

Conversation Design Samples: Kozan Soykal

Using game-design interactions to demonstrate content design for conversational AI

1) Conversation Flow Examples

Flow A — Map Hotspots & Hover Label (ClickBG scene)

Kozan's input

- Create a simple **ClickBG** scene with a fixed background.
- Some regions are **hotspots** with an **always-visible** faint outline.
- On **hover**: show **full outline + translucent fill + a label** near the hotspot.
- On **click**: trigger **VNOverlayLoader.ShowVN("VNScene")** as an **additive overlay**, pausing base input until closed.
- Use **polygonal shapes** I can edit by hand in the Scene view.
- Add a **timeline gate** so hotspots can be enabled/disabled by story phase (start at phase 1).
- Start with **one** hotspot we can clone.

Assistant thought pattern (summarized)

- Convert requirements into a prefab pattern: collider = truth source; visuals auto-generated.
- Make the label robust across canvas/camera settings; support fixed screen placement near a world anchor.
- Keep it timeline-aware now, and ready to plug into global **GameState** later.
- Anticipate common Unity pitfalls (compile deps, canvas camera, layout clamping) and build toggles/fallbacks.

Resulting structure / architecture

- **Hotspot2D** (MonoBehaviour):
 - Requires **PolygonCollider2D** (isTrigger).
Renders **outline** via **LineRenderer** + **fill mesh** via a tiny triangulator.
Hover → highlight & label; Click → call **VNOverlayLoader.ShowVN("VNScene")**.
Optional **labelAnchor** transform + **labelOffsetWorld**.
- **HotspotHoverUI** (screen-space UI helper):
 - Places label near a supplied world position using **viewport mapping**.

- **clampToCanvas** toggle; **pixelOffset**; rebuilds layout before measuring.
 - Backward-compatible overload for **Show(text, pos)** that defaults to **Camera.main**.
- **ClickBGTimeline** (singleton):
 - **IsActive(min,max)** gating with **phase=1** by default; future hook to **GameState**.
- **Editor workflow**:
 - Place **BG sprite**, add **Sorting Layers** (**Background, Hotspots**).
 - Place **Canvas** and a small **TMP label panel** (pivot ~ **(0.5, 0)**).
 - Create **Hotspot_EastBarn**: draw polygon points; assign optional **LabelAnchor**.
 - Duplicate to create more hotspots; set **minTimeline/maxTimeline** as needed.

Error / fallback handling in this flow (what I asked for and how it was handled)

- **Missing dependency** (**Triangulator**) → provide a **self-contained** version inside **Hotspot2D.cs**.
 - **Label “drift” / wrong position** → use **viewport-based mapping**, ensure **canvas.worldCamera** for Screen-Space Camera, add **optional clamping**.
 - **Signature break (CS7036)** → two paths: (1) add **overload** to keep old calls, (2) pass **Camera.main**.
-

Flow B — Combat UI: Dynamic Portraits & Left Bench

Kozan’s input

- In **UI_Combat_Simple**, the **active portrait** should update automatically: show **Derfel** when it’s Derfel’s turn, **Birne** when it’s Birne’s turn.
- The **left bench** should display the **inactive character(s)** with **HP/MP/IP bars** and **portrait**.
- When it’s Derfel’s turn, left bench shows Birne (and vice versa).
- During **enemy turns**, bench shows **both PCs**.
- Prefer **drop-in scripts** and **wiring steps**; if you need to inspect a **.cs**, ask first.

Assistant thought pattern (summarized)

- Subscribe UI to the game’s existing **turn and stats events** rather than polling.
- Treat UI as “**dumb**” (display only): have a single controller react to **TurnManager** and **UnitRuntime.OnStatsChanged**.
- Avoid editor fragility: fill portraits/bars from **unit data**; don’t hard-wire sprite

references in UI prefabs.

Resulting structure / architecture

- **UI_Combat_Simple**
 - Subscribes to `TurnManager.OnTurnChanged(activeUnit)` to swap the **active portrait** automatically.
 - Holds references to a **LeftBench** presenter and the **ActivePortrait** presenter.
- **ActivePortraitPresenter**
 - `Bind(UnitRuntime pc)` → populate portrait from `UnitDefinitionSO` / runtime portrait field.
 - Listens to `OnStatsChanged` to reflect buffs/debuffs if needed.
- **LeftBenchPresenter**
 - `Show(IEnumerable<UnitRuntime> inactivePCs)` → renders bench items for the **non-active** PCs.
 - Each bench item binds bars to **HP/MP/IP** and a portrait supplied by unit data.
- **Wiring**
 - On **player turn**: `LeftBench.Show(other PCs)`.
 - On **enemy turn**: `LeftBench.Show(all PCs)`.
 - Portrait lookups: first try runtime's portrait ref; fall back to definition.

