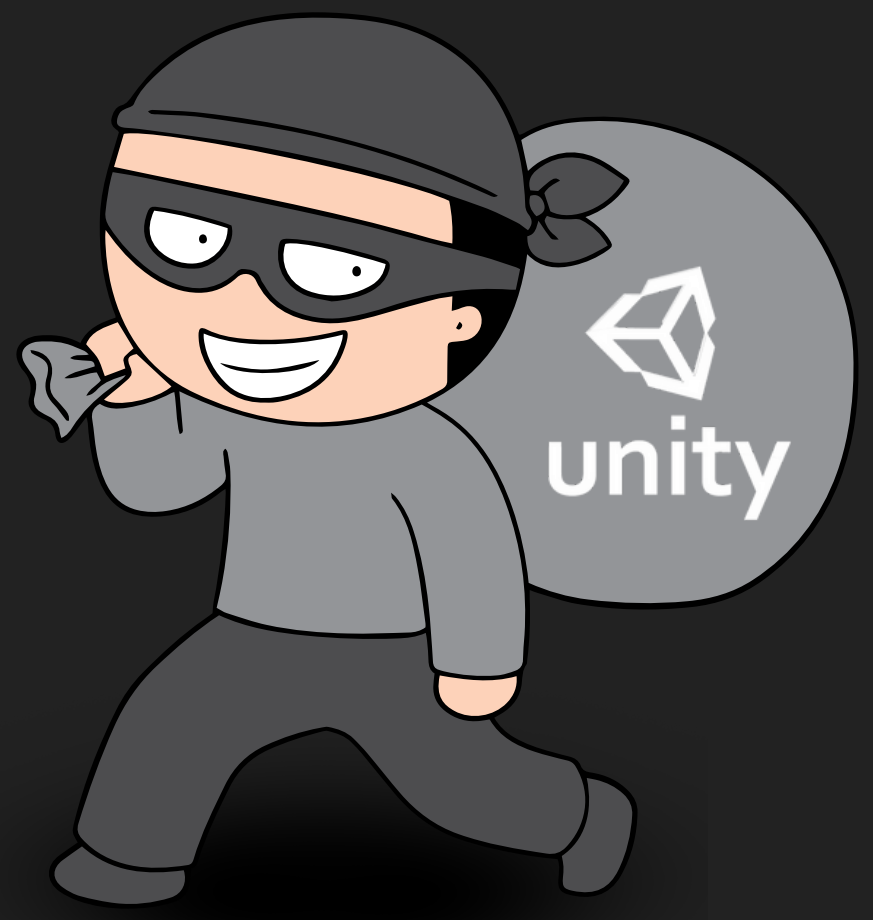
50

UNITY CHEAT-SHEETS



1. EVENT FUNCTIONS
2. COMPONENT COMMUNICATION
3. DATA STRUCTURES
4. INSPECTOR ATTRIBUTES
5. DEBUGGING
6. NEW INPUT SYSTEM: 1
7. NEW INPUT SYSTEM: 2
8. RENDERING PIPELINE
9. ANIMATIONS
10. AI: NAVMESH AGENT
11. UI
12. COLLISIONS
13. CAMERA
14. PHYSICS RIGIDBODY
15. SHADERS AND MATERIALS
16. PARTICLE SYSTEM
17. TIMELINE AND CINEMACHINE
18. GAME OBJECTS
19. TERRAIN SYSTEM
20. AUDIO SYSTEM
21. MULTIPLAYER NETCODE
22. PHYSICS - Part 2
23. UI - Part 2
24. ANIMATIONS - Part 2
25. VECTOR3 OPERATIONS
26. PARTICLE SYSTEM - PART 2
27. ASSET MANAGEMENT
28. PROFILER OPTIMIZATION
29. POST PROCESSING
30. EDITOR SCRIPTING
31. LOCALIZATION SYSTEM
32. MULTISCENE EDITING
33. TILEMAPS
34. VFX GRAPH
35. POST PROCESSING PART 2
36. NETWORK OPTIMIZATION
37. BUILD PIPELINE
38. AI NAVIGATION - PART 2
39. REWARDED VIDEO ADS
40. MATHEMATICS
41. TAGS & LAYERS
42. PHYSICS PART 3
43. PROJECT SETTINGS
44. PROBUILDER: MESH MODELING
45. DOTS
46. XR: PART 1
47. XR: PART 2
48. MULTIPLAYER NETCODE: PART - 2
49. RECORDER
50. ANIMATION RIGGING

UNITY CHEAT-SHEET

EVENT FUNCTIONS

```
// Unity Event Functions Bible
public class EventFunctionsCheatSheet : MonoBehaviour
{
    // Initialization & Lifecycle
    void Awake() { }          // First: Core setup & references
    void OnEnable() { }       // When object/component activates
    void Start() { }          // Before first frame, after all Awake()
    void OnDisable() { }      // When object/component deactivates
    void OnDestroy() { }      // Final cleanup before destruction

    // Update Loops
    void Update() { }         // Every frame - main game logic
    void FixedUpdate() { }    // Fixed timestep - physics only
    void LateUpdate() { }     // After Updates - cameras & final adjustments

    // Physics & Collisions
    void OnCollisionEnter(Collision other) { }  // First contact
    void OnCollisionStay(Collision other) { }   // During contact
    void OnCollisionExit(Collision other) { }   // Contact ends
    void OnTriggerEnter(Collider other) { }     // Enter trigger zone
    void OnTriggerStay(Collider other) { }      // Inside trigger
    void OnTriggerExit(Collider other) { }      // Exit trigger

    // Rendering & Visibility
    void OnBecameVisible() { }     // Object enters any camera view
    void OnBecameInvisible() { }   // Object exits all camera views
    void OnRenderObject() { }      // After camera renders scene
    void OnWillRenderObject() { } // Before each camera render
    void OnGUI() { }               // IMGUI rendering (Editor/Debug)

    // Application Events
    void OnApplicationFocus(bool focus) { }   // Window focus change
    void OnApplicationPause(bool pause) { }   // App pause/resume
    void OnApplicationQuit() { }              // Before app closes
}
```

UNITY's VILLAINS CHEAT-SHEET:
COMPONENT COMMUNICATION

```
public class ComponentCommunicationGuide : MonoBehaviour
{
    // Direct Component Access
    GetComponent<T>();
    GetComponents<T>();
    GetComponentInChildren<T>();
    GetComponentInParent<T>();

    // Scene-Wide Component Search
    FindObjectOfType<T>();
    FindObjectsOfType<T>();

    // GameObject Search
    GameObject.Find(string name);
    GameObject.FindWithTag(string tag);
    GameObject.FindGameObjectsWithTag(string tag);

    // Direct References (Recommended)
    [SerializeField] componentRef;
    cachedComponent;
}
```

DATA STRUCTURES
UNITY CHEAT-SHEET

```
// Unity Data Structures Cheat Sheet
public class DataStructuresGuide : MonoBehaviour
{
    // Basic Collections
    int[] fixedArray;                          // Fixed size, contiguous memory
    List<T> dynamicList;                       // Flexible size, GC allocated
    Dictionary<K,V> lookupTable;               // Key-value pairs, hash-based

    // Specialized Unity Collections
    [SerializeField]
    SerializedDictionary<K,V> editorDict;      // Inspector-friendly dictionary

    ObservableList<T> reactiveList;            // Event-driven list
    CircularBuffer<T> ringBuffer;              // Fixed-size wraparound
    ObjectPool<T> recyclePool;                 // Object recycling

    // High Performance (Native Collections)
    NativeArray<T> burstArray;                 // Burst-compatible array
    NativeList<T> burstList;                   // Dynamic native list
    NativeHashMap<K,V> burstDict;              // Native dictionary

    // Memory Layout Patterns
    struct ComponentData                       // Cache-friendly parallel arrays
    {
        float[] positions;
        float[] velocities;
        float[] health;
    }

    // Spatial Partitioning
    SpatialHash<T> worldPartition;             // Space-based lookups
}
```

