), each with different lengths. Use the same TickRotator Empty approach but with different rotation values.

2.6 Create the Numerals

The numerals (100, 150, 200... 650) sit just inside the major tick marks. There are 11 numerals. We will use Blenders Text object and then convert to mesh.

Step 1: Switch to Front View (Numpad 1) and Orthographic (Numpad 5).

Step 2: Press Shift + A > Text.

Step 3: A text object labelled Text appears. Press Tab to enter Edit Mode for the text.

Step 4: Select all the default text and delete it. Type: 100

Step 5: Press Tab to exit Edit Mode.

Step 6: With the text selected, go to Properties panel > Object Data Properties (the A icon that looks like a font character).

Step 7: Under Font, you can leave the default (BFont) or load a cleaner sans-serif font. Under Geometry > Extrude, set to 0.005 to give the text slight depth.

Step 8: Set the text Size to 0.08 in the same panel.

Step 9: Under Paragraph > Alignment, set Horizontal to Center.

Step 10: Position this text at the correct location for the 100 label: move it to approximately (X: -0.48, Y: -0.48, Z: 0.01), which is the inner portion near the 7 oclock major tick.

Step 11: Rotate the text to face outward/be readable: Press R, Z, and rotate to match the angle of the 100 degF tick mark.

Step 12: Repeat steps 2-11 for each numeral: 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, 650. Position each at the corresponding major tick mark location.

TIP: This is 11 text objects to create and position manually. It is tedious but straightforward. Once you have all 11, select them all, right-click and choose Convert To > Mesh. This turns the text curves into mesh geometry, which exports more reliably to FBX.

Step 13: Add the label. Create one more text object reading T-HOT degF. Set size to 0.06, centre it horizontally, and position it at approximately (0, -0.35, 0.01) - centred below the needle pivot. Rename it to Label_T_HOT.

2.7 Create the Needle

The needle is the part that will be animated in Unity. It must be a separate object with its pivot (origin) at the centre of the gauge.

Step 1: Press Shift + A > Mesh > Cube.

Step 2: Press Tab to enter Edit Mode.

Step 3: We need a tapered shape - wide at the pivot and thin at the tip. Press 1 to switch to Vertex Select mode.

Step 4: The default cube has 8 vertices. Select the top 4 vertices (the ones at +Y). Press S, X, type 0.3, Enter. This narrows the top, creating a taper.

Step 5: Press Tab to return to Object Mode.

Step 6: Now scale the whole object: Press S, X, type 0.015, Enter. Press S, Y, type 0.7, Enter. Press S, Z, type 0.008, Enter.

Step 7: This gives you a thin, flat, tapered pointer about 0.7 units long.

Step 8: Move the needle so its BASE (wide end) is at the origin and it extends upward: Press G, Y, type 0.35, Enter.

Step 9: CRITICAL: Set the origin to the base of the needle. Go to the header menu: Object > Set Origin > Origin to 3D Cursor. Since the 3D cursor is at the world origin (0,0,0), and the base of the needle is near the origin, this sets the pivot point at the gauge centre. The needle will rotate around this point.

WARNING: The needles origin (pivot point) MUST be at the centre of the gauge (world origin), not at the centre of the needle mesh. If you get this wrong, the needle will orbit weirdly in Unity instead of sweeping like a real gauge pointer.

Step 10: Move the needle slightly forward in Z so it sits above the face plate: Press G, Z, type 0.015, Enter.

Step 11: Rename this object to Needle.

2.8 Create the Needle Hub (Centre Cap)

Step 1: Press Shift + A > Mesh > Cylinder.

Step 2: Set: Vertices: 32, Radius: 0.05, Depth: 0.03, Location Z: 0.02.

Step 3: Rename to NeedleHub.

2.9 Create the Glass Cover

The glass gives the gauge a realistic look - a slightly convex transparent disc in front of everything.

Step 1: Press Shift + A > Mesh > UV Sphere.

Step 2: Set: Segments: 32, Rings: 16, Radius: 0.95.

Step 3: This creates a full sphere. We only want the front cap. Press Tab to enter Edit Mode.

Step 4: Press 3 for Face Select. Press A to select all faces.

Step 5: Switch to Right view (Numpad 3). Use Box Select (B key, then drag) to deselect the front-facing cap. We want to DELETE everything except the front hemisphere/cap.

