UNITY 6000.0.62F1 · COMPLETE OFFLINE DOCUMENTATION

Mood Arrow Developer Guide

Install. Edit. Reskin. Build your release.

szekipapa77@gmail.com



CHAPTER 01

Start here: Mood Arrow

Mood Arrow is a standalone editable Unity game template. Choose a feeling arrow, aim the first-person bow and release a shot at a townsfolk target. Their emotion changes how they react to nearby people. Complete the displayed scenario objective within the available arrows. Read the current goal rather than treating every target as an enemy or claiming an online social system.

Includes 25 curated levels, source-backed generation, a game-specific visual editor and validation, 4 guided helpers, local saving and portrait/landscape scenes. Keep the required private game Kit and publisher-owned Szekipapa77Ads layer. Provider SDK binaries are excluded. There is no bundled HLK, purchases, multiplayer server or remote LiveOps backend. Import only one game Kit into a project because the Kit classes share names.

CHAPTER 02

Requirements, installation and first play

Use Unity **6000.0.62f1**, URP **17.0.4** and Unity UI **2.0.0**. Install matching Android SDK/NDK/OpenJDK and WebGL modules in Unity Hub. Retain the supplied manifest and ProjectSettings.

1. Extract the standalone project to a short writable local path, then open it in the matching editor. Wait for package import and compilation.
2. For native unitypackage import, create a matching URP project and use Assets > Import Package > Custom Package. Import every game, Kit, Szekipapa77Ads and WebGL-template entry.
3. Run Szekipapa77 > Mood Arrow > Setup. It configures rendering and player settings; review its changes when integrating with existing work.
4. Active Input Handling must support the classic Input Manager: choose Old or Both. New Input System only is insufficient.
5. Open `Assets/Games/MoodArrow/Scenes/MoodArrow_Mobile.unity` and press Play. Follow the level card and first-level instructions.
6. Stop Play mode before editing level assets. Keep all .meta GUIDs. The game uses classic Unity UI Text; do not assume an extra TextMesh Pro import is required.

CHAPTER 03

Project folders and exact source map

All game-relative paths below start at `Assets/Games/MoodArrow`. Runtime UI is built in code; editing an unrelated static Canvas does not reskin it. Scripts contains game rules and views; Editor contains authoring tools; Kit supplies UI/audio/save/platform support; Resources/moodarrow contains level JSON, art, music and sound; Settings owns URP assets; Scenes contains both entry scenes.

- `Scripts/MaArt.cs`
- `Scripts/MaData.cs`
- `Scripts/MaGen.cs`
- `Scripts/MaLevels.cs`
- `Scripts/MaMeshes.cs`
- `Scripts/MaSfx.cs`
- `Scripts/MaSim.cs`
- `Scripts/MoodArrowGame.cs`
- `Scripts/MoodArrowGame.Hud.cs`
- `Scripts/MoodArrowGame.World.cs`

Main flow: MoodArrowGame.cs. UI and popup consumers: MoodArrowGame.Hud.cs. Model/generator/loading: MaData.cs, MaGen.cs, MaLevels.cs, MaSim.cs. Authoring: Editor/MoodArrowLevelEditor.cs. Shared ad settings live outside the game at Assets/Szekipapa77Ads/Resources/AdsSettings.asset.

CHAPTER 04

Gameplay, controls and progression

Choose a feeling arrow, aim the first-person bow and release a shot at a townsfolk target. Their emotion changes how they react to nearby people. Complete the displayed scenario objective within the available arrows. Read the current goal rather than treating every target as an enemy or claiming an online social system.

The initial bank has 25 curated levels. Read the supplied loader/generator before changing continuation rules. Play the guided opening, ordinary progress, completion and retry paths; preserve saved indexes during a visual reskin. The Mobile scene is portrait and the WebGL-named scene is landscape, but the supplied Build menu selects portrait for both build targets. Select landscape explicitly in your Build Profile if desired. Test a near-startup scene replacement as well as a normal restart.

CHAPTER 05

Tutorials and every helper

Fresh local progress triggers the first-level guidance. Newly unlocked helpers have introduction and guided-use flows. Reset only a test save when reviewing tutorials. A free guided first use can differ from ordinary charge consumption. Exact names/unlocks/free charges: Scripts/MoodArrowGame.cs; effect/help strings: Scripts/MoodArrowGame.Hud.cs.

Helper	Unlock	Free charges	Action and effect
Aid	4	2	Stuck? Aid turns your bow to the right person and picks the right arrow.
Rewind	6	3	Take back your last shot: everyone walks back and you get the arrow back.
Quiver	10	2	Adds 2 arrows to your quiver for this level.
Skip	14	1	Skips this level (you keep 1 star).

Activate the tool and perform its required target action: arming a tool alone does not demonstrate its effect. A successful use requires the stated preconditions from this game: show both activation and its actual effect. Exhausted charges may offer a rewarded ad; verify success/cancel/failure and grant a reward once only.

CHAPTER 06

Visual level editor: worked editing session

Open **Szekipapa77 > Mood Arrow > Level Editor**. Exact implementation: `Editor/MoodArrowLevelEditor.cs`.

Open the Mood Arrow editor and select an early level. Use Dup or Duplicate before editing. Change its visible name and one rule or board element with the exact controls below. Save explicitly and run the current entry through the editor validation and play controls. The per-game source-control table includes the current brush, geometry, target, rule and AI fields; do not treat hierarchy or resource IDs as display text.

1. Back up Resources/moodarrow/Levels/levels.json.
2. Duplicate a simple early level and edit one small element.
3. Use the game-specific validation controls and resolve every reported issue.
4. Save explicitly; stop and restart Play mode, then use Play this level or navigate to the edited index.
5. Check portrait and landscape and rebuild demo outputs after accepting data changes.

The exact current authoring controls, including tooltips and generator switches, follow. These are game-specific source references, not a generic editor feature list.

Source	Exact current code
`MoodArrowLevelEditor.cs:50`	`[MenuItem("Szekipapa77/Mood Arrow/Level Editor")]`
`MoodArrowLevelEditor.cs:53`	`[MenuItem("Szekipapa77/Mood Arrow/Validate All Levels")]`
`MoodArrowLevelEditor.cs:119`	`if (GUILayout.Button(_dirty ? "Save *" : "Save", EditorStyles.toolbarButton, GUILayout.Width(70))) Save();`
`MoodArrowLevelEditor.cs:120`	`if (GUILayout.Button("Reload", EditorStyles.toolbarButton, GUILayout.Width(60)) && (!_dirty || EditorUtility.DisplayDialog("Reload", "Discard unsaved changes?", "Discard", "Cancel"))) Load();`
`MoodArrowLevelEditor.cs:123`	`if (GUILayout.Button("Validate", EditorStyles.toolbarButton, GUILayout.Width(70)) && Cur != null) ValidateOne(_sel);`
`MoodArrowLevelEditor.cs:124`	`if (GUILayout.Button("Validate All", EditorStyles.toolbarButton, GUILayout.Width(85))) ValidateAll();`