UNITY CHEAT-SHEET
INSPECTOR ATTRIBUTES

PART-4

```
public class UnityAttributesCheatSheet : MonoBehaviour{
    // Data Serialization
    [SerializeField]                    // Make private fields visible in Inspector
    [Serializable]                      // Mark class/struct for serialization
    [Range(min, max)]                   // Add numeric range constraints
    [Header("Header Text")]             // Add section headers
    [Tooltip("Tooltip Text")]           // Show hover information
    [HideInInspector]                   // Hide public fields
    [Space(10)]                         // Add vertical spacing
    [TextArea]                          // Multi-line text with scrollbar
    [Multiline]                         // Multi-line text without scrollbar
    [ColorUsage(true, true)]            // Customize color picker (HDR, alpha)
    [Min(0)]                            // Set minimum value
    [Max(100)]                          // Set maximum value
    [FormerlySerializedAs("oldName")] // Maintain serialization after rename
    [ContextMenu("Function")]           // Add Inspector context menu
    [ContextMenuItem("Menu", "Function")] // Add field context menu
    [Delayed]                           // Update only after edit complete
    [NonSerialized]                     // Prevent serialization

    // Custom Editor Attributes
    [CustomEditor(typeof(Type))]    // Create custom Inspector
    [PropertyDrawer(typeof(Type))] // Custom property rendering
    [CanEditMultipleObjects]        // Enable multi-object editing
    [ExecuteInEditMode]             // Run in Editor mode
    [ExecuteAlways]                 // Run in Editor and Play mode
    [CreateAssetMenu]               // Create custom assets
    [AddComponentMenu("Path")]      // Add to Component menu
    [DisallowMultipleComponent]     // Prevent component duplication
    [DefaultExecutionOrder(0)]      // Set script execution order
    [HelpURL("URL")]                // Add help documentation link
    [InitializeOnLoad]              // Execute on Editor load
    [InitializeOnLoadMethod]        // Method called on load
    [SelectionBase]                 // Set primary selection target
}
```

UNITY CHEAT-SHEET
DEBUGGING

PART-5

```
// Unity Debugging Cheat Sheet
public class DebuggingGuide : MonoBehaviour
{
    // Console Debug Functions
    Debug.Log("message");                                   // General information logging
    Debug.LogWarning("warning");                            // Non-critical issues
    Debug.LogError("error");                                // Critical problems
    Debug.LogAssertion("assert");                           // Condition validation
    Debug.LogException(exception);                          // Exception handling
    Debug.ClearDeveloperConsole();                          // Clear the console logs
    Debug.Break();                                          // Pause the editor for inspection

    // Scene View Debug
    Debug.DrawLine(start, end, color);                      // Draw lines in the scene
    Debug.DrawRay(origin, direction, color);                // Visualize raycasts
    Debug.DrawLine(transform.position, targetPos, Color.green, 2f);  // Line with duration
    Debug.DrawRay(transform.position, Vector3.forward * 10, Color.red);  // Directional ray

    // Editor Gizmos
    void OnDrawGizmos() =>                                  // Draw a sphere at object position
            Gizmos.DrawSphere(transform.position, 1f);

    void OnDrawGizmosSelected() =>                          // Draw wireframe cube
            Gizmos.DrawWireCube(transform.position, Vector3.one);


    // Performance Profiling
    ProfilerMarker marker = new ProfilerMarker("PerformanceMarker");
    marker.Begin();                                         // Begin profiling a section of code
    marker.End();                                           // End profiling

    Profiler.BeginSample("SampleTask");                     // Start a custom profiler sample
    Profiler.EndSample();                                   // End the sample

    // Physics Debugging
    Debug.Break();                                          // Pause the editor on a specific frame
    Physics.IgnoreCollision(collider1, collider2, true);    // Temporarily ignore collisions
    Physics.OverlapSphere(transform.position, 5f);          // Visualize overlapping objects

    // Network Debugging
    Debug.Log($"Network status: {Application.internetReachability}");  // Check network status
    Debug.Log($"Host name: {SystemInfo.deviceName}");       // Log the host name

    // Additional Utility
    Debug.Log($"Active scene: {UnityEngine.SceneManagement.SceneManager.GetActiveScene().name}"); // Active scene
    Debug.Log($"Frame count: {Time.frameCount}");           // Log current frame count
    Debug.Log($"Delta time: {Time.deltaTime}");             // Log time between frames
}
```

UNITY CHEAT-SHEET

PART-6/50

1/2 NEW INPUT SYSTEM

```
// Unity Input System Bible
// Setup & Configuration
[SerializeField] InputActionAsset inputAsset;          // Asset reference
[SerializeField] PlayerInput playerInput;              // Player Input Component
InputActionMap gameplayMap;                            // Action Map reference
InputAction moveAction;                                // Single action

// Action Maps & Bindings
new InputActionMap("GameplayMap");                     // Create map
action.AddBinding("<Keyboard>/w");                     // Basic binding
action.AddBinding("<Gamepad>/leftStick");              // Controller binding
action.AddBinding("<Mouse>/delta");                    // Mouse binding
action.AddBinding("<Touchscreen>/primaryTouch");       // Touch binding
action.AddBinding("<XRController>/trigger");           // VR binding

// Composite Bindings
action.AddCompositeBinding("2DVector")                 // WASD/Arrow Setup
    .With("Up", "<Keyboard>/w")
    .With("Down", "<Keyboard>/s")
    .With("Left", "<Keyboard>/a")
    .With("Right", "<Keyboard>/d");

// Input Path Syntax Examples
"<Gamepad>/"                                           // All gamepad inputs
"<Keyboard>/anyKey"                                    // Any keyboard key
"<Mouse>/leftButton"                                   // Mouse buttons
"<XRController>{LeftHand}/trigger"                     // VR controls

// Reading Input Values
action.ReadValue<Vector2>();                           // Get vector input
action.ReadValue<float>();                             // Get scalar input
action.IsPressed();                                    // Button state
action.WasPressedThisFrame();                          // Frame check
action.WasReleasedThisFrame();                         // Release check
action.IsInProgress();                                 // Action status

// Input Events & Callbacks
action.performed += ctx => { };                        // Input performed
action.started += ctx => { };                          // Input started
action.canceled += ctx => { };                         // Input canceled
InputSystem.onDeviceChange += (d,c) => { };            // Device events
playerInput.onActionTriggered += ctx => { };           // PlayerInput events
```

MORE ABOUT NEW INPUT SYSTEM IN NEXT SHEET

UNITY CHEAT-SHEET

PART-7/50

2/2 NEW INPUT SYSTEM

```
// Interaction Types
action.AddBinding().WithInteraction("Tap");          // Quick press
action.AddBinding().WithInteraction("Hold");         // Press and hold
action.AddBinding().WithInteraction("SlowTap");      // Delayed tap
action.AddBinding().WithInteraction("MultiTap");     // Multiple taps

// Device Management
InputSystem.devices;                                 // All devices
Gamepad.all;                                         // All gamepads
Gamepad.current;                                     // Active gamepad
InputSystem.EnableDevice(device);                    // Enable device
InputSystem.DisableDevice(device);                   // Disable device

// Haptic Feedback
gamepad.SetMotorSpeeds(leftMotor, rightMotor);       // Set vibration
gamepad.PauseHaptics();                              // Pause haptics
gamepad.ResumeHaptics();                             // Resume haptics
gamepad.ResetHaptics();                              // Stop haptics

// Control Schemes
playerInput.SwitchCurrentControlScheme("Gamepad");   // Switch to gamepad
playerInput.currentControlScheme;                    // Current scheme
playerInput.devices;                                 // Active devices

// Input Settings
InputSystem.settings.updateMode;                     // Update timing
InputSystem.settings.filterNoiseOnCurrent;           // Noise filtering
action.processors = "normalize(min=0,max=1)";        // Input processing

// Debugging Tools
Debug.Log(action.enabled);                           // Action state
Debug.Log(action.controls);                          // Bound controls
Debug.Log(InputSystem.devices.Count);                // Device count
```