Step 6: Actually, an easier approach: switch back to Front view. Press A to deselect all. Then use Circle Select (C key) or Box Select (B key) to select only the back half of the sphere (everything at Z < 0). Press X > Faces to delete them.

Step 7: Now flatten the remaining cap: press Tab to exit Edit Mode. Press S, Z, type 0.15, Enter. This squishes the hemisphere into a subtle dome.

Step 8: Position it at Z = 0.025 (just in front of everything else): Press G, Z, type 0.025, Enter.

Step 9: Rename to GlassCover.

TIP: If the sphere approach is too complex, you can also just use a cylinder with Vertices: 64, Radius: 0.94, Depth: 0.005 and skip the dome. A flat glass disc still looks good.

2.10 Verify Your Object List

Check the Outliner panel. You should have these objects:

Object Name

Type

BackPlate

Cylinder

FacePlate

Cylinder

Bezel

Torus

TickMark

Cube (arrayed, applied)

(11 numeral objects)

Text converted to Mesh

Label_T_HOT

Text converted to Mesh

Needle

Cube (tapered)

NeedleHub

Cylinder

GlassCover

Sphere (modified)

TickRotator

Empty (can delete now)

Step 1: You can safely delete the TickRotator empty now - select it and press X.

Step 2: Press Ctrl + S to save.

Part 3 - Materials and Appearance

3.1 How Materials Work in Blender

Each object can have one or more materials that define its colour, shininess, transparency, etc. When we export to Unity, these materials come along and we can either use them or replace them with Unity materials.

To add a material to an object: Select the object. Go to the Properties panel > Material Properties tab (the sphere icon). Click New.

3.2 Material Assignments

Create and assign the following materials. For each one, select the object, create a New material, rename it, and set the values in the Surface section (which defaults to Principled BSDF):

Object

Material Name

Base Color

Metallic

Roughness

Other

BackPlate

MAT_DarkMetal

Hex: 1A1A1A

0.9

0.5

-

FacePlate

MAT_GaugeFace

Hex: 0D0D0D

0.0

0.8

Nearly pure black, matte

Bezel

MAT_Chrome

Hex: C0C0C0

1.0

0.15

Mirror-like chrome

TickMark

MAT_WhiteEmissive

Hex: FFFFFF

0.0

0.5

Emission: 0.5 (faint glow)

Numerals

MAT_WhiteEmissive

Same as ticks

-

-

Reuse the same material

Label_T_HOT

MAT_WhiteEmissive

Same as ticks

-

-

Reuse the same material

Needle

MAT_NeedleRed

Hex: CC2200

0.0

0.4

Emission: 1.0 (glowing red)

NeedleHub

MAT_Chrome

Same as Bezel

-

-

Reuse

GlassCover

MAT_Glass

Hex: FFFFFF

0.0

0.05

Alpha: 0.08, see below

3.2.1 Setting Up the Glass Material

Step 1: Select the GlassCover object.

Step 2: In Material Properties, click New. Rename to MAT_Glass.

Step 3: Set Base Color to white (FFFFFF).

Step 4: Set Roughness to 0.05 (very glossy).

Step 5: Set Alpha to 0.08 (nearly fully transparent).

Step 6: CRITICAL: Under the materials Settings section (scroll down in the material panel), find Blend Mode and change it from Opaque to Alpha Blend.

Step 7: Also in Settings, check Backface Culling so you only see the front face of the glass.

3.3 Preview Your Gauge

Step 1: In the 3D Viewport header, find the viewport shading buttons (four circles in the top-right of the viewport). Click the rightmost one - Material Preview (or press Z and select Material Preview).

Step 2: You should now see your gauge with colours and materials applied. The black face plate with white ticks, chrome bezel, and red needle should be clearly visible.

Step 3: Orbit around (MMB drag) to verify everything looks correct from the front.

Step 4: Press Ctrl + S to save.

Part 4 - Final Cleanup Before Export

4.1 Smooth Shading

Cylinders and tori look faceted by default. We need to set Smooth Shading on the curved objects.

Step 1: Select BackPlate. Right-click > Shade Smooth.

Step 2: Repeat for FacePlate, Bezel, NeedleHub, and GlassCover.

Step 3: Do NOT smooth-shade the TickMark or Needle objects - they should remain flat-shaded (sharp edges look correct for these).