MoodArrowLevelEditor.cs:127	if (GUILayout.Button(new GUIContent("Play this level", "Optional quick test: saves, opens the WebGL scene at this level and enters Play mode. Editing never needs it."), EditorStyles.toolbarButton, GUILayout.Width(100)) && Cur != null) Play(_sel + 1);
MoodArrowLevelEditor.cs:138	if (GUILayout.Button("Add")) { Record("add level"); var l = new List<MaLevel>(_file.levels); l.Insert(Mathf.Min(_sel + 1, l.Count), NewLevel()); _file.levels = l.ToArray(); _sel = Mathf.Min(_sel + 1, _file.levels.Length - 1); _person = -1; _dirty = true; _results.Clear(); }
MoodArrowLevelEditor.cs:139	if (GUILayout.Button("Dup") && Cur != null) { Record("duplicate level"); var l = new List<MaLevel>(_file.levels); var c = Cur.Clone(); c.name += " copy"; l.Insert(_sel + 1, c); _file.levels = l.ToArray(); _sel++; _dirty = true; _results.Clear(); }
MoodArrowLevelEditor.cs:140	if (GUILayout.Button("Del") && Cur != null && EditorUtility.DisplayDialog("Delete level", $"Delete level {_sel + 1} ({Cur.name})?", "Delete", "Cancel")) { Record("delete level"); var l = new List<MaLevel>(_file.levels); l.RemoveAt(_sel); _file.levels = l.ToArray(); _sel = Mathf.Clamp(_sel, 0, Mathf.Max(0, _file.levels.Length - 1)); _person = -1; _dirty = true; _results.Clear(); }
MoodArrowLevelEditor.cs:141	if (GUILayout.Button("Up") && _sel > 0) { Record("move level"); (_file.levels[_sel], _file.levels[_sel - 1]) = (_file.levels[_sel - 1], _file.levels[_sel]); _sel--; _dirty = true; _results.Clear(); }
MoodArrowLevelEditor.cs:142	if (GUILayout.Button("Down") && _sel < _file.levels.Length - 1) { Record("move level"); (_file.levels[_sel], _file.levels[_sel + 1]) = (_file.levels[_sel + 1], _file.levels[_sel]); _sel++; _dirty = true; _results.Clear(); }
MoodArrowLevelEditor.cs:175	EditorGUILayout.LabelField($"Level {_sel + 1}", EditorStyles.boldLabel);
MoodArrowLevelEditor.cs:177	string name = EditorGUILayout.TextField("Name", l.name);

MoodArrowLevelEditor.cs:179	int scene = EditorGUILayout.Popup("Decor", l.scene, SceneNames);
MoodArrowLevelEditor.cs:180	bool br = EditorGUILayout.Toggle("Breather", l.breather);
MoodArrowLevelEditor.cs:183	int budget = EditorGUILayout.IntSlider(new GUIContent("Arrows (budget)", "Arrows the player gets. The solver's optimum earns 3 stars."), l.budget, 1, 12);
MoodArrowLevelEditor.cs:184	if (GUILayout.Button("= opt + 1", GUILayout.Width(80))) budget = Mathf.Max(1, l.opt + 1);
MoodArrowLevelEditor.cs:186	string hint = EditorGUILayout.TextField(new GUIContent("First-time hint", "Shown on the level card the first time (use \\n for a new line)."), l.hint.Replace("\n", "\\n")).Replace("\\n", "\n");
MoodArrowLevelEditor.cs:189	EditorGUILayout.PrefixLabel(new GUIContent("Key arrows", "Arrows the solver may use (the player's bar offers every unlocked arrow)."));
MoodArrowLevelEditor.cs:214	EditorGUILayout.HelpBox("Top-down plaza, the player stands at the bottom edge looking up. Move: drag a person. Add: click an empty spot. " +
MoodArrowLevelEditor.cs:226	if (probs.Length > 0) EditorGUILayout.HelpBox("Problem: " + probs, MessageType.Error);
MoodArrowLevelEditor.cs:230	EditorGUILayout.HelpBox($"Solver: {res}\nDifficulty score {MaGen.Difficulty(l, res):0} / 100. 3 stars: {res.Moves} arrows, 2 stars: {res.Moves + 1}. Budget {l.budget}" + (l.budget < res.Moves ? " (TOO LOW!)" : "") + $".\nSolution: {path}",
MoodArrowLevelEditor.cs:233	else EditorGUILayout.HelpBox($"Cached optimum {l.opt} arrows ({l.states} states). Press Validate to re-check.", MessageType.None);
MoodArrowLevelEditor.cs:239	EditorGUILayout.LabelField("Mission (parts are done in order)", EditorStyles.boldLabel);
MoodArrowLevelEditor.cs:256	if (GUILayout.Button("X", GUILayout.Width(24)) && l.missions.Length > 1)

MoodArrowLevelEditor.cs:263	`if (GUILayout.Button("Add part", GUILayout.Width(100)) && l.missions.Length < 3)`
MoodArrowLevelEditor.cs:427	`if (_person < 0 || _person >= l.people.Length) { EditorGUILayout.LabelField("Click a person (Move brush) to edit their name, look and position.", EditorStyles.miniLabel); return; }`
MoodArrowLevelEditor.cs:433	`int look = EditorGUILayout.IntSlider("Look", p.look, 0, 47);`
MoodArrowLevelEditor.cs:434	`float x = EditorGUILayout.FloatField("x", p.x, GUILayout.Width(160));`
MoodArrowLevelEditor.cs:435	`float z = EditorGUILayout.FloatField("z (depth)", p.z, GUILayout.Width(180));`
MoodArrowLevelEditor.cs:467	`EditorGUILayout.LabelField("Difficulty curve (optimal arrows x parts; blue = breather, red = unsolvable). Click a bar to select.", EditorStyles.miniBoldLabel);`
MoodArrowLevelEditor.cs:487	`EditorGUILayout.LabelField("Auto Generate (same MaGen code as the runtime generator; every level is solver-checked)", EditorStyles.boldLabel);`
MoodArrowLevelEditor.cs:489	`_genCount = Mathf.Clamp(EditorGUILayout.IntField("Count", _genCount), 1, 60);`
MoodArrowLevelEditor.cs:490	`_genStart = Mathf.Max(1, EditorGUILayout.IntField("Start index (1-based)", _genStart));`
MoodArrowLevelEditor.cs:491	`_genSeed = EditorGUILayout.IntField("Seed", _genSeed);`
MoodArrowLevelEditor.cs:494	`_genFrom = EditorGUILayout.IntSlider("Start difficulty (arrows)", _genFrom, 1, 6);`
MoodArrowLevelEditor.cs:495	`_genTo = EditorGUILayout.IntSlider("End difficulty (arrows)", _genTo, 1, 6);`
MoodArrowLevelEditor.cs:498	`_genBreather = EditorGUILayout.IntSlider("Breather every", _genBreather, 0, 8);`
MoodArrowLevelEditor.cs:499	`_genPeople = EditorGUILayout.IntSlider("People", _genPeople, 2, 7);`

`MoodArrowLevelEditor.cs:500`	`_genParts = EditorGUILayout.IntSlider("Mission parts", _genParts, 1, 3);`
`MoodArrowLevelEditor.cs:503`	`EditorGUILayout.PrefixLabel("Key arrows");`
`MoodArrowLevelEditor.cs:508`	`_genScene = EditorGUILayout.Popup("Decor", _genScene + 1, sceneOpts) - 1;`
`MoodArrowLevelEditor.cs:509`	`_genReplace = EditorGUILayout.Toggle(new GUIContent("Replace range", "On: overwrite levels from the start index. Off: insert them there."), _genReplace);`
`MoodArrowLevelEditor.cs:522`	`if (GUILayout.Button("Generate", GUILayout.Height(26))) Generate();`

CHAPTER 07

Generator, insertion and replacement

Use the actual generation panel and its fields listed below. Set a reproducible seed and a small count, then review the start/index and replace option before clicking Generate. The exact generation implementation is reproduced separately with line references so append, insert and replacement behavior can be distinguished. Save and reopen the data file to verify the intended entry count; runtime generated continuation is not a project-asset save.