MORE ABOUT NEW INPUT SYSTEM IN PREVIOUS SHEET

UNITY CHEAT-SHEET
RENDERING PIPELINE

PART-8/50

```cpp
public class UnityRenderPipeline : MonoBehaviour
{
    // === Camera === //
    Camera.main.RenderWithShader();     // Renders the camera's view using a replacement shader.
    Camera.main.allowHDR;               // Indicates if High Dynamic Range (HDR) rendering is enabled.
    Camera.main.allowMSAA;              // Indicates if Multi-Sample Anti-Aliasing (MSAA) is enabled.
    Camera.main.renderingPath;          // Specifies the rendering path used by the camera.

    // === QualitySettings === //
    QualitySettings.SetQualityLevel();  // Sets the current quality level by index.
    QualitySettings.antiAliasing;       // Gets or sets the level of Multi-Sample Anti-Aliasing (MSAA).
    QualitySettings.softParticles;      // Indicates if soft blending is used for particles.
    QualitySettings.shadows;            // Specifies the shadow quality.
    QualitySettings.shadowResolution;   // Defines the resolution of shadows.
    QualitySettings.shadowDistance;     // Sets the maximum distance at which shadows are visible.
    QualitySettings.shadowCascades;     // Determines the number of cascades used for directional light shadows.

    // === Light === //
    light.renderMode;                   // Specifies whether the light uses per-pixel or per-vertex rendering.
    light.shadows;                      // Determines the type of shadows the light casts.

    // === ReflectionProbe === //
    reflectionProbe.mode;               // Sets the probe to real-time or baked reflections.
    reflectionProbe.resolution;         // Specifies the resolution of the reflection probe.
    reflectionProbe.refreshMode;        // Determines when the reflection probe updates its reflections.

    // === RenderSettings === //
    RenderSettings.ambientMode;         // Sets the mode of ambient light in the scene.
    RenderSettings.ambientProbe;        // Provides the ambient light data for the scene.

    // === Shader === //
    Shader.globalRenderPipeline;        // Specifies the name of the current render pipeline.
    Shader.Find();                      // Retrieves a shader by its name.

    // === Material === //
    material.EnableKeyword();           // Enables a shader keyword for this material.
    material.globalIlluminationFlags;   // Controls how the material interacts with global illumination.
    material.enableInstancing;          // Indicates if GPU instancing is enabled for this material.

    // === Graphics === //
    Graphics.DrawMeshInstanced();       // Renders multiple instances of a mesh with the same material.
    Graphics.DrawProceduralNow();       // Draws procedural geometry immediately.

    // === LODGroup === //
    LODGroup.ForceLOD();                // Forces the LOD group to display a specific LOD level.
    new LOD();                          // Initializes a new Level of Detail (LOD) instance.
}
```

UNITY CHEAT-SHEET
ANIMATIONS

PART-9/50

```cs
public class Animations : MonoBehaviour
{
    // === Animator === //
    animator.SetBool();                         // Sets a boolean parameter.
    animator.SetFloat();                        // Sets a float parameter.
    animator.SetInteger();                      // Sets an integer parameter.
    animator.SetTrigger();                      // Sets a trigger parameter.
    animator.ResetTrigger();                    // Resets a trigger parameter.
    animator.Play();                            // Plays an animation by name.
    animator.CrossFade();                       // Crossfades to another animation.
    animator.PlayInFixedTime();                 // Plays an animation at a fixed time.
    animator.StopPlayback();                    // Stops current playback.
    animator.IsInTransition();                  // Checks if a layer is in transition.
    animator.GetCurrentAnimatorStateInfo();     // Gets the current state info of a layer.
    animator.GetNextAnimatorStateInfo();        // Gets the next state info of a layer.
    animator.GetCurrentAnimatorClipInfo();      // Gets the current clip info of a layer.
    animator.Rebind();                          // Rebinds the animator.
    animator.Update();                          // Updates the animator with delta time.
    animator.SetLayerWeight();                  // Sets blending weight for a layer.
    animator.GetLayerWeight();                  // Gets blending weight of a layer.
    animator.GetLayerName();                    // Gets the name of a layer.
    animator.GetLayerIndex();                   // Gets the index of a layer.
    animator.applyRootMotion;                   // Enables or disables root motion.
    animator.bodyPosition;                      // Gets or sets animator's body position.
    animator.bodyRotation;                      // Gets or sets animator's body rotation.
    animator.updateMode;                        // Sets how the animator is updated.
    animator.stabilizeFeet;                     // Enables foot stabilization.

    // === Animator.StringToHash === //
    Animator.StringToHash();                    // Converts a parameter name to hash for performance.

    // === AnimationEvent === //
    AnimationEvent.functionName;                // Name of the callback function.
    AnimationEvent.floatParameter;              // Float argument for the callback.
    AnimationEvent.intParameter;                // Int argument for the callback.
    AnimationEvent.stringParameter;             // String argument for the callback.
    AnimationEvent.time;                        // Time in seconds when event triggers.

    // === AnimationCurve === //
    AnimationCurve.Linear();                    // Creates a linear animation curve.
    AnimationCurve.EaseInOut();                 // Creates an ease-in-out curve.
    AnimationCurve.AddKey();                    // Adds a keyframe.
    AnimationCurve.RemoveKey();                 // Removes a keyframe by index.
    AnimationCurve.MoveKey();                   // Moves a keyframe to a new time.
    animationCurve.Evaluate();                  // Evaluates the curve at a given time.
}
```

UNITY CHEAT-SHEET

PART-10/50

AI: NAVMESH AGENT

```
public class AINavmesh : MonoBehaviour
{
    // === NavMeshAgent === //
    agent.SetDestination();              // Sets the target destination.
    agent.ResetPath();                   // Clears the current navigation path.
    agent.Move();                        // Moves the agent manually.
    agent.Warp();                        // Warps the agent to a specific position.
    agent.CompleteOffMeshLink();         // Completes movement across an off-mesh link.
    agent.isStopped;                     // Checks if agent is currently stopped.
    agent.velocity;                      // Current velocity of the agent.
    agent.remainingDistance;             // Distance remaining to the destination.
    agent.pathPending;                   // Whether a path is being computed.
    agent.hasPath;                       // Whether the agent has a current path.
    agent.pathStatus;                    // The status of the current path.
    agent.isOnOffMeshLink;               // Checks if agent is on an off-mesh link.
    agent.currentOffMeshLinkData;        // Data about the current off-mesh link.
    agent.speed;                         // Speed of the agent.
    agent.angularSpeed;                  // Angular rotation speed.
    agent.acceleration;                  // Acceleration rate.
    agent.stoppingDistance;              // Distance to stop before reaching destination.
    agent.autoBraking;                   // Enables or disables slowing near destination.
    agent.radius;                        // Collision radius of the agent.

    // === NavMesh === //
    NavMesh.CalculatePath();             // Calculates a path between two points.
    NavMesh.SamplePosition();            // Finds the closest point on the NavMesh.
    NavMesh.FindClosestEdge();           // Finds the nearest NavMesh edge from a point.
    NavMesh.GetAreaFromName();           // Gets area index by name.
    NavMesh.AddNavMeshData();            // Adds NavMesh data to the scene.
    NavMesh.RemoveAllNavMeshData();      // Removes all NavMesh data.
    NavMesh.Raycast();                   // Performs line trace to check NavMesh collisions.

    // === NavMeshPath === //
    agent.CalculatePath();               // Calculates path and stores in NavMeshPath.
    path.status;                         // Gets the current path status.

    // === NavMeshHit === //
    hit.position;                        // Position where the raycast hit.
    hit.normal;                          // Surface normal at hit point.
    hit.mask;                            // Area mask of the hit.
    hit.distance;                        // Distance from origin to hit.

    // === NavMeshQueryFilter === //
    filter.areaMask;                     // Defines area mask for queries.
    filter.agentTypeID;                  // Specifies agent type ID.

    // === NavMeshObstacle === //
    obstacle.carving;                    // Enables dynamic carving for obstacle.
    obstacle.carveOnlyStationary;        // Carves only when stationary.
    obstacle.shape;                      // Shape of the obstacle.
}
```