4.2 Apply All Transforms

WARNING: This step is CRITICAL for a clean export to Unity. If you skip this, your objects may appear at wrong scales, rotations, or positions in Unity.

Step 1: Press A to select all objects.

Step 2: Press Ctrl + A and choose All Transforms. This bakes the current position, rotation, and scale into the mesh data and resets the object transforms to identity (Location 0, Rotation 0, Scale 1).

4.3 Parent All to an Empty (Gauge Root)

For clean organisation in Unity, we want all gauge parts to be children of a single parent object.

Step 1: Press Shift + A > Empty > Plain Axes. Rename it to GaugeRoot. Position at (0, 0, 0).

Step 2: Select ALL gauge objects (BackPlate, FacePlate, Bezel, TickMark, all numerals, Label, Needle, NeedleHub, GlassCover) but NOT the GaugeRoot empty yet.

Step 3: Now ALSO select GaugeRoot (Shift + click it). It must be the LAST thing selected - this makes it the parent.

Step 4: Press Ctrl + P > Object. This parents all selected objects to the GaugeRoot empty.

Step 5: In the Outliner, you should now see GaugeRoot with all parts nested under it as children. The Needle should be a direct child of GaugeRoot (not nested under anything else) because Unity will rotate the Needle independently.

Step 6: Press Ctrl + S to save.

Part 5 - Exporting from Blender 5.0

5.1 FBX Export Settings

Step 1: Go to File > Export > FBX (.fbx).

Step 2: In the file browser that opens, navigate to your Unity projects Assets folder. Create a subfolder called Models if one doesnt exist. Example path: C:\Users\craig\Projects\Critical\Assets\Models\

Step 3: Set the filename to T_HOT_Gauge.fbx.

Step 4: In the export settings panel on the right side, configure:

Setting

Value

Why

Selected Objects

Check this ON

Only export our gauge, not hidden junk

Scale

1.0

Keeps Blender units = Unity units (1 = 1 metre)

Apply Scalings

FBX All

Bakes all scale transforms into the FBX

Forward

-Z Forward

Corrects Blenders Z-up to Unitys coordinate system

Up

Y Up

Unity uses Y as the up axis

Apply Transform

Check this ON

Critical: prevents rotation issues in Unity

Mesh > Smoothing

Face

Preserves our smooth/flat shading choices

Mesh > Apply Modifiers

Check this ON

Bakes any remaining modifiers

Armature

Uncheck (not needed)

We have no skeleton/bones

Animation

Uncheck for now

We will animate in Unity, not Blender

Step 5: Click Export FBX.

WARNING: Before clicking Export, make sure you have all objects selected in the viewport (press A), because we checked Selected Objects. If nothing is selected, nothing will export.

5.2 Save an Operator Preset

So you do not have to set all these options again every time:

Step 1: Before clicking Export, click the + button next to the presets dropdown at the top of the export panel.

Step 2: Name it Unity FBX Export.

Step 3: Next time you export, just select this preset and all settings will be pre-filled.

Part 6 - Importing into Unity 6

6.1 Import the FBX

Step 1: Open your Unity project (Critical).

Step 2: If you exported directly into Assets/Models/, Unity will have already detected the file. Switch to Unity and wait for the import progress bar to finish.

Step 3: If you exported elsewhere, drag and drop the T_HOT_Gauge.fbx file into the Assets/Models/ folder in the Unity Project window.

Step 4: Click on the imported T_HOT_Gauge asset in the Project window. The Inspector will show Model Import Settings.

6.2 Import Settings

Step 1: In the Model tab of Import Settings:

Setting

Value

Why

Scale Factor

1

If your gauge appears tiny, try 100

Convert Units

Checked

Converts Blender metres to Unity units

Import BlendShapes

Unchecked

Not needed

Import Visibility

Checked

Preserves visibility states

Import Cameras

Unchecked

We deleted the camera in Blender

Import Lights

Unchecked

We deleted the light in Blender

Step 2: Click Apply at the bottom of the Inspector.

6.3 Extract and Fix Materials

Step 1: Click the Materials tab in the Import Settings.

Step 2: Under Material Creation Mode, select Import via MaterialDescription.

Step 3: Click Extract Materials and save them to a Materials subfolder (e.g., Assets/Models/Materials/).