Source	Exact current code
`MoodArrowLevelEditor.cs:46`	`bool _genReplace;`
`MoodArrowLevelEditor.cs:509`	`_genReplace = EditorGUILayout.Toggle(new GUIContent("Replace range", "On: overwrite levels from the start index. Off: insert them there."), _genReplace);`
`MoodArrowLevelEditor.cs:568`	`if (_genReplace) { int rem = Mathf.Min(made.Count, list.Count - at); list.RemoveRange(at, rem); }`
`MoodArrowLevelEditor.cs:569`	`list.InsertRange(at, made);`

Worked extension: back up the bank, generate a small set at moderate difficulty using the exact per-game controls, validate them, then Save. Reopen the JSON and confirm the count changed. Do not assume every editor has the same insertion behavior: the actual implementation below distinguishes insert, append and replace. Runtime continuation does not automatically append new records to project JSON. Validation and bot scores help find problems but do not certify a pleasant human difficulty curve.

CHAPTER 08

Actual level JSON and identifiers

Canonical asset: `Assets/Games/MoodArrow/Resources/moodarrow/Levels/levels.json`. Array order controls visible numbering. Preserve numeric/boolean types, root version fields where present and resource/save identifiers. The actual first record is shown below; use the editor for structured coordinates and block/word data.


```
{
  "name": "First Crush",
  "scene": 0,
  "people": [
    {
      "name": "Milo",
      "x": -1.1,
      "z": 6.2,
      "look": 3
    },
    {
      "name": "Rosa",
      "x": 1.1,
      "z": 6.4,
      "look": 18
    }
  ],
  "missions": [
    {
      "a": 0,
      "mood": 0,
      "b": 1
    }
  ],
  "arrows": [
    0
  ],
  "budget": 3,
  "opt": 1,
  "states": 1,
  "breather": false,
  "hint": ""
}
```


Root keys: levels. Inspect MaData.cs, MaGen.cs, MaLevels.cs, MaSim.cs for field semantics and validation. Keep visible names/hints distinct from technical keys. The game-specific board, wave, card, opponent or question records are gameplay data; UI translation is separate.

CHAPTER 09

Reskin recipe: interface colours

Stop Play mode and open `Scripts/MaArt.cs`. Find `Ink`, currently `#3A1E3F`. Change that hex literal to `#42C9B0`, preserve its symbol/type, compile and reopen the scene. Review HUD, menus, helper cards, settings/results and both orientations. Art sprites and procedural fallback shapes can have additional colours; edit them deliberately.

Actual consumers of this symbol are listed first, then all current palette/hex assignments. Editor-only palettes do not control the runtime. Preserve state/goal contrast and rebuild after accepting a change.

Source	Exact current code
`MaArt.cs:61 / public static Color Shade(Color c, float k) => new Color(c.r * k, c.g * k, c.b * k, c.a);`	`img.AddOutline(Ink, 3);`
`MoodArrowGame.Hud.cs:187 / void MissionCard(float x, float y, float w, float h)`	`card.AddOutline(Ink, 4); card.AddShadow(new Color(0, 0, 0, 0.3f), 8);`
`MoodArrowGame.Hud.cs:222 / void RefreshMission()`	`var txt = HzUI.Label("What", _missionCard, MissionText(D, m), (int)(h * 0.17f), Ink);`
`MoodArrowGame.Hud.cs:239 / void Portrait(RectTransform parent, int i, float size, Vector2 pos)`	`ring.AddOutline(Ink, 3);`
`MoodArrowGame.Hud.cs:246 / void Portrait(RectTransform parent, int i, float size, Vector2 pos)`	`var eye = HzUI.Panel("Eye", ring.transform, Ink, 6); eye.raycastTarget = false;`
`MoodArrowGame.Hud.cs:249 / void Portrait(RectTransform parent, int i, float size, Vector2 pos)`	`var name = HzUI.Label("Name", ring.transform, D.people[i].name, (int)(size * 0.24f), Ink);`
`MoodArrowGame.Hud.cs:440 / HzPopup MaPopupFull(string title, string message, Color accent, Sprite art, string subtitle, int stars, (string label, Color color, Action action)[] buttons)`	`card.AddOutline(Ink, 4);`
`MoodArrowGame.Hud.cs:482 / HzPopup MaPopupFull(string title, string message, Color accent, Sprite art, string subtitle, int stars, (string label, Color color, Action action)[] buttons)`	`p.Message = HzUI.Label("Msg", card.transform, message, 40, Ink, TextAnchor.MiddleCenter, false);`
`MoodArrowGame.Hud.cs:586 / void ShowTitle()`	`t.AddOutline(Ink, 8); t.AddShadow(new Color(0, 0, 0, 0.45f), 12);`

MoodArrowGame.Hud.cs:598 / void ShowTitle()	tag.AddOutline(Ink, 4);
MoodArrowGame.Hud.cs:673 / void ShowLevels()	card.AddOutline(Ink, 4); card.AddShadow(new Color(0, 0, 0, 0.5f), 16);
MoodArrowGame.Hud.cs:757 / void ShowHintPerson(int i)	ic.AddOutline(Ink, 4); ic.AddShadow(new Color(0, 0, 0, 0.4f), 4);
MoodArrowGame.Hud.cs:804 / void RefreshTutorial()	hand.AddShadow(new Color(0, 0, 0, 0.5f), 5); hand.AddOutline(Ink, 3);
MoodArrowGame.Hud.cs:890 / void RefreshGuide()	hand.AddShadow(new Color(0, 0, 0, 0.5f), 4); hand.AddOutline(Ink, 2);
MoodArrowGame.World.cs:370 / void Fountain(Transform root, Vector3 p)	tag.raycastTarget = false; tag.AddOutline(new Color(Ink.r, Ink.g, Ink.b, 0.5f), 2);
MoodArrowGame.World.cs:372 / void Fountain(Transform root, Vector3 p)	v.tagText = HzUI.Label("N", tag.transform, d.name, 30, Ink); v.tagText.rectTransform.Stretch();

Source	Exact current code
MaArt.cs:10 / class/static declarations	public static readonly Color Ink = HzUI.Hex("#3A1E3F"), Violet = HzUI.Hex("#8A4FFF"), VioletDark = HzUI.Hex("#5B2DB8"), Cream = HzUI.Hex("#FFF8EC"),
MaArt.cs:11 / class/static declarations	Coral = HzUI.Hex("#FF6F7D"), Mint = HzUI.Hex("#3FCB8A"), Sun = HzUI.Hex("#FFC93C"), Sky = HzUI.Hex("#4FB3F2"), Sub = HzUI.Hex("#8C7A99"),
MaArt.cs:12 / class/static declarations	CardText = HzUI.Hex("#3A1E3F"), Locked = HzUI.Hex("#B9AFC4"), PopupCard = HzUI.Hex("#FFFDF7"), Leaf = HzUI.Hex("#2FBF71"), Orange = HzUI.Hex("#FF9A3C");
MaArt.cs:17 / class/static declarations	HzUI.Hex("#FF6F9C"), HzUI.Hex("#F0443A"), HzUI.Hex("#4FA3F0"), HzUI.Hex("#8CD43C"), HzUI.Hex("#A57CF0"), HzUI.Hex("#6B3FB8"), HzUI.Hex("#FF9A3C"),
MaArt.cs:24 / class/static declarations	public static readonly string[] SkinHex = { "#F6D2B8", "#E8B48E", "#C68863", "#8D5A3B" };