UNITY CHEAT-SHEET

PART-11/50

UI

```
public class UI : MonoBehaviour
{
    // === Canvas === //
    Canvas.renderMode;                  // Canvas render mode.
    Canvas.worldCamera;                 // Camera used for rendering.
    Canvas.pixelPerfect;                // Enables pixel-perfect rendering.
    Canvas.sortingOrder;                // Canvas sorting order.
    Canvas.ForceUpdateCanvases();       // Forces canvas layout update.

    // === RectTransform === //
    rectTransform.anchoredPosition; // Pivot position relative to anchors.
    rectTransform.anchorMin;        // Lower anchor point.
    rectTransform.anchorMax;        // Upper anchor point.
    rectTransform.offsetMin;        // Offset from lower anchor.
    rectTransform.offsetMax;        // Offset from upper anchor.
    rectTransform.sizeDelta;        // Size between anchors.
    rectTransform.pivot;            // Pivot point for rotation/scaling.

    // === CanvasGroup === //
    canvasGroup.alpha;              // Canvas transparency.
    canvasGroup.interactable;       // Enables UI interaction.
    canvasGroup.blocksRaycasts;     // Allows raycast detection.

    // === TextMeshProUGUI === //
    tmp.text;                       // Text content.
    tmp.fontSize;                   // Font size.
    tmp.alignment;                  // Text alignment.
    tmp.color;                      // Text color.

    // === RawImage === //
    rawImage.texture;               // Displayed texture.
    rawImage.color;                 // Tint color.
    rawImage.uvRect;                // UV mapping rectangle.

    // === Button === //
    button.onClick.AddListener();   // Button click listener.
    button.interactable;            // Button interactivity.

    // === Image === //
    image.sprite;                   // Displayed sprite.
    image.color;                    // Image tint color.
    image.fillAmount;               // Fill amount (e.g., progress).
}
```

UNITY CHEAT-SHEET
COLLISIONS

PART-12/50

```
public class UnityCollisions : MonoBehaviour
{
    // === Collider === //
    collider.enabled;                       // Enables or disables the collider.
    collider.isTrigger;                     // Collider acts as a trigger, not a solid object.
    collider.material;                      // Physics material for friction and bounce.
    collider.bounds;                        // World-space bounding box of the collider.
    collider.sharedMaterial;                // Shared physics material across colliders.
    collider.contactOffset;                 // Skin width padding for collision detection.

    // === BoxCollider === //
    boxCollider.size;                       // Defines the dimensions of the box.
    boxCollider.center;                     // Set the center of the box collider.

    // === SphereCollider === //
    sphereCollider.radius;                  // Radius of the sphere collider.
    sphereCollider.center;                  // Center of the sphere collider.

    // === CapsuleCollider === //
    capsuleCollider.radius;                 // Radius of the capsule.
    capsuleCollider.height;                 // Height of the capsule.
    capsuleCollider.direction;              // Axis of capsule height (X=0, Y=1, Z=2).
    capsuleCollider.center;                 // Center of the capsule collider.

    // === MeshCollider === //
    meshCollider.sharedMesh;                // Mesh assigned to the mesh collider.
    meshCollider.convex;                    // Should be set to true for dynamic rigidbodies.

    // === TerrainCollider === //
    terrainCollider.terrainData;            // Assign terrain data to the terrain collider.
    terrainCollider.bounds;                 // World-space bounds of the terrain collider.

    // === WheelCollider === //
    wheelCollider.motorTorque;              // Applies motor torque to the wheel.
    wheelCollider.brakeTorque;              // Applies brake torque to the wheel.
    wheelCollider.steerAngle;               // Steering angle in degrees.
    wheelCollider.suspensionSpring;         // Suspension settings for the wheel collider.

    // === Collision Events === //
    OnCollisionEnter();                     // Called on collision start.
    OnCollisionStay();                      // Called while colliding.
    OnCollisionExit();                      // Called when collision ends.
    OnTriggerEnter();                       // Called when entering trigger collider.
    OnTriggerStay();                        // Called while staying in trigger.
    OnTriggerExit();                        // Called when exiting trigger collider.

    // === Physics (Static Methods) === //
    Physics.IgnoreCollision();              // Prevent collision between two colliders.
    Physics.IgnoreLayerCollision();         // Ignore collisions between layers.
    Physics.OverlapSphere();                // Returns all colliders within a sphere.
    Physics.CheckBox();                     // Checks if a box overlaps colliders.
    Physics.CheckSphere();                  // Checks if a sphere overlaps colliders.
    Physics.CheckCapsule();                 // Checks if a capsule overlaps colliders.
    Physics.Raycast();                      // Casts a ray and checks for collision.
    Physics.RaycastAll();                   // Returns all ray hits.
    Physics.RaycastNonAlloc();              // Raycast without allocating new array.
    Physics.BoxCast();                      // Casts a box and checks for collision.
}
```

UNITY CHEAT-SHEET

PART-13/50

CAMERA

```
public class UnityCameraCheatSheet : MonoBehaviour
{
    // === Camera Properties === //
    camera.fieldOfView;                         // View angle in degrees.
    camera.orthographic;                        // Enables orthographic mode.
    camera.orthographicSize;                    // Half height of the camera view.
    camera.aspect;                              // Width-to-height ratio.
    camera.nearClipPlane;                       // Closest render distance.
    camera.farClipPlane;                        // Farthest render distance.
    camera.depth;                               // Rendering order.
    camera.cullingMask;                         // Layers the camera renders.
    camera.backgroundColor;                     // Camera clear color.
    camera.clearFlags;                          // How the screen is cleared.

    // === Rendering and Effects === //
    camera.allowHDR;                            // Enables HDR rendering.
    camera.allowMSAA;                           // Enables MSAA.
    camera.renderingPath;                       // Rendering technique used.
    camera.targetTexture;                       // Output render texture.
    camera.depthTextureMode;                    // Depth texture generation.

    // === Camera Transformations === //
    camera.WorldToScreenPoint();                // World to screen position.
    camera.ScreenToWorldPoint();                // Screen to world position.
    camera.ScreenToViewportPoint();             // Screen to viewport position.
    camera.ViewportToScreenPoint();             // Viewport to screen position.
    camera.ViewportToWorldPoint();              // Viewport to world position.
    camera.ScreenPointToRay();                  // Screen point to ray.
    camera.ViewportPointToRay();                // Viewport point to ray.

    // === Physical Camera Properties === //
    camera.usePhysicalProperties;               // Enables physical camera mode.
    camera.focalLength;                         // Lens focal length (mm).
    camera.sensorSize;                          // Camera sensor size (mm).
    camera.lensShift;                           // Offset lens horizontally/vertically.
    camera.gateFit;                             // Fit mode for gate and sensor.

    // === Multi-Display Settings === //
    camera.targetDisplay;                       // Display index to render to.

    // === Camera Events === //
    OnPreCull();                                // Before culling.
    OnPreRender();                              // Before rendering.
    OnPostRender();                             // After rendering.

    // === Multi-Pass and Custom Renders === //
    camera.AddCommandBuffer();                  // Adds a custom render pass.
    camera.RemoveCommandBuffer();               // Removes a render pass.
}
```

UNITY CHEAT-SHEET

PHSIYCS: RIGIDBODY

PART-14/50

```
// Basic Movement
rb.velocity = new Vector3(5f, 0f, 0f);
rb.AddForce(Vector3.up * 500f);
rb.AddForce(Vector3.up * 10f, ForceMode.Impulse);
rb.MovePosition(transform.position + movement * Time.fixedDeltaTime);
Vector3 pointVel = rb.GetPointVelocity(worldPoint);
Vector3 relativeVel = rb.GetRelativePointVelocity(relativePoint);

// Rotation
rb.angularVelocity = new Vector3(0f, 90f, 0f);
rb.AddTorque(transform.up * 50f);
rb.AddRelativeTorque(Vector3.up * 10f);
rb.MoveRotation(Quaternion.Euler(0f, 90f, 0f));

// Impact & Physics State
rb.AddExplosionForce(1000f, explosionCenter, 10f, 2f);
rb.AddForceAtPosition(Vector3.up * 10f, transform.position + offset);
rb.Sleep();
rb.WakeUp();
bool isSleeping = rb.IsSleeping();

// Properties & Constraints
rb.mass = 5f;
rb.drag = 1f;
rb.angularDrag = 0.5f;
rb.useGravity = true;
rb.isKinematic = false;
rb.constraints = RigidbodyConstraints.FreezePositionX |
                 RigidbodyConstraints.FreezeRotationZ;

// Center of Mass
rb.centerOfMass = new Vector3(0f, 1f, 0f);
rb.ResetCenterOfMass();

// Velocity Operations
rb.GetPointVelocity(worldPoint);
rb.GetRelativePointVelocity(relativePoint);
```