Step 4: Now select each extracted material in the Project window and adjust its Shader and properties in the Inspector. Unitys default URP Lit shader works well:

Blender Material

Unity Shader

Key Settings

MAT_Chrome

URP/Lit

Metallic: 1.0, Smoothness: 0.85, Color: C0C0C0

MAT_DarkMetal

URP/Lit

Metallic: 0.9, Smoothness: 0.5, Color: 1A1A1A

MAT_GaugeFace

URP/Lit

Metallic: 0, Smoothness: 0.2, Color: 0D0D0D

MAT_WhiteEmissive

URP/Lit

Emission: FFFFFF at intensity 0.5

MAT_NeedleRed

URP/Lit

Emission: CC2200 at intensity 1.0, Color: CC2200

MAT_Glass

URP/Lit

Surface Type: Transparent, Alpha: 0.08, Smoothness: 0.95

Step 5: Click Apply on each material after editing.

6.4 Place the Gauge in the Scene

Step 1: Drag the T_HOT_Gauge prefab from the Project window into the Hierarchy or Scene view.

Step 2: Verify the gauge appears correctly. If it is rotated oddly, select the root object and set Rotation to (0, 0, 0).

Step 3: If scale is wrong (too large or too small), adjust the Scale Factor in the Import Settings and click Apply again. A common fix is to set Scale Factor to 100 if the gauge appears as a tiny dot.

Part 7 - Animating the Needle in Unity

7.1 Understanding the Animation Approach

The needle needs to rotate around the Z axis (the axis pointing out of the gauge face) based on the T_HOT temperature value from the simulation. We have two choices:

Script-driven rotation (recommended): A C# script reads the current T_HOT value and sets the needles rotation. This is simple, direct, and integrates naturally with your existing simulation engine.

Unity Animator with animation clips: Useful for pre-baked visual flourishes (startup sweep, oscillation) but overkill for a value-driven indicator.

We will use the script-driven approach as it connects directly to your physics engines output values.

7.2 The Rotation Mapping

The needle sweeps 270 deg of arc across the temperature range 100 degF to 650 degF:

Temperature (degF)

Needle Angle (deg from 12 oclock)

Gauge Position

100 (minimum)

225 deg (or -135 deg)

7 oclock

375 (midpoint)

90 deg (or -270 deg)

12 oclock (top centre)

650 (maximum)

-45 deg (or 315 deg)

5 oclock

The mapping formula is:

float fraction = (temperature - 100f) / (650f - 100f); // 0 to 1

```
float angleDeg = 225f - (fraction * 270f);              // 225 deg to -45 deg
needleTransform.localRotation = Quaternion.Euler(0, 0, angleDeg);
```

7.3 Create the GaugeNeedleDriver Script

Create a new C# script in your Unity project. Place it in Assets/Scripts/UI/ (or wherever your GUI scripts live).

Here is the complete script:

```
using UnityEngine;
/// <summary>
/// Drives a 2.5D gauge needle rotation based on a float value.
/// Attach to the Needle child object of the imported gauge.
/// </summary>
public class GaugeNeedleDriver : MonoBehaviour
{
[Header("Gauge Range")]
[Tooltip("Minimum value on the gauge scale")]
public float minValue = 100f;
[Tooltip("Maximum value on the gauge scale")]
public float maxValue = 650f;
[Header("Sweep Angles (degrees from 12 o'clock, clockwise positive)")]
[Tooltip("Angle at minimum value (7 o'clock = 225)")]
public float angleAtMin = 225f;
[Tooltip("Angle at maximum value (5 o'clock = -45)")]
public float angleAtMax = -45f;
[Header("Smoothing")]
[Tooltip("How quickly the needle tracks the target (higher = faster)")]
[Range(1f, 20f)]
public float smoothSpeed = 5f;
// Current displayed value (smoothed)
private float _currentAngle;
/// <summary>
/// Call this from your simulation to update the gauge.
/// </summary>
public void SetValue(float temperature)
{
float clamped = Mathf.Clamp(temperature, minValue, maxValue);
float fraction = (clamped - minValue) / (maxValue - minValue);
float targetAngle = Mathf.Lerp(angleAtMin, angleAtMax, fraction);
```

```
_currentAngle = Mathf.Lerp(_currentAngle, targetAngle,
Time.deltaTime * smoothSpeed);
transform.localRotation = Quaternion.Euler(0f, 0f, _currentAngle);
}
private void Start()
{
_currentAngle = angleAtMin; // Start at minimum
transform.localRotation = Quaternion.Euler(0f, 0f, _currentAngle);
}
}
```

7.4 Attach the Script

Step 1: In the Hierarchy, expand the T_HOT_Gauge object until you find the Needle child object.