MaArt.cs:25 / class/static declarations	public static readonly string[] HairHex = { "#3B2A20", "#E0B04A", "#B5502C", "#2A2A33", "#7A4A2E", "#EDE3D0" };
MaArt.cs:26 / class/static declarations	public static readonly string[] ShirtHex = { "#FF6F7D", "#4FB3F2", "#FFC93C", "#3FCB8A", "#A57CF0", "#FF9A3C", "#F2F2F2", "#2F6FD1", "#E85FB0", "#5ED3C6" };
MaArt.cs:27 / class/static declarations	public static readonly string[] PantsHex = { "#2F3C66", "#4A3A30", "#2E2E38", "#5B6E8C", "#7A5236", "#3B5A44" };
MoodArrowGame.cs:30 / class/static declarations	public static readonly string[] BoostColor = { "#FFB52E", "#2EC4B6", "#B0703A", "#FF5FA2" };
MoodArrowGame.Hud.cs:360 / void RefreshHud()	_quiverTxt.color = _arrowsLeft <= 1 ? HzUI.Hex("#FFB0B0") : Color.white;
MoodArrowGame.Hud.cs:383 / void RefreshBoosters()	v.Ring.color = _guide == b ? HzUI.Hex("#FFE066") : new Color(1, 1, 1, 0);
MoodArrowGame.Hud.cs:391 / void RefreshCrosshair()	var c = on ? HzUI.Hex("#3BE07A") : Color.white;
MoodArrowGame.Hud.cs:584 / void ShowTitle()	var t = HzUI.Label("W" + w, logo, word[w], IsPortrait ? 190 : 170, w == 0 ? HzUI.Hex("#FF6F9C") : Sun);
MoodArrowGame.Hud.cs:620 / void ShowSettingsPopup()	("Levels", HzUI.Hex("#4FA3D9"), ShowLevels),
MoodArrowGame.Hud.cs:676 / void ShowLevels()	var ribbon = HzUI.Panel("Ribbon", card.transform, HzUI.Hex("#4FA3D9"), 44);
MoodArrowGame.Hud.cs:701 / void ShowLevels()	var si = HzIcons.Icon(rt, HzIcons.Star, 34, s < st ? HzUI.Hex("#FFF2A0") : new Color(0, 0, 0, 0.2f));
MoodArrowGame.World.cs:105 / void BuildWorld()	_cam.backgroundColor = HzUI.Hex("#9ED8FF");
MoodArrowGame.World.cs:112 / void BuildWorld()	RenderSettings.fogColor = HzUI.Hex("#BFE6FF");
MoodArrowGame.World.cs:114 / void BuildWorld()	RenderSettings.ambientSkyColor = HzUI.Hex("#BFD9F2"); RenderSettings.ambientEquatorColor = HzUI.Hex("#E8D9C4"); RenderSettings.ambientGroundColor = HzUI.Hex("#8C7A6A");

`MoodArrowGame.World.cs:118 / void BuildWorld()`	`sun.color = HzUI.Hex("#FFF1DC"); sun.intensity = 1.35f; sun.shadows = LightShadows.Soft; sun.shadowStrength = 0.6f;`
`MoodArrowGame.World.cs:122 / void BuildWorld()`	`fill.type = LightType.Directional; fill.color = HzUI.Hex("#BFD4FF"); fill.intensity = 0.35f; fill.shadows = LightShadows.None;`
`MoodArrowGame.World.cs:149 / void TunePost()`	`new Theme { ground = "#9AD27A", plaza = "#E9DCC2", accent = "#FF6F7D", walls = new[] { "#F6E3B4", "#F9C9A8", "#CDE7F0", "#F4D6E4", "#E6E0F7" }, props = 0 },`
`MoodArrowGame.World.cs:150 / void TunePost()`	`new Theme { ground = "#F2DDA4", plaza = "#D9B887", accent = "#4FB3F2", walls = new[] { "#FFFFFF", "#FFE3A3", "#BEE7F2", "#FFD0C2" }, props = 1 },`
`MoodArrowGame.World.cs:151 / void TunePost()`	`new Theme { ground = "#86CC6A", plaza = "#B9DE8E", accent = "#FFC93C", walls = new[] { "#E3F2D5", "#F7E7C6", "#D7EAF7" }, props = 2 },`
`MoodArrowGame.World.cs:152 / void TunePost()`	`new Theme { ground = "#9AD27A", plaza = "#D9C8B0", accent = "#2F6FD1", walls = new[] { "#F2B88C", "#F7D9A6", "#E8C3A0" }, props = 3 },`
`MoodArrowGame.World.cs:153 / void TunePost()`	`new Theme { ground = "#8FD08A", plaza = "#EEE6DA", accent = "#A57CF0", walls = new[] { "#F7F3EC", "#EDE3F5", "#F4E9D6" }, props = 4 },`
`MoodArrowGame.World.cs:154 / void TunePost()`	`new Theme { ground = "#C9C2B5", plaza = "#E7D7BF", accent = "#FF9A3C", walls = new[] { "#F9D8A8", "#F5C1C1", "#C8E6C9", "#C9DDF6", "#FBE7B0" }, props = 5 },`
`MoodArrowGame.World.cs:180 / void BuildScene(int scene)`	`Part("Sea", root, MaMeshes.Plane(160, 60), HzUI.Hex("#3FA9E0"), new Vector3(0, 0.02f, 42), default, 0.85f, false);`
`MoodArrowGame.World.cs:181 / void BuildScene(int scene)`	`Part("Foam", root, MaMeshes.Box(new Vector3(160, 0.05f, 0.6f), 0.02f, 1), HzUI.Hex("#E8F7FF"), new Vector3(0, 0.03f, 12.3f), default, 0.3f, false);`
`MoodArrowGame.World.cs:182 / void BuildScene(int scene)`	`for (int i = 0; i < 5; i++) Hut(root, new Vector3(-11 + i * 5.5f + rng.Range(-0.6f, 0.6f), 0, 15.5f + rng.Range(-0.5f, 1.5f)), HzUI.Hex(th.walls[i % th.walls.Length]), HzUI.Hex(i % 2 == 0 ? th.accent : "#FF6F7D"));`

MoodArrowGame.World.cs:184 / void BuildScene(int scene)	for (int i = 0; i < 3; i++) Umbrella(root, new Vector3(rng.Range(-6f, 6f), 0, 11.6f + rng.Range(-0.3f, 0.6f)), HzUI.Hex(i == 1 ? "#4FB3F2" : "#FF6F7D"));
MoodArrowGame.World.cs:189 / void BuildScene(int scene)	Part("Road", root, MaMeshes.Box(new Vector3(70, 0.03f, 3.2f), 0.01f, 1), HzUI.Hex("#6E6E7A"), new Vector3(0, 0.01f, 12.9f), default, 0.2f, false);
MoodArrowGame.World.cs:190 / void BuildScene(int scene)	for (int i = -4; i <= 4; i++) Part("Lane", root, MaMeshes.Box(new Vector3(1.6f, 0.035f, 0.14f), 0.02f, 1), HzUI.Hex("#F4F1E6"), new Vector3(i * 3.6f, 0.012f, 12.9f), default, 0.2f, false);
MoodArrowGame.World.cs:191 / void BuildScene(int scene)	Part("Walk", root, MaMeshes.Box(new Vector3(70, 0.14f, 2.0f), 0.04f, 1), HzUI.Hex("#D8D2C6"), new Vector3(0, 0.06f, 15.5f), default, 0.1f, false);
MoodArrowGame.World.cs:205 / void BuildScene(int scene)	else if (th.props == 5) Stall(root, new Vector3(sx, 0, sz), HzUI.Hex(i % 4 == 1 ? th.accent : "#FF6F7D"));
MoodArrowGame.World.cs:211 / void BuildScene(int scene)	if (th.props == 3) { Part("Hoop", root, MaMeshes.Cylinder(0.06f, 0.06f, 3f, 8), HzUI.Hex("#7A7A86"), new Vector3(7.2f, 0, 9.5f)); Part("Board", root, MaMeshes.Box(new Vector3(1.2f, 0.8f, 0.08f), 0.03f, 1), Color.white, new Vector3(7.2f, 3.1f, 9.5f)); }
MoodArrowGame.World.cs:217 / void BuildScene(int scene)	Part("Hill", root, MaMeshes.Sphere(1, 18), HzUI.Hex(i % 2 == 0 ? "#7CC46A" : "#8ED27C"), new Vector3(-48 + i * 16 + rng.Range(-3f, 3f), -s * 0.55f, 48 + rng.Range(0f, 8f)), new Vector3(s * 1.6f, s, s), 0.05f, false);
MoodArrowGame.World.cs:226 / void Building(Transform root, Vector3 p, float w, float h, Color wall, Color accent, int theme, int idx)	Part("Roof", b, MaMeshes.Box(new Vector3(w + 0.4f, 0.35f, 3.6f), 0.1f, 1), theme == 4 ? HzUI.Hex("#8C5A9E") : Shade(accent, 0.85f), new Vector3(0, h + 0.15f, 0), default, 0.2f);
MoodArrowGame.World.cs:227 / void Building(Transform root, Vector3 p, float w, float h, Color wall, Color accent, int theme, int idx)	var glass = HzUI.Hex("#3E5A7A");