UNITY CHEAT-SHEET

PART-15/50

SHADERS AND MATERIALS

```
public class ShadersAndMaterials : MonoBehaviour
{
    // === Material === //
    material.SetColor();                        // Set color.
    material.SetTexture();                      // Set texture.
    material.SetFloat();                        // Set float.
    material.mainTexture;                       // Main texture.
    material.mainTextureScale;                  // Texture scale.
    material.mainTextureOffset;                 // Texture offset.
    material.shader;                            // Assigned shader.

    // === Shader Keywords === //
    material.EnableKeyword();                   // Enable keyword.
    material.DisableKeyword();                  // Disable keyword.
    material.IsKeywordEnabled();                // Check keyword state.

    // === Shader Properties === //
    material.SetMatrix();                       // Set matrix.
    material.SetVector();                       // Set vector.

    // === GPU Instancing === //
    material.enableInstancing;                  // GPU instancing toggle.

    // === Rendering === //
    material.renderQueue;                       // Render queue.
    material.globalIlluminationFlags;           // GI flags.

    // === Shader Utilities === //
    Shader.PropertyToID();                      // Get property ID.
    Shader.SetGlobalFloat();                    // Set global float.
    Shader.SetGlobalVector();                   // Set global vector.
    Shader.SetGlobalColor();                    // Set global color.
    Shader.SetGlobalTexture();                  // Set global texture.
    Shader.EnableKeyword();                     // Enable global keyword.
    Shader.DisableKeyword();                    // Disable global keyword.
    Shader.WarmupAllShaders();                  // Precompile shaders.

    // === Advanced Shader Properties === //
    material.SetOverrideTag();                  // Override render tag.
    material.GetTag();                          // Get render tag.

    // === Shader Debugging === //
    material.shader.isSupported;                // Shader supported?
    material.shader.passCount;                  // Pass count.
    material.shader.name;                       // Shader name.

    // === Shader Graph === //
    material.SetVector();                       // Set Shader Graph vector.
    material.SetTexture();                      // Set Shader Graph texture.

    // === Shader Passes & Pipeline === //
    material.SetShaderPassEnabled();            // Enable pass.
    material.GetShaderPassEnabled();            // Check pass state.
    Shader.Find();                              // Find shader.
    material.shader.name;                       // Pipeline shader name.
    material.CopyPropertiesFromMaterial();      // Copy material props.
    material.doubleSidedGI;                     // Enable double-sided GI.
}
```

UNITY CHEAT-SHEET

PART-16/50

PARTICLE SYSTEM

```
public class ParticleSystem : MonoBehaviour
{
    // === Main Module === //
    main.duration;                          // System duration.
    main.loop;                              // Looping enabled.
    main.prewarm;                           // Prewarm enabled.
    main.startLifetime;                     // Initial particle lifetime.
    main.startSpeed;                        // Initial particle speed.
    main.startSize;                         // Initial particle size.
    main.startRotation;                     // Initial particle rotation.
    main.startColor;                        // Initial particle color.
    main.gravityModifier;                   // Gravity multiplier.

    // === Emission Module === //
    emission.enabled;                       // Emission enabled.
    emission.rateOverTime;                  // Emission rate over time.
    emission.rateOverDistance;              // Emission rate over distance.
    emission.SetBursts();                   // Set burst emissions.
    emission.GetBursts();                   // Get burst emissions.

    // === Shape Module === //
    shape.enabled;                          // Shape module enabled.
    shape.shapeType;                        // Shape of emission.
    shape.angle;                            // Angle of cone emission.
    shape.radius;                           // Radius of shape.
    shape.arc;                              // Arc angle for emission.

    // === Velocity over Lifetime === //
    velocityOverLifetime.enabled;           // Velocity over lifetime enabled.
    velocityOverLifetime.x;                 // X-axis velocity curve.
    velocityOverLifetime.y;                 // Y-axis velocity curve.
    velocityOverLifetime.z;                 // Z-axis velocity curve.

    // === Limit Velocity over Lifetime === //
    limitVelocityOverLifetime.enabled;      // Limit velocity over lifetime enabled.
    limitVelocityOverLifetime.limit;        // Velocity limit.
    limitVelocityOverLifetime.dampen;       // Dampen factor.

    // === Inherit Velocity Module === //
    inheritVelocity.enabled;                // Inherit velocity enabled.
    inheritVelocity.mode;                   // Inherit velocity mode.
    inheritVelocity.curve;                  // Velocity inheritance curve.

    // === Force over Lifetime Module === //
    forceOverLifetime.enabled;              // Force over lifetime enabled.
    forceOverLifetime.x;                    // X-axis force curve.
    forceOverLifetime.y;                    // Y-axis force curve.
    forceOverLifetime.z;                    // Z-axis force curve.

    // === Noise Module === //
    noise.enabled;                          // Noise module enabled.
    noise.strength;                         // Strength of noise.
    noise.frequency;                        // Frequency of noise.
    noise.scrollSpeed;                      // Scroll speed of noise.
}
```

UNITY CHEAT-SHEET

PART-17/50

TIMELINE AND CINEMACHINE

```
public class TimelineAndCinemachine : MonoBehaviour
{
    // === PlayableDirector === //
    playableDirector.playableAsset;                    // Assigns a timeline asset to the director.
    playableDirector.time;                             // Sets the current time on the timeline.
    playableDirector.Play();                           // Starts playing the timeline.
    playableDirector.Pause();                          // Pauses timeline playback.
    playableDirector.Stop();                           // Stops the timeline.
    playableDirector.Evaluate();                       // Evaluates timeline at the current time.
    playableDirector.playOnAwake;                      // Plays the timeline on Awake.
    playableDirector.extrapolationMode;                // Sets out-of-range time behavior.
    playableDirector.timeUpdateMode;                   // Sets how the timeline updates.
    playableDirector.duration;                         // Returns the duration of the timeline.
    playableDirector.state;                            // Gets the current state of the timeline.
    playableDirector.RebuildGraph();                   // Rebuilds the PlayableGraph for the timeline.

    // === Timeline Track Binding === //
    playableDirector.SetGenericBinding();              // Binds a track to a GameObject or component.
    playableDirector.ClearGenericBinding();            // Clears a previously bound track.
    timeLineAsset.CreateTrack();                       // Creates a new track (e.g., AnimationTrack).

    // === CinemachineVirtualCamera === //
    cinemachineVirtualCamera.m_Lens.FieldOfView;       // Sets the field of view.
    cinemachineVirtualCamera.m_Lens.Orthographic;      // Toggles orthographic mode.
    cinemachineVirtualCamera.m_Lens.NearClipPlane;     // Sets near clip plane distance.
    cinemachineVirtualCamera.m_Lens.FarClipPlane;      // Sets far clip plane distance.
    cinemachineVirtualCamera.Priority;                 // Sets camera priority.
    cinemachineVirtualCamera.m_FollowOffset;           // Sets the camera follow offset.
    cinemachineVirtualCamera.m_LookAtTargetAttachment; // Sets look-at target strength.

    // === CinemachineTransposer === //
    cinemachineTransposer.m_FollowOffset;              // Sets the follow offset.
    cinemachineTransposer.m_XDamping;                  // Sets damping on X-axis.
    cinemachineTransposer.m_YDamping;                  // Sets damping on Y-axis.
    cinemachineTransposer.m_ZDamping;                  // Sets damping on Z-axis.

    // === CinemachineComposer === //
    cinemachineComposer.m_TrackedObjectOffset;         // Sets the tracked object offset.
    cinemachineComposer.m_SoftZoneWidth;               // Width of the soft zone.
    cinemachineComposer.m_DeadZoneWidth;               // Width of the dead zone.
    cinemachineComposer.m_BiasX;                       // Bias along the X-axis.
    cinemachineComposer.m_BiasY;                       // Bias along the Y-axis.

    // === Follow and LookAt === //
    cinemachineVirtualCamera.Follow;                   // Sets the object to follow.
    cinemachineVirtualCamera.LookAt;                   // Sets the object to look at.

    // === CinemachineBlendDefinition === //
    blendDef.m_Style;                                  // Blend style (EaseInOut, etc).
    blendDef.m_Time;                                   // Blend duration.
    virtualCamera.SetBlendOverride();                  // Applies custom blend settings.
}
```