Benefits

- Minimal coupling; new PCs slot in via data.
 - No hard-coded images in the UI; everything flows from **unit data + events**.
 - Predictable bench behavior across **player** and **enemy** rounds.
-

2) Voice & Tone Guidelines

These do's and don'ts are taken from how I direct the assistant in this project. They define the agent's voice I expect.

Set scope and turn-taking

- **Do:** Just confirm you read, no other response. Do not provide any code yet.
- **Don't:** Include explanations, extra notes, or commentary when I've asked only for an acknowledgment.

Ask before inspecting code or structure

- **Do:** "Let me know if you need to look at what's inside the `.cs` files."
- **Do:** "If you're not aware how pieces connect, ask to see the hierarchy/structure."
- **Don't:** Assume file contents or scene wiring.

Deliver copy-pasteables

- **Do:** "Give me a **drop-in** for `Hotspot2D.cs`."
- **Do:** "Provide wiring instructions, step-by-step."
- **Don't:** Hand-wave with generalities like "you can figure it out in the editor."

Respect existing systems & assumptions

- **Do:** "Reuse the same VN overlay scene as Dungeon; keep input paused while VN is open."
- **Do:** "Carry prior context (Derfel & Birne, `UnitRuntime`, `OnStatsChanged`)."
- **Don't:** Replace or rewrite core systems without a reason.

Offer choices; state trade-offs

- **Do:** "Two options: add a backward-compatible overload **or** pass `Camera.main` at the call site."
- **Don't:** Present a single path as the only solution when alternatives exist.

Be concise and actionable

- **Do:** "Save as `FileName.cs`. Paste exactly. Then do: 1) add component, 2) set pivot, 3) assign label."
- **Don't:** Provide large, unstructured paragraphs without steps or defaults.

3) Error & Fallback Handling — Guidelines I expect you (the assistant) to follow

Ask-first triggers

1. **Code context unknown** → "If you do not remember the last state of a `.cs`, ask to see the current file content."
2. **Structure unknown** → "If you're not aware of how components connect (Unity hierarchy, web `.css` / `.html` linkage, build graph), ask to see the structure."
3. **Environment affects behavior** → Ask about **Canvas mode**, **world camera**,

sorting layers, assembly definitions, platform.

4. **Risk of overwriting defaults** → Confirm before changing **control visuals**, **input routing**, or **shared assets**.

Verification checklists (use before proposing fixes)

- **Unity UI placement**: Canvas mode, `canvas.worldCamera`, panel **pivot/anchors**, layout rebuild, optional **clamp** toggle.
- **Compile errors**: Confirm **file/class names** match; check **asmdefs**, **namespaces**, **Editor folder** pitfalls.
- **Event-driven UI**: Verify there is an **event** to subscribe (e.g., `OnTurnChanged`, `OnStatsChanged`) before suggesting polling.
- **Data flow**: Portraits and stats should come from **runtime/definition data**, not hard-wired prefab references.

Fix strategy (how to propose changes)

- Prefer **minimal, reversible** changes first.
- When a change **breaks signatures**, offer **two clear options** (e.g., *overload* vs. *update call site*).
- When behavior may vary by setup, offer a **named toggle** (e.g., `clampToCanvas`) instead of silent assumptions.
- Provide **drop-in files** with exact filenames and **editor steps** (what to add, where to assign).

Acknowledge uncertainty (what to say)

- “Based on your current description, I’m assuming X. If that’s wrong, show me Y and I’ll adjust.”
- “I can’t see your latest `UI_Combat_Simple.cs`—paste it if the binding differs.”

Escalate / handoff

- If a fix requires risky refactors or unclear product intent, **pause** and ask for **human decision**: “Do you want A (faster) or B (cleaner API)?”
- If external constraints block progress (e.g., missing package/asset), document the **exact missing piece** and a **workable interim** (e.g., bundled helper).

Examples from this session

- Missing class → provided **self-contained** `Triangulator` to remove coupling.
- Label drift → verified canvas/camera, switched to **viewport mapping**, added **clamp** toggle.

- Signature error (CS7036) → gave **two** paths: overload **or** pass camera arg.
-

Appendix — Reusable Artifacts

- **Hotspot2D.cs** (polygon collider → outline/fill, hover label, click to VN overlay, optional label anchor).
- **HotspotHoverUI.cs** (viewport mapping, layout rebuild, clamp toggle, `Show(text, pos[, camera])`).
- **ClickBGTimeline.cs** (phase gate; fail-open in editor; hook for `GameState`).
- **Combat UI presenters** (Active portrait & Left bench bind to `TurnManager` + `UnitRuntime.OnStatsChanged`).
- **Data-first UI** (portraits & bars sourced from unit data, not prefab fields).