Step 2: Select the Needle object.

Step 3: Drag the GaugeNeedleDriver script onto it (or use Add Component > GaugeNeedleDriver).

Step 4: In the Inspector, verify the default values: Min Value: 100, Max Value: 650, Angle At Min: 225, Angle At Max: -45, Smooth Speed: 5.

7.5 Test It

To test without the full simulation running, you can add a temporary test script or modify GaugeNeedleDriver to include an Update test:

```
// Add this temporarily to GaugeNeedleDriver for testing:
[Header("Debug")]
public bool testMode = false;
[Range(100f, 650f)]
public float testTemperature = 100f;
private void Update()
{
if (testMode) SetValue(testTemperature);
}
```

Check Test Mode in the Inspector, enter Play mode, and drag the Test Temperature slider. The needle should sweep smoothly across the gauge face.

Part 8 - Integrating with the Reactor GUI

8.1 Placement on the Manual Operation Panel

The Reactor Manual Operation GUI is activated by pressing Key 1. The gauge needs to appear as part of this panels visual layout.

8.1.1 Option A: World-Space UI (Recommended for 2.5D)

Since the gauge is a 3D model, it looks best rendered in 3D space rather than as a flat UI overlay. The approach is:

Step 1: Position the gauge as a child of the panels 3D layout (if the GUI exists as a 3D panel in world space), or place it

in front of a dedicated camera.

Step 2: Create a dedicated Render Texture camera that views only the gauge (using Layer filtering).

Step 3: Display the Render Texture on a Raw Image UI element on the reactor GUI canvas.

8.1.2 Option B: Direct 3D Placement

If your reactor panel is a 3D object in the scene, you can simply parent the gauge to the panel and position it at the appropriate location (where the T_HOT indicator sits on the control board).

8.2 Render Texture Approach (Detailed Steps)

Step 1: Create a Render Texture: In the Project window, right-click > Create > Render Texture. Name it RT_GaugeT_HOT. Set resolution to 512 x 512 (sufficient for a gauge).

Step 2: Create a Gauge Camera: In the Hierarchy, right-click > Camera. Name it GaugeCamera_T_HOT.

Step 3: Position the camera directly in front of the gauge, looking at it. Set Projection to Orthographic. Adjust Orthographic Size so the gauge fills the camera view with a small margin.

Step 4: Set the cameras Target Texture to RT_GaugeT_HOT.

Step 5: Set the cameras Culling Mask to a dedicated layer (e.g., create a layer called GaugeRender). Move all gauge objects to this layer. This prevents the gauge from appearing in the main camera and the main scene from appearing in the gauge camera.

Step 6: Set the Background Type to Solid Color and make it transparent (RGBA: 0, 0, 0, 0) so the gauge has no background.

Step 7: On the Reactor GUI Canvas, add a UI > Raw Image element where the T_HOT gauge should appear. Set its Texture to RT_GaugeT_HOT.

Step 8: Size the Raw Image to match the gauges desired screen size on the panel.

8.3 Connecting to the Simulation

Your existing simulation engine updates temperature values every frame. You need to call GaugeNeedleDriver.SetValue() with the current T_HOT value.

In whatever script manages the GUI updates (likely your ReactorGUIManager or ManualOperationPanel), add a reference to the gauge:

[SerializeField] private GaugeNeedleDriver tHotGauge;

Then, in the update loop where you already update text displays, add:

tHotGauge.SetValue(currentState.T_hot);

Where currentState.T_hot is whatever field holds the hot leg temperature in your current simulation state struct or class.

8.4 Lighting the Gauge

For the gauge to look good in isolation (rendered by its own camera), add a small lighting setup:

Step 1: Add a Point Light or Spot Light near the gauge camera, aimed at the gauge face. Set intensity to something subtle. Set this light to the same GaugeRender layer so it only affects the gauge.

Step 2: A second fill light at lower intensity from the side adds depth and makes the chrome bezel gleam.

Step 3: Experiment with light colour - a slightly warm white (hex: FFF5E0) mimics the incandescent control room lighting found in real Westinghouse control rooms.