`MoodArrowGame.World.cs:238 / void Building(Transform root, Vector3 p, float w, float h, Color wall, Color accent, int theme, int idx)`	`Part("Door", b, MaMeshes.Box(new Vector3(1.1f, 1.9f, 0.14f), 0.06f, 1), theme == 3 ? HzUI.Hex("#2F6FD1") : Shade(accent, 0.75f), new Vector3(-w / 2 + (cols / 2 + 0.5f) * (w / cols), 0.95f, -1.64f), default, 0.3f, false);`
`MoodArrowGame.World.cs:245 / void Tree(Transform root, Vector3 p, float s)`	`Part("Trunk", t, MaMeshes.Cylinder(0.16f, 0.12f, 1.6f, 10), HzUI.Hex("#8A5A3A"), Vector3.zero);`
`MoodArrowGame.World.cs:246 / void Tree(Transform root, Vector3 p, float s)`	`Part("Leaves", t, MaMeshes.Sphere(0.95f, 14), HzUI.Hex("#4FB55A"), new Vector3(0, 2.1f, 0), new Vector3(1, 0.9f, 1), 0.15f);`
`MoodArrowGame.World.cs:247 / void Tree(Transform root, Vector3 p, float s)`	`Part("Leaves2", t, MaMeshes.Sphere(0.7f, 12), HzUI.Hex("#62C76A"), new Vector3(0.35f, 2.75f, -0.1f), default, 0.15f);`
`MoodArrowGame.World.cs:253 / void Palm(Transform root, Vector3 p)`	`for (int i = 0; i < 6; i++) Part("Seg", t, MaMeshes.Cylinder(0.15f, 0.13f, 0.55f, 8), HzUI.Hex(i % 2 == 0 ? "#B07A4A" : "#9C6A3E"), new Vector3(i * 0.06f, i * 0.52f, 0));`
`MoodArrowGame.World.cs:256 / void Palm(Transform root, Vector3 p)`	`var leaf = Part("Leaf", t, MaMeshes.Box(new Vector3(0.35f, 0.06f, 1.6f), 0.03f, 1), HzUI.Hex("#3FAE5A"), new Vector3(0.36f, 3.2f, 0), default, 0.2f);`
`MoodArrowGame.World.cs:282 / void Lamp(Transform root, Vector3 p)`	`Part("Pole", t, MaMeshes.Cylinder(0.07f, 0.05f, 3f, 8), HzUI.Hex("#3A3F55"), Vector3.zero, default, 0.6f);`
`MoodArrowGame.World.cs:283 / void Lamp(Transform root, Vector3 p)`	`Part("Bulb", t, MaMeshes.Sphere(0.22f, 10), HzUI.Hex("#FFF2B8"), new Vector3(0, 3.1f, 0), default, 0.8f);`
`MoodArrowGame.World.cs:291 / void Bench(Transform root, Vector3 p, Color c)`	`Part("LegL", t, MaMeshes.Box(new Vector3(0.08f, 0.45f, 0.45f), 0.02f, 1), HzUI.Hex("#3A3F55"), new Vector3(-0.75f, 0.22f, 0));`
`MoodArrowGame.World.cs:292 / void Bench(Transform root, Vector3 p, Color c)`	`Part("LegR", t, MaMeshes.Box(new Vector3(0.08f, 0.45f, 0.45f), 0.02f, 1), HzUI.Hex("#3A3F55"), new Vector3(0.75f, 0.22f, 0));`
`MoodArrowGame.World.cs:298 / void Stall(Transform root, Vector3 p, Color c)`	`Part("Table", t, MaMeshes.Box(new Vector3(1.6f, 0.9f, 0.9f), 0.05f, 1), HzUI.Hex("#C89A6A"), new Vector3(0, 0.45f, 0));`

`MoodArrowGame.World.cs:302 / void Stall(Transform root, Vector3 p, Color c)`	`for (int i = 0; i < 4; i++) Part("Fruit", t, MaMeshes.Sphere(0.14f, 8), HzUI.Hex(i % 2 == 0 ? "#FF6F3C" : "#FFD23C"), new Vector3(-0.5f + i * 0.33f, 1.0f, -0.1f), default, 0.5f);`
`MoodArrowGame.World.cs:308 / void Fountain(Transform root, Vector3 p)`	`Part("Basin", t, MaMeshes.Cylinder(1.3f, 1.4f, 0.5f, 20), HzUI.Hex("#D8D2C6"), Vector3.zero, default, 0.2f);`
`MoodArrowGame.World.cs:309 / void Fountain(Transform root, Vector3 p)`	`Part("Water", t, MaMeshes.Cylinder(1.15f, 1.15f, 0.04f, 20), HzUI.Hex("#5CC3F0"), new Vector3(0, 0.44f, 0), default, 0.9f, false);`
`MoodArrowGame.World.cs:310 / void Fountain(Transform root, Vector3 p)`	`Part("Column", t, MaMeshes.Cylinder(0.18f, 0.14f, 1.3f, 10), HzUI.Hex("#E8E2D6"), new Vector3(0, 0.4f, 0), default, 0.2f);`
`MoodArrowGame.World.cs:311 / void Fountain(Transform root, Vector3 p)`	`Part("Bowl", t, MaMeshes.Cylinder(0.25f, 0.55f, 0.2f, 14), HzUI.Hex("#E8E2D6"), new Vector3(0, 1.6f, 0), default, 0.2f);`
`MoodArrowGame.World.cs:328 / void Fountain(Transform root, Vector3 p)`	`Color skin = Skin(look), hair = Hair(look), shirt = Shirt(look), pants = Pants(look), shoe = HzUI.Hex("#2E2A33");`
`MoodArrowGame.World.cs:356 / void Fountain(Transform root, Vector3 p)`	`var eye = HzUI.Hex("#2A1E2E");`
`MoodArrowGame.World.cs:359 / void Fountain(Transform root, Vector3 p)`	`Part("Cheek", v.head, MaMeshes.Box(new Vector3(0.12f, 0.03f, 0.03f), 0.012f, 1), HzUI.Hex("#C0475A"), new Vector3(0, 0.17f, 0.245f), default, 0.4f, false);`
`MoodArrowGame.World.cs:363 / void Fountain(Transform root, Vector3 p)`	`var h = Part("Horn", v.horns, MaMeshes.Cylinder(0.06f, 0.0f, 0.2f, 8), HzUI.Hex("#E23A3A"), new Vector3(s * 0.13f, -0.04f, 0.02f), default, 0.6f);`
`MoodArrowGame.World.cs:611 / void BuildBow()`	`Part("Limb", _bow, MaMeshes.BowLimb(), HzUI.Hex("#E08A3A"), Vector3.zero, default, 0.55f, false);`
`MoodArrowGame.World.cs:612 / void BuildBow()`	`Part("Grip", _bow, MaMeshes.Cylinder(0.062f, 0.062f, 0.26f, 12), HzUI.Hex("#E2384B"), new Vector3(0, -0.13f, 0), default, 0.4f, false);`
`MoodArrowGame.World.cs:616 / void BuildBow()`	`_stringA = Part("StringA", _bow, MaMeshes.Box(new Vector3(0.008f, 1f, 0.008f), 0.002f, 1), HzUI.Hex("#FFF8EC"), Vector3.zero, default, 0.2f, false);`