UNITY CHEAT-SHEET

GAME OBJECTS

Problem: To find Object in scene

Solution:

```
// Finding GameObjects
GameObject.Find("ObjectName")
GameObject.FindWithTag("TagName")
GameObject.FindGameObjectsWithTag("TagName")
```

Problem: Getting Attached Components

Solution:

```
// Component Management
gameObject.AddComponent<T>()
gameObject.GetComponent<T>()
gameObject.GetComponents<T>()
gameObject.GetComponentInChildren<T>()
gameObject.GetComponentInParent<T>()
```

Problem: Creating and Destroying Game Objects

Solution:

```
// Instantiation and Destruction
Instantiate(gameObject)
Destroy(gameObject)
Destroy(gameObject, float delay)
DontDestroyOnLoad(gameObject)
```

Problem: Creating and Destroying Game Objects

Solution:

```
// ═══ Hierarchy Control Functions with Child, Sibling, or Parent Keywords ═══ //
gameObject.transform.SetParent(parentTransform, worldPositionStays);
gameObject.transform.parent = parentTransform;
gameObject.transform.DetachChildren();
gameObject.transform.Find("ChildName");
gameObject.transform.GetChild(0);
int childCount = gameObject.transform.childCount;
gameObject.transform.IsChildOf(parentTransform);
gameObject.transform.GetSiblingIndex();
gameObject.transform.SetSiblingIndex(2);
gameObject.transform.root;
```

UNITY CHEAT-SHEET

PART-19/50

TERRAIN SYSTEM

```
public class TerrainSystem : MonoBehaviour
{
    // === Terrain Properties === //
    terrain.terrainData;                        // Assigns terrain data.
    terrain.heightmapPixelError;                // LOD accuracy.
    terrain.basemapDistance;                    // Distance to display base map.
    terrain.shadowCastingMode;                  // Enables/disables terrain shadows.
    terrain.materialTemplate;                   // Assigns custom terrain material.
    terrain.drawTreesAndFoliage;                // Toggles tree/foliage rendering.
    terrain.reflectionProbeUsage;               // Reflection probe mode.
    terrain.collectDetailPatches;               // Enables detail patch collection.
    terrain.allowAutoConnect;                   // Auto-connects to neighboring terrains.
    terrain.groupingID;                         // Group ID for terrain auto-connection.

    // === TerrainData Properties === //
    terrainData.heightmapResolution;            // Heightmap resolution.
    terrainData.size;                           // Terrain dimensions.
    terrainData.alphamapResolution;             // Splat map resolution.
    terrainData.baseMapResolution;              // Base map resolution.
    terrainData.SetHeights();                   // Set terrain heightmap.
    terrainData.SetAlphamaps();                 // Set alpha/splat map.
    terrainData.SetDetailResolution();          // Set detail map resolution.
    terrainData.wavingGrassStrength;            // Grass wave strength.
    terrainData.wavingGrassSpeed;               // Grass wave speed.
    terrainData.wavingGrassAmount;              // Grass wave amount.
    terrainData.wavingGrassTint;                // Grass tint color.
    terrainData.GetHeights();                   // Get terrain heightmap.

    // === Terrain Layers === //
    terrainLayer.diffuseTexture;                // Base texture.
    terrainLayer.normalMapTexture;              // Normal map.
    terrainLayer.maskMapTexture;                // Mask map.
    terrainLayer.tileSize;                      // Texture tile size.
    terrainLayer.tileOffset;                    // Texture tile offset.

    // === Tree and Detail Management === //
    terrainData.treeInstances;                  // All tree instances.
    terrainData.treePrototypes;                 // Tree prefab prototypes.
    terrainData.detailObjectDistance;           // Detail object draw distance.
    terrainData.detailObjectDensity;            // Detail object density.
    terrainData.RefreshPrototypes();            // Refresh prototype data.
    terrainData.SetDetailLayer();               // Set specific detail map layer.

    // === Terrain Physics === //
    terrainData.SetHeightsDelayLOD();           // Update heights with LOD.
    terrainData.SyncHeightmap();                // Sync physics with heightmap.
    terrain.terrainData.ContainsPosition();     // Check if point is within terrain.

    // === Terrain Texture Blending === //
    terrainData.GetAlphamaps();                 // Get splat/alpha map.
    terrainData.SetAlphamaps();                 // Set splat/alpha map.

    // === Terrain Debugging === //
    terrainData.GetInterpolatedHeight();        // Height at normalized coords.
    terrainData.GetInterpolatedNormal();        // Normal at normalized coords.
    terrainData.GetTreeInstance();              // Get specific tree instance.
}
```

UNITY CHEAT-SHEET

PART-20/50

AUDIO SYSTEM

```
public class AudioSystem : MonoBehaviour
{
    // === AudioSource === //
    audioSource.clip;                   // Assigned clip.
    audioSource.volume;                 // Set volume.
    audioSource.pitch;                  // Set pitch.
    audioSource.loop;                   // Enable looping.
    audioSource.mute;                   // Mute toggle.
    audioSource.spatialBlend;           // 2D/3D blend.
    audioSource.priority;               // Playback priority.
    audioSource.Play();                 // Play audio.
    audioSource.Stop();                 // Stop audio.
    audioSource.Pause();                // Pause audio.

    // === AudioListener === //
    AudioListener.pause;                // Pause all audio.
    AudioListener.volume;               // Global volume.
    AudioListener.GetSpectrumData();    // Get FFT spectrum data.
    AudioListener.GetOutputData();      // Get raw audio output.

    // === AudioClip === //
    audioClip.length;                   // Clip length in seconds.
    audioClip.channels;                 // Number of audio channels.
    audioClip.frequency;                // Sampling frequency.
    audioClip.LoadAudioData();          // Load audio to memory.
    audioClip.UnloadAudioData();        // Unload clip data.

    // === AudioMixer === //
    audioMixer.SetFloat();              // Set exposed param.
    audioMixer.GetFloat();              // Get exposed param.
    audioMixer.ClearFloat();            // Reset param to default.
    audioMixer.TransitionToSnapshots(); // Crossfade to snapshot.

    // === AudioMixerGroup === //
    audioSource.outputAudioMixerGroup;  // Assign mixer group.

    // === Microphone === //
    Microphone.devices;                 // Available microphones.
    Microphone.Start();                 // Start recording.
    Microphone.End();                   // Stop recording.
    Microphone.IsRecording();           // Is recording?
    Microphone.GetPosition();           // Current record position.

    // === AudioSettings === //
    AudioSettings.GetConfiguration();   // Audio config.
    AudioSettings.Reset();              // Reset audio system.
    AudioSettings.dspTime;              // DSP time.

    // === Audio Rolloff === //
    audioSource.minDistance;            // Min rolloff dist.
    audioSource.maxDistance;            // Max rolloff dist.
    audioSource.rolloffMode;            // Attenuation mode.
}
```