Part 9 - Polish and Realism Details

9.1 Add Needle Oscillation (Optional)

Real gauge needles have a slight oscillation due to sensor noise and fluid turbulence. You can add this for realism:

// In GaugeNeedleDriver.SetValue(), add noise:

float noise = Mathf.PerlinNoise(Time.time * 2f, 0f) - 0.5f;

float noiseAmplitude = 0.3f; // degrees

_currentAngle += noise * noiseAmplitude;

9.2 Add a Red Danger Zone

On a real T_HOT gauge, the region above ~620 degF is often marked with a red arc to indicate approach to design limits. You can model this as a thin red arc mesh in Blender, or add it as a texture on the face plate.

9.3 Make It Reusable for Other Gauges

The same gauge model and GaugeNeedleDriver script can be reused for T_COLD, T_AVG, T_PZR, and pressure gauges by simply changing the minValue, maxValue, and label. To make variants:

Step 1: Duplicate the gauge prefab in Unity.

Step 2: Change the GaugeNeedleDriver values on the new instance.

Step 3: For the label, either swap the Label_T_HOT text mesh for a new one, or use a TextMeshPro component in Unity positioned over the gauge face.

Gauge

Min Value

Max Value

Unit

T_HOT (Hot Leg)

100

650

degF

T_COLD (Cold Leg)

100

650

degF

T_AVG (Average)

100

650

degF

T_PZR (Pressurizer)

100

700

degF

P_PZR (PZR Pressure)

0

2500

psig

PZR Level

0

100

%

9.4 Performance Considerations

Each Render Texture camera adds a draw call. For a panel with 6-8 gauges, this is negligible on modern hardware. The mesh itself is very low-poly (a few hundred triangles at most) and will have zero performance impact.

Part 10 - Troubleshooting

Problem

Likely Cause

Fix

Gauge appears as tiny dot in Unity

Scale mismatch

Set Scale Factor to 100 in Unity Import Settings

Gauge is rotated 90 deg in Unity

Coordinate system difference

In Blender export: set Forward: -Z, Up: Y, Apply Transform: ON

Materials are pink/missing

Shader not set for your render pipeline

Re-assign materials using URP/Lit shader

Needle rotates on wrong axis

Axis confusion between Blender and Unity

Try Quaternion.Euler(0, 0, angle) vs (angle, 0, 0) vs (0, angle, 0)

Needle rotates from its centre, not its base

Origin not set to base in Blender

In Blender: select Needle, set 3D Cursor to origin, Object > Set Origin > Origin to 3D Cursor

Glass is fully opaque

Blend mode not set

Select MAT_Glass in Unity, set Surface Type to Transparent

Ticks/numerals not visible

Emission not enabled

Enable Emission on the material and set colour to white

FBX export fails or is empty

Objects not selected

Press A to select all before exporting with Selected Objects checked

Smooth shading looks weird on ticks

Smooth shade on flat objects

Right-click > Shade Flat on tick/needle objects

Appendix A - Quick Reference Cheat Sheet

Blender Keyboard Shortcuts Used in This Guide

Shortcut

Action

Shift + A

Add object menu

Tab

Toggle Object / Edit Mode

G

Grab (move)

R

Rotate

S

Scale

X / Delete

Delete selection

A

Select all / Deselect all

Ctrl + A

Apply transforms menu

Ctrl + P

Parent menu

Ctrl + R

Loop cut

Ctrl + Z

Undo

Ctrl + S

Save

Numpad 1 / 3 / 7

Front / Right / Top view

Numpad 5

Toggle Perspective / Orthographic

Numpad 0

Camera view

MMB drag

Orbit view

Shift + MMB drag

Pan view

Scroll wheel

Zoom

Shift + S

Snap / Cursor menu

Z

Shading mode pie menu

B

Box select

C

Circle select

1, 2, 3 (Edit Mode)

Vertex, Edge, Face select

E

Extrude

I

Inset faces

Unity GaugeNeedleDriver API

Member

Type

Description

minValue

float

Minimum gauge scale value (default: 100)

maxValue

float

Maximum gauge scale value (default: 650)

angleAtMin

float

Rotation angle at minimum (default: 225 deg)

angleAtMax

float

Rotation angle at maximum (default: -45 deg)

smoothSpeed

float

Lerp speed for needle movement (default: 5)

SetValue(float)

method

Call each frame with current temperature