`MoodArrowGame.World.cs:617 / void BuildBow()`	`_stringB = Part("StringB", _bow, MaMeshes.Box(new Vector3(0.008f, 1f, 0.008f), 0.002f, 1), HzUI.Hex("#FFF8EC"), Vector3.zero, default, 0.2f, false);`
`MoodArrowGame.World.cs:628 / void BuildBow()`	`Part("Shaft", a, MaMeshes.Box(new Vector3(0.022f, 0.022f, 0.85f), 0.008f, 1), HzUI.Hex("#F2A65A"), new Vector3(0, 0, 0.425f), default, 0.3f, false);`
`MoodArrowGame.World.cs:629 / void BuildBow()`	`Part("FletchA", a, MaMeshes.Box(new Vector3(0.006f, 0.07f, 0.14f), 0.003f, 1), HzUI.Hex("#FF6F7D"), new Vector3(0, 0.03f, 0.06f), default, 0.3f, false);`
`MoodArrowGame.World.cs:630 / void BuildBow()`	`Part("FletchB", a, MaMeshes.Box(new Vector3(0.07f, 0.006f, 0.14f), 0.003f, 1), HzUI.Hex("#FF6F7D"), new Vector3(0, 0, 0.06f), default, 0.3f, false);`
`MoodArrowGame.World.cs:725 / void Burst(Vector3 at, int mood, int count, float speed = 1.6f, bool gravity = false, float size = 64)`	`if (mood == (int)MaMood.Sick || mood == (int)MaMood.Sad) { f.img.sprite = HzUI.Circle(); f.img.color = mood == (int)MaMood.Sick ? HzUI.Hex("#9BE04A") : HzUI.Hex("#7CC6FF"); f.size *= 0.45f; }`
`MoodArrowGame.World.cs:754 / void Stream(Vector3 a, Vector3 b, int count)`	`f.img.sprite = HzUI.Circle(); f.img.color = HzUI.Hex(n % 2 == 0 ? "#9BE04A" : "#C9F07A");`

CHAPTER 10

Reskin recipe: visible menu and HUD text

Open `Scripts/MoodArrowGame.Hud.cs`, find the displayed literal `"Settings"`, and replace its displayed value with `"Options"`. Keep hierarchy names and technical IDs unchanged. In HzUI.Label/Button calls the first string may be an object name; change the displayed text argument. Read the method references below to distinguish construction from later refresh assignments.

Preserve interpolation braces, invariant number formatting, ON/OFF updates and line breaks. Update both initial and refresh paths, portrait and landscape. Exact names/unlocks/free charges: Scripts/MoodArrowGame.cs; effect/help strings: Scripts/MoodArrowGame.Hud.cs. Level names/hints remain separate JSON content. If a title appears in art, Title property or another screen, update those visible uses deliberately without renaming GameId, namespace, resources or save keys.

Hard-coded game strings do not automatically use shared localization. To translate the interface, route every display assignment through your translation table. Dictionaries, word banks and theme pools are independent gameplay assets and need separate validation. Test help, settings, results, tutorial and helper introductions after a fresh launch.

Source	Exact current code
`MaData.cs:119 / public bool Matches(MaMission m) => a == m.a && mood == m.mood && (m.b < 0 || b == m.b);`	`public static readonly string[] MoodName = { "Love", "Angry", "Sad", "Sick", "Silly", "Spooky", "Laugh" };`
`MoodArrowGame.cs:27 / class/static declarations`	`public static readonly string[] BoostName = { "Aid", "Rewind", "Quiver", "Skip" };`
`MoodArrowGame.cs:213 / void OnArrowArrive(int hit, int mood, Vector3 point)`	`Toast(_arrowsLeft > 0 ? "Missed! Aim the crosshair at a person" : "Missed!", null, 1.1f, 0.3f);`
`MoodArrowGame.cs:318 / IEnumerator PlayReaction(MaReaction r)`	`else Toast("Nobody is close enough to laugh", null, 1.2f, 0.3f);`
`MoodArrowGame.cs:329 / IEnumerator PlayReaction(MaReaction r)`	`if (r.crowd.Count == 0) Toast("Nobody is close enough to be spooked", null, 1.2f, 0.3f);`
`MoodArrowGame.cs:346 / IEnumerator PlayReaction(MaReaction r)`	`Toast("Part " + (r.partsAfter + 1).ToString(CultureInfo.InvariantCulture) + " of " + D.missions.Length.ToString(CultureInfo.InvariantCulture) + "!", Sun, 1.2f, 0.3f);`
`MoodArrowGame.cs:403 / void OutOfArrows()`	`MaPopup("Out of arrows!", "The mission isn't done yet.\nGet 2 more arrows or try again.", Coral, Get("booster_quiver"), D.name, buttons.ToArray());`

MoodArrowGame.cs:414 / void OnBooster(int b)	Toast($"{BoostName[b]} unlocks at level {BoostUnlock[b]}", null, 1.3f, 0.3f);
MoodArrowGame.cs:422 / void OnBooster(int b)	MaPopup("More " + BoostName[b] + "?", "Watch a short ad to get\n1 more " + BoostName[b] + ".", HzUI.Hex(BoostColor[b]), Get(BoostArt[b]), null,
MoodArrowGame.cs:423 / void OnBooster(int b)	("Watch Ad", Leaf, () => HzPlatform.Rewarded(GameId, "booster_" + BoostName[b].ToLowerInvariant(), ok =>
MoodArrowGame.cs:438 / void OnBooster(int b)	Toast("+2 arrows!", Sun, 1.1f, 0.3f);
MoodArrowGame.cs:463 / void UseAid(bool free)	Toast(Unlocked(BRewind) ? "No way from here: try Rewind or Restart" : "No way from here: try Restart", null, 1.6f, 0.3f);
MoodArrowGame.cs:472 / void UseAid(bool free)	Toast($"Shoot {D.people[s.person].name} with {MaNames.MoodName[s.mood]}! ({sol.Moves} arrow{(sol.Moves == 1 ? "" : "s")} to go)", Sun, 1.8f, 0.3f);
MoodArrowGame.cs:477 / void UseRewind(bool free)	if (_history.Count == 0) { HzAudio.Play(Sfx.Invalid, 0.5f); Toast("Nothing to rewind yet", null, 1.1f, 0.3f); return; }
MoodArrowGame.cs:497 / void SkipLevel()	Toast("Level skipped", Sun, 1.0f, 0.3f);
MoodArrowGame.cs:506 / void SelectMood(int m, bool sound = true)	Toast($"{MaNames.MoodName[m]} arrow unlocks at level {MaNames.MoodUnlock[m]}", null, 1.3f, 0.3f);
MoodArrowGame.Hud.cs:142 / void LevelPill(float x, float y, float w, float h)	_levelTxt = HzUI.Label("Level", pill.transform, "", (int)(h * 0.48f), Sun); _levelTxt.rectTransform.Place(new Vector2(0.5f, 1), new Vector2(w - 20, h * 0.58f), new Vector2(0, -h * 0.33f));
MoodArrowGame.Hud.cs:144 / void LevelPill(float x, float y, float w, float h)	_nameTxt = HzUI.Label("Name", pill.transform, "", (int)(h * 0.24f), Cream, TextAnchor.MiddleCenter, false); _nameTxt.fontStyle = FontStyle.Bold;
MoodArrowGame.Hud.cs:163 / void QuiverCard(float x, float y, float w, float h)	_quiverTxt = HzUI.Label("N", card.transform, "", (int)(h * 0.5f), Color.white, TextAnchor.MiddleLeft);