UNITY CHEAT-SHEET

PART-21/50

MULTIPLAYER NETCODE

```
public class UnityNetcode : MonoBehaviour
{
    // === NetworkManager === //
    NetworkManager.Singleton;                          // Singleton instance of the NetworkManager.
    NetworkManager.StartHost();                        // Starts server and local client.
    NetworkManager.StartServer();                      // Starts the server only.
    NetworkManager.StartClient();                      // Starts the client only.
    NetworkManager.Shutdown();                         // Shuts down the network session.
    NetworkManager.OnServerStarted += ();              // Triggered when the server starts.
    NetworkManager.OnClientConnectedCallback += ();    // Called when a client connects.
    NetworkManager.OnClientDisconnectCallback += ();   // Called when a client disconnects.

    networkManager.IsHost;                             // True if running as host.
    networkManager.IsServer;                           // True if running as server.
    networkManager.IsClient;                           // True if running as client.
    networkManager.IsListening;                        // True if listening for connections.
    networkManager.IsConnectedClient;                  // True if the client is connected.
    networkManager.LocalClientId;                      // Local client's unique ID.
    networkManager.ConnectedClients;                   // Dictionary of connected clients.
    networkManager.ConnectedClientsList;               // List of connected clients.
    networkManager.ConnectedClientsIds;                // List of connected client IDs.
    networkManager.NetworkConfig;                      // Configuration for networking.
    networkManager.SceneManager;                       // Scene manager for network loading.
    networkManager.CustomMessagingManager;             // Manager for custom messages.
    networkManager.NetworkTimeSystem;                  // Tracks synced network time.
    networkManager.NetworkTickSystem;                  // Tracks synced tick count.
    networkManager.ConnectionApprovalCallback += ();   // Callback for client approval.

    // === NetworkObject === //
    networkObject.Spawn();                             // Spawns the object on the network.
    networkObject.SpawnWithOwnership(0);               // Spawns the object with a specific owner.
    networkObject.Despawn();                           // Despawns the object.
    networkObject.NetworkObjectId;                     // Unique ID of the object.
    networkObject.OwnerClientId;                       // Client ID that owns the object.
    networkObject.IsOwner;                             // True if this client owns the object.
    networkObject.IsOwnedByServer;                     // True if the server owns the object.
    networkObject.IsSpawned;                           // True if the object is spawned.
    networkObject.TrySetParent(transform);             // Sets the parent of the object (server only).
    networkObject.RemoveParent();                      // Removes the object's parent (server only).
    networkObject.ChangeOwnership(1);                  // Transfers ownership to a client.
    networkObject.RemoveOwnership();                   // Removes client ownership.
}
```

UNITY CHEAT-SHEET

PART-22/50

PHYSICS PART 2

```
public class UnityPhysics : MonoBehaviour
{
    // === Rigidbody === //
    rigidbody.AddForce();                   // Apply force.
    rigidbody.AddTorque();                  // Apply torque.
    rigidbody.interpolation;                // Motion smoothing.
    rigidbody.collisionDetectionMode;       // Collision detection mode.
    rigidbody.isKinematic;                  // Toggle kinematic state.
    rigidbody.Sleep();                      // Put to sleep.
    rigidbody.WakeUp();                     // Wake up.
    rigidbody.useGravity;                   // Enable gravity.
    rigidbody.constraints;                  // Set movement/rotation constraints.

    // === Physics === //
    Physics.Raycast();                      // Perform raycast.
    Physics.OverlapSphere();                // Detect colliders in radius.
    Physics.autoSimulation;                 // Enable/disable auto simulation.
    Physics.Simulate();                     // Manually simulate physics.
    Physics.gravity;                        // Set custom gravity.
    Physics.IgnoreCollision();              // Ignore collisions.

    // === PhysicMaterial === //
    PhysicMaterial.bounciness;              // Bounciness level.
    PhysicMaterial.frictionCombine;         // Friction combine mode.

    // === Collider === //
    collider.material;                      // Assign physics material.
    collider.isTrigger;                     // Set as trigger.

    // === Joint === //
    hingeJoint.useSpring;                   // Enable spring.
    hingeJoint.spring;                      // Set spring properties.

    // === LayerMask === //
    LayerMask.GetMask();                    // Create layer mask.
    LayerMask.NameToLayer();                // Convert name to layer index.
    LayerMask.LayerToName();                // Convert layer index to name.
    LayerMask.value;                        // Layer mask bitmask value.
    LayerMask.op_Implicit();                // Implicit conversion between LayerMask and int.
}
```

UNITY CHEAT-SHEET

PART-23/50

UI - Part 2

```
public class UnityUIPart2 : MonoBehaviour
{
    // === Slider === //
    slider.value;                                   // Current value.
    slider.minValue;                                // Minimum value.
    slider.maxValue;                                // Maximum value.
    slider.wholeNumbers;                            // Use whole numbers only.
    slider.onValueChanged.AddListener();            // Listener for value change.

    // === Toggle === //
    toggle.isOn;                                    // Toggle state.
    toggle.onValueChanged.AddListener();            // Listener for toggle change.
    toggle.graphic;                                 // Target graphic for toggle.
    toggle.interactable;                            // Is toggle interactable?

    // === ScrollRect === //
    scrollRect.content;                             // Scrollable content.
    scrollRect.vertical;                            // Enable vertical scroll.
    scrollRect.horizontal;                          // Enable horizontal scroll.
    scrollRect.normalizedPosition;                  // Scroll position (0-1).
    scrollRect.velocity;                            // Current scroll velocity.
    scrollRect.onValueChanged.AddListener();        // Listener for scroll movement.

    // === InputField === //
    inputField.text;                                // Input text.
    inputField.placeholder;                         // Placeholder graphic/text.
    inputField.characterLimit;                      // Max characters allowed.
    inputField.lineType;                            // Line type (single/multi-line).
    inputField.contentType;                         // Content type (standard, password, etc.).
    inputField.onValueChanged.AddListener();        // Listener for text change.
    inputField.onEndEdit.AddListener();             // Listener for edit completion.

    // === Dropdown === //
    dropdown.options;                               // List of dropdown options.
    dropdown.AddOptions();                          // Add options to dropdown.
    dropdown.value;                                 // Selected index.
    dropdown.captionText;                           // Label displaying selected option.
    dropdown.onValueChanged.AddListener();          // Listener for selection change.

    // === EventSystem === //
    EventSystem.current.SetSelectedGameObject();    // Set selected UI object.
    EventSystem.current.currentSelectedGameObject;  // Currently selected object.
    EventSystem.current.IsPointerOverGameObject();  // Is pointer over any UI?

    // === Scrollbar === //
    scrollbar.value;                                // Scrollbar value.
    scrollbar.direction;                            // Scrollbar direction.
    scrollbar.numberOfSteps;                        // Discrete scroll steps.
    scrollbar.onValueChanged.AddListener();         // Listener for scrollbar value.

    // === GraphicRaycaster === //
    graphicRaycaster.ignoreReversedGraphics;        // Ignore backfacing UI elements.
    graphicRaycaster.blockingObjects;               // Blocking object settings.
    graphicRaycaster.enabled;                       // Enable or disable raycaster.
}
```

UNITY CHEAT-SHEET

PART-24/50

ANIMATIONS - Part 2

```
public class UnityAnimation : MonoBehaviour
{
    // === PlayableGraph === //
    PlayableGraph.Create();                                  // Creates a new PlayableGraph.
    playableGraph.Play();                                    // Starts the playable graph.
    playableGraph.Stop();                                    // Stops the playable graph.
    playableGraph.IsValid();                                 // Checks if the graph is valid.
    playableGraph.Destroy();                                 // Destroys the playable graph.
    playableGraph.IsPlaying();                               // Checks if the graph is playing.
    playableGraph.Evaluate();                                // Manually evaluates the graph.

    // === AnimationPlayableOutput === //
    AnimationPlayableOutput.Create();                        // Creates an animation output for a graph.
    animationOutput.SetTarget();                             // Sets the Animator target.
    animationOutput.GetTarget();                             // Gets the Animator target.

    // === AnimationClipPlayable === //
    AnimationClipPlayable.Create();                          // Creates a playable from an AnimationClip.
    animationClipPlayable.SetApplyFootIK();                  // Enables/disables foot IK.
    animationClipPlayable.SetRemoveStartOffset();            // Removes clip start offset.
    animationClipPlayable.SetTime();                         // Sets playback time.
    animationClipPlayable.SetSpeed();                        // Sets playback speed.
    animationClipPlayable.GetAnimationClip();                // Returns the AnimationClip used.

    // === AnimationMixerPlayable === //
    AnimationMixerPlayable.Create();                         // Creates a new mixer playable.
    mixerPlayable.ConnectInput();                            // Connects an input to the mixer.
    mixerPlayable.SetInputWeight();                          // Sets input weight on the mixer.

    // === AnimationLayerMixerPlayable === //
    AnimationLayerMixerPlayable.Create();                    // Creates a new layer mixer.
    layerMixerPlayable.SetLayerMaskFromAvatarMask();         // Sets layer mask from avatar mask.
    layerMixerPlayable.SetLayerAdditive();                   // Sets a layer to additive blending.

    // === AnimatorControllerPlayable === //
    AnimatorControllerPlayable.Create();                     // Creates an animator controller playable.
    animatorControllerPlayable.Play();                       // Plays a state by name.
    animatorControllerPlayable.GetCurrentAnimatorStateInfo();// Gets current state info.
    animatorControllerPlayable.GetNextAnimatorStateInfo();   // Gets next state info.

    // === AnimatorStateInfo === //
    stateInfo.IsName();                                      // Checks if state matches given name.
    stateInfo.IsTag();                                       // Checks if state has specified tag.
    stateInfo.normalizedTime;                                // Normalized time (0-1).
    stateInfo.length;                                        // Length of the animation state.
    stateInfo.loop;                                          // Indicates if state loops.

    // === AvatarMask === //
    avatarMask.AddTransformPath();                           // Adds a transform path to the mask.
    avatarMask.RemoveTransformPath();                        // Removes a transform path by index.
    avatarMask.GetTransformPath();                           // Gets transform path by index.
    avatarMask.transformCount;                               // Number of transforms in the mask.

    // === AnimationMode (Editor only) === //
    AnimationMode.StartAnimationMode();                      // Starts Unity animation mode.
    AnimationMode.StopAnimationMode();                       // Stops Unity animation mode.
    AnimationMode.SampleAnimationClip();                     // Samples clip on GameObject at time.
}
```