`MoodArrowGame.Hud.cs:180 / void StarsCard(float x, float y, float w, float h)`	`_targetTxt = HzUI.Label("T", card.transform, "", (int)(h * 0.22f), Cream, TextAnchor.MiddleCenter, false); _targetTxt.fontStyle = FontStyle.Bold;`
`MoodArrowGame.Hud.cs:222 / void RefreshMission()`	`var txt = HzUI.Label("What", _missionCard, MissionText(D, m), (int)(h * 0.17f), Ink);`
`MoodArrowGame.Hud.cs:228 / void RefreshMission()`	`var pt = HzUI.Label("T", pill.transform, $"Part {part + 1}/{D.missions.Length}", 28, Color.white); pt.rectTransform.Stretch();`
`MoodArrowGame.Hud.cs:230 / void RefreshMission()`	`var mis = HzUI.Label("Mission", _missionCard, "MISSION", 26, Sub); mis.rectTransform.Place(new Vector2(1, 1), new Vector2(170, 44), new Vector2(-92, -8));`
`MoodArrowGame.Hud.cs:249 / void Portrait(RectTransform parent, int i, float size, Vector2 pos)`	`var name = HzUI.Label("Name", ring.transform, D.people[i].name, (int)(size * 0.24f), Ink);`
`MoodArrowGame.Hud.cs:274 / void ArrowBar(float cx, float cy, float size, float step)`	`var ka = HzUI.Label("KeyA", _layer, "A", 34, new Color(1, 1, 1, 0.55f)); At(ka.rectTransform, x0 - step * 0.5f - 50, cy, 60, 60);`
`MoodArrowGame.Hud.cs:275 / void ArrowBar(float cx, float cy, float size, float step)`	`var kd = HzUI.Label("KeyD", _layer, "D", 34, new Color(1, 1, 1, 0.55f)); At(kd.rectTransform, x0 + step * (n - 0.5f) + 50, cy, 60, 60);`
`MoodArrowGame.Hud.cs:280 / void ArrowBar(float cx, float cy, float size, float step)`	`var btn = HzUI.Button("Arrow" + m, _layer, null, Color.white, Color.white, () => SelectMood(mm), 30, (int)(size * 0.3f));`
`MoodArrowGame.Hud.cs:296 / void ArrowBar(float cx, float cy, float size, float step)`	`var lv = HzUI.Label("Lv", lk.transform, "Lv " + MaNames.MoodUnlock[m].ToString(CultureInfo.InvariantCulture), (int)(size * 0.2f), Color.white);`
`MoodArrowGame.Hud.cs:300 / void ArrowBar(float cx, float cy, float size, float step)`	`var cap = HzUI.Label("Cap", rt, MaNames.MoodName[m], (int)(size * 0.2f), Color.white);`
`MoodArrowGame.Hud.cs:323 / void RefreshArrowBar()`	`var btn = HzUI.Button("Booster" + b, parent, null, Color.white, Color.white, () => OnBooster(bb), 40, (int)(size * 0.32f));`
`MoodArrowGame.Hud.cs:344 / void RefreshArrowBar()`	`v.BadgeText = HzUI.Label("N", bd.transform, "", (int)(size * 0.24f), Color.white); v.BadgeText.rectTransform.Stretch();`

`MoodArrowGame.Hud.cs:346 / void RefreshArrowBar()`	`v.Caption = HzUI.Label("Cap", v.Root, "", (int)(size * 0.22f), Color.white);`
`MoodArrowGame.Hud.cs:357 / void RefreshHud()`	`_levelTxt.text = "LEVEL " + _playLevel.ToString(inv);`
`MoodArrowGame.Hud.cs:358 / void RefreshHud()`	`_nameTxt.text = D.name + (D.breather ? " (breather)" : "");`
`MoodArrowGame.Hud.cs:359 / void RefreshHud()`	`_quiverTxt.text = "x " + _arrowsLeft.ToString(inv);`
`MoodArrowGame.Hud.cs:362 / void RefreshHud()`	`_targetTxt.text = $"3 stars:\n{D.opt} arrow{(D.opt == 1 ? "" : "s")}";`
`MoodArrowGame.Hud.cs:380 / void RefreshBoosters()`	`v.BadgeText.text = _guide == b ? "1" : c > 0 ? c.ToString(CultureInfo.InvariantCulture) : "+";`
`MoodArrowGame.Hud.cs:382 / void RefreshBoosters()`	`v.Caption.text = un ? BoostName[b] : "Level " + BoostUnlock[b].ToString(CultureInfo.InvariantCulture);`
`MoodArrowGame.Hud.cs:472 / HzPopup MaPopupFull(string title, string message, Color accent, Sprite art, string subtitle, int stars, (string label, Color color, Action action)[] buttons)`	`var t = HzUI.Label("Title", ribbon.transform, title, 66, Color.white); t.rectTransform.Stretch(6);`
`MoodArrowGame.Hud.cs:477 / HzPopup MaPopupFull(string title, string message, Color accent, Sprite art, string subtitle, int stars, (string label, Color color, Action action)[] buttons)`	`var st = HzUI.Label("Sub", card.transform, subtitle, 40, Violet); st.rectTransform.Place(new Vector2(0.5f, 1), new Vector2(cw - 80, 50), new Vector2(0, y - 25));`
`MoodArrowGame.Hud.cs:482 / HzPopup MaPopupFull(string title, string message, Color accent, Sprite art, string subtitle, int stars, (string label, Color color, Action action)[] buttons)`	`p.Message = HzUI.Label("Msg", card.transform, message, 40, Ink, TextAnchor.MiddleCenter, false);`
`MoodArrowGame.Hud.cs:535 / void UnlockPopup(int b, Action done)`	`MaPopup("New: " + BoostName[b] + "!", BoostDesc[b] + "\nYou get " + BoostStart[b].ToString(CultureInfo.InvariantCulture) + " free!", HzUI.Hex(BoostColor[b]), Get(BoostArt[b]), null,`
`MoodArrowGame.Hud.cs:554 / void ArrowUnlockPopup(int m, Action done)`	`MaPopup("New arrow: " + MaNames.MoodName[m] + "!", ArrowDesc[m], MoodColor[m], Get(MoodArt[m]), null, ("Cool!", Leaf, () => { SelectMood(m, false); done?.Invoke(); }));`

`MoodArrowGame.Hud.cs:566 / void ShowLevelCard(bool tutorial)`	`var p = MaPopup("Level " + _playLevel.ToString(CultureInfo.InvariantCulture), msg, Sun, art, D.name, ("Play", Leaf, (Action)Begin));`
`MoodArrowGame.Hud.cs:584 / void ShowTitle()`	`var t = HzUI.Label("W" + w, logo, word[w], IsPortrait ? 190 : 170, w == 0 ? HzUI.Hex("#FF6F9C") : Sun);`
`MoodArrowGame.Hud.cs:596 / void ShowTitle()`	`var tag = HzUI.Label("Tag", dim.transform, "Shoot feelings. Make people act on them!", IsPortrait ? 44 : 42, Color.white);`
`MoodArrowGame.Hud.cs:599 / void ShowTitle()`	`var play = HzUI.Button("Play", dim.transform, "Play", Leaf, Color.white, () =>`
`MoodArrowGame.Hud.cs:616 / void ShowSettingsPopup()`	`var p = MaPopup("Settings", "", Violet, null, null,`
`MoodArrowGame.Hud.cs:619 / void ShowSettingsPopup()`	`("How to Play", Orange, () => HelpPage(0)),`
`MoodArrowGame.Hud.cs:631 / static string SoundLabel() => HzAudio.Muted ? "Sound: OFF" : "Sound: ON";`	`static string SoundLabel() => HzAudio.Muted ? "Sound: OFF" : "Sound: ON";`
`MoodArrowGame.Hud.cs:632 / static string MusicLabel() => HzAudio.MusicMuted ? "Music: OFF" : "Music: ON";`	`static string MusicLabel() => HzAudio.MusicMuted ? "Music: OFF" : "Music: ON";`
`MoodArrowGame.Hud.cs:647 / void HelpPage(int i)`	`MaPopup($"How to Play {i + 1}/{HelpPages.Length}", HelpPages[i], Sky, art, null,`
`MoodArrowGame.Hud.cs:648 / void HelpPage(int i)`	`(last ? "Got it" : "Next", Leaf, last ? (Action)null : () => HelpPage(i + 1)));`
`MoodArrowGame.Hud.cs:678 / void ShowLevels()`	`var t = HzUI.Label("Title", ribbon.transform, "Levels", 66, Color.white); t.rectTransform.Stretch(6); t.AddOutline(new Color(0, 0, 0, 0.25f), 3);`
`MoodArrowGame.Hud.cs:688 / void ShowLevels()`	`var b = HzUI.Button("L" + lv, card.transform, null, col, Color.white, () =>`
`MoodArrowGame.Hud.cs:697 / void ShowLevels()`	`var n = HzUI.Label("N", rt, lv.ToString(CultureInfo.InvariantCulture), 54, Color.white); n.rectTransform.Place(new Vector2(0.5f, 0.5f), new Vector2(cell, 70), new Vector2(0, 14));`

`MoodArrowGame.Hud.cs:710 / void ShowLevels()`	`var prev = HzUI.Button("Prev", card.transform, "", Violet, Color.white, () => { p.Close(); _levelsPage = page - 1; HzAudio.Play(Sfx.Page, 0.5f); ShowLevels(); }, 40, 36);`
`MoodArrowGame.Hud.cs:716 / void ShowLevels()`	`var next = HzUI.Button("NextPage", card.transform, "", Violet, Color.white, () => { p.Close(); _levelsPage = page + 1; HzAudio.Play(Sfx.Page, 0.5f); ShowLevels(); }, 40, 36);`
`MoodArrowGame.Hud.cs:720 / void ShowLevels()`	`var close = HzUI.Button("Close", card.transform, "Close", Leaf, Color.white, () => p.Close(), 50, 36);`
`MoodArrowGame.Hud.cs:740 / void ShowTip(string text)`	`_tipText = HzUI.Label("T", tip.transform, text, IsPortrait ? 36 : 34, Cream, TextAnchor.MiddleCenter, false);`
`MoodArrowGame.Hud.cs:914 / void CompleteGuide(int b)`	`Toast("Nice! " + BoostName[b] + " is ready when you need it", Sun, 1.3f, 0.3f);`
`MoodArrowGame.World.cs:372 / void Fountain(Transform root, Vector3 p)`	`v.tagText = HzUI.Label("N", tag.transform, d.name, 30, Ink); v.tagText.rectTransform.Stretch();`

CHAPTER 11

Artwork, fonts, audio and licences

Replace game images under Resources/moodarrow through Unity while retaining resource keys, .meta GUIDs, Sprite/UI import settings, alpha, pivots and slicing. Procedural graphics in the art/view sources may need separate colour changes. Replacing one PNG does not change every fallback. Keep third-party font and other licence notices with the corresponding shipped assets.

Effects are under Resources/moodarrow/Sfx. Preserve each actual load key and importer settings when replacing sounds. Supplied music: Resources/moodarrow/Music/town_loop.ogg. Preserve that load key and verify loop/compression and the independent Music toggle. Kit UI uses supplied fonts; preserve actual licence notices and verify new glyphs.

CHAPTER 12

Local saves, reset and migration

HzSave wraps PlayerPrefs and flushes local state on focus loss/quit. WebGL persistence uses browser storage tied to the hosting origin. Clearing site data or changing origin starts a new local save; this is not cloud synchronization. Inspect HzSave/PlayerPrefs calls in the actual game source before altering progression, stars, helper charges or tutorial flags.

Verify fresh launch, a completed arena, helper use, sound options, focus loss and restart. Use a dedicated test profile for resets. Keep technical save keys stable during colour/text reskins. If changing schema or keys, implement and test a migration rather than silently discarding existing users' progress.

CHAPTER 13

Advertising integration and configuration

Asset: `Assets/Szekipapa77Ads/Resources/AdsSettings.asset`. Inspect backend, useGoogleTestIds and missingSdkFallback in the publisher-owned Ads configuration. Buyer projects include adapter/integration code and exclude external provider SDK binaries. WebGL and a missing-provider fallback simulate ads; these are not live commercial impressions.

For real Android ads install the supported provider SDK from its official source into your working project, enter your own app/placement IDs and use test ads first. Verify banner/interstitial/rewarded callbacks on supported hardware. For production use your own IDs and disable simulated missing-SDK fallback. Build tooling rejects unsafe production fallback settings. No IAP, remote backend or HLK is included in this publisher edition.

CHAPTER 14

Android build and release

Install Unity 6000.0.62f1 Android modules through Hub. Select Android, run Setup and review application ID `com.szekipapa77.moodarrow`, version, scene orientation and signing. The build command writes `Builds/Android/moodarrow.apk`. Confirm BuildResult.Succeeded and the actual output file; intermediate report totals are not the final APK size.

Use your own keystore/signing and distribution settings. A test-signed demo is not a signed store release. A provider-free build simulates ads; a provider test-demo does not prove production ad-account configuration. Test real hardware touch, safe areas, focus/audio, save/relaunch and reward callbacks. Keep passwords, keystores and private service credentials out of buyer packages.

CHAPTER 15

WebGL build and hosting

Install matching WebGL support. Run Setup and Build WebGL; output is `Builds/WebGL/moodarrow/`. Serve index.html with all loader/framework/data/WASM/StreamingAssets files over HTTP(S), not file://. Match compression, Content-Encoding and MIME types to the host. Outer page sizing and actual scene orientation are different settings.

The publisher's playable demo is hosted on HyperCodeStore. Upload the actual runtime files rather than merely changing a URL. Test a fresh anonymous load, real pointer controls, audio after interaction, saved progress/reload and phone/desktop sizing. Rebuild and rehost when code/assets change. A successful HTTP page alone is not gameplay verification.

CHAPTER 16

Troubleshooting

Compile errors: restore matching packages and the full private Kit/Ads layer; only one game Kit belongs in a project. **Pink/invisible rendering:** run Setup and inspect Graphics and every Quality URP assignment. **No input:** classic input handling must be enabled; check canvas focus and blocking modals. **Missing resources:** restore exact paths, .meta GUIDs and importer settings.

Edits do not appear: stop Play, Save in the level editor, reload the canonical bank and check current saved level. **Text reverts:** update construction and refresh assignments from the exact table. **No live ads:** provider binaries are deliberately excluded; inspect your installed provider and fallback mode. **Scene reload exception:** retain owner-lifetime and generation guards on delayed callbacks. **WebGL fails:** inspect browser Console, compression headers and missing payloads.

CHAPTER 17

Support and final release checklist

Publisher: Szekipapa77. Support: szekipapa77@gmail.com. Include game title, exact Unity version, platform, first Console error and reproduction steps. Never send passwords/tokens.

1. Clean-import the standalone project and native package.
2. Play both scenes and replace a scene near startup.
3. Review tutorial, normal completion/retry and every helper activation plus effect.
4. Validate editor changes and test reskin text/colours in both layouts.
5. Rebuild affected Android/WebGL outputs.
6. Verify complete public APK bytes, actual browser demo, documentation and video links.
7. Keep licences and exclude all internal prompts, planning/capture/build/QA tooling from buyers.