UNITY CHEAT-SHEET

PART-25/50

VECTOR3 OPERATIONS

```
public class UnityVector3Operations : MonoBehaviour
{
    // === Vector3 Constants === //
    Vector3.zero;                                   // Zero vector.
    Vector3.one;                                    // Vector with all components 1.
    Vector3.up;                                     // Unit vector up.
    Vector3.forward;                                // Unit vector forward.
    Vector3.right;                                  // Unit vector right.

    // === Vector3 Static Methods === //
    Vector3.Dot();                                  // Dot product of two vectors.
    Vector3.Cross();                                // Cross product of two vectors.
    Vector3.Lerp();                                 // Linear interpolation.
    Vector3.Slerp();                                // Spherical interpolation.
    Vector3.MoveTowards();                          // Move towards target.
    Vector3.Distance();                             // Distance between vectors.
    Vector3.Reflect();                              // Reflect vector off normal.
    Vector3.Project();                              // Project one vector onto another.
    Vector3.Angle();                                // Angle between vectors.
    Vector3.SignedAngle();                          // Signed angle between vectors.
    Vector3.SmoothDamp();                           // Gradual interpolation.
    Vector3.Scale();                                // Component-wise multiplication.
    Vector3.ClampMagnitude();                       // Clamp vector magnitude.
    Vector3.Min();                                  // Component-wise minimum.
    Vector3.Max();                                  // Component-wise maximum.
    Vector3.RotateTowards();                        // Rotate vector toward another.

    // === Vector3 Instance Methods === //
    position.Normalize();                           // Normalize this vector.
    position.ToString();                            // Format vector as string.
    position.Set();                                 // Set vector components.

    // === Vector3 Arithmetic Operations === //
    Vector3.one + Vector3.up;                       // Vector addition.
    Vector3.one - Vector3.up;                       // Vector subtraction.
    Vector3.one* 2;                                 // Scale vector.
    Vector3.one / 2;                                // Divide vector.

    // === Vector3 Comparisons === //
    Vector3.one == Vector3.up;                      // Equality comparison.
    Vector3.one != Vector3.up;                      // Inequality comparison.

    // === Vector3 Utilities === //
    -Vector3.one;                                   // Invert vector.
    (Vector3.one - Vector3.zero).normalized;        // Normalize direction.
    Vector3.Lerp();                                 // Interpolation between vectors.
}
```

UNITY CHEAT-SHEET

PART-26/50

PARTICLE SYSTEM - PART 2

```
public class ParticleSystem2 : MonoBehaviour
{
    // === Color over Lifetime Module === //
    colorOverLifetime.enabled;                  // Color over lifetime enabled.
    colorOverLifetime.color;                    // Color gradient over lifetime.

    // === Size over Lifetime Module === //
    sizeOverLifetime.enabled;                   // Size over lifetime enabled.
    sizeOverLifetime.size;                      // Size curve over lifetime.

    // === Lights Module === //
    lights.enabled;                             // Lights module enabled.
    lights.ratio;                               // Ratio of particles with lights.
    lights.intensity;                           // Intensity of lights.

    // === Custom Data Module === //
    customData.enabled;                         // Custom data enabled.
    customData.SetColor();                      // Set custom color data.
    customData.SetVector();                     // Set custom vector data.

    // === Renderer Module === //
    renderer.renderMode;                        // Rendering mode.
    renderer.material;                          // Material used for rendering.
    renderer.trailMaterial;                     // Material used for trails.
    renderer.mesh;                              // Mesh used for rendering.

    // === Collision Module === //
    collision.enabled;                          // Collision enabled.
    collision.type;                             // Type of collision.
    collision.dampen;                           // Dampen on collision.
    collision.bounce;                           // Bounce on collision.
    collision.lifetimeLoss;                     // Lifetime loss on collision.

    // === Trigger Module === //
    trigger.enabled;                            // Trigger module enabled.
    trigger.SetCollider();                      // Set collider for trigger.
    trigger.GetCollider();                      // Get collider for trigger.

    // === Sub Emitters Module === //
    subEmitters.enabled;                        // Sub emitters enabled.
    subEmitters.AddSubEmitter();                // Add a sub emitter.
    subEmitters.RemoveSubEmitter();             // Remove a sub emitter.

    // === Texture Sheet Animation Module === //
    textureSheetAnimation.enabled;              // Texture sheet animation enabled.
    textureSheetAnimation.numTilesX;            // Number of tiles in X-axis.
    textureSheetAnimation.numTilesY;            // Number of tiles in Y-axis.
    textureSheetAnimation.animation;            // Animation type.
}
```

UNITY CHEAT-SHEET

PART-27/50

ASSET MANAGEMENT

```
public class UnityAssetManagement : MonoBehaviour
{
    // === Resources === //
    Resources.Load();                           // Loads an asset from the Resources folder.
    Resources.LoadAll();                        // Loads all assets from a folder.
    Resources.UnloadAsset();                    // Unloads a specific asset.
    Resources.UnloadUnusedAssets();             // Unloads unused assets from memory.

    // === AssetBundle === //
    AssetBundle.LoadFromFile();                 // Loads an AssetBundle from a file.
    assetBundle.LoadAsset();                    // Loads an asset from the bundle.
    assetBundle.Unload();                       // Unloads the AssetBundle from memory.

    // === Addressables === //
    Addressables.LoadAssetAsync();              // Loads an Addressable asset asynchronously.
    Addressables.InstantiateAsync();            // Instantiates an Addressable asset.
    Addressables.ReleaseInstance();             // Releases an Addressable instance.

    // === AssetDatabase (Editor Only) === //
    AssetDatabase.LoadAssetAtPath();            // Loads an asset using its path.
    AssetDatabase.CreateAsset();                // Creates a new asset file.
    AssetDatabase.SaveAssets();                 // Saves all asset changes.
    AssetDatabase.Refresh();                    // Refreshes the AssetDatabase.

    AssetDatabase.AddObjectToAsset();           // Adds an object to an existing asset.
    AssetDatabase.ImportAsset();                // Imports or reimports an asset.
    AssetDatabase.DeleteAsset();                // Deletes an asset by path.
    AssetDatabase.MoveAsset();                  // Moves an asset to a new location.
    AssetDatabase.FindAssets();                 // Finds assets by type or label.
    AssetDatabase.GetAssetPath();               // Gets the path of a loaded asset.
    AssetDatabase.IsValidFolder();              // Checks if a folder exists.
    AssetDatabase.GetMainAssetTypeAtPath();     // Gets the type of the main asset.
    AssetDatabase.GetDependencies();            // Gets all dependencies for an asset.
    AssetDatabase.Contains();                   // Checks if an asset exists at path.
    AssetDatabase.AssetPathToGUID();            // Converts asset path to GUID.
    AssetDatabase.IsOpenForEdit();              // Checks if asset is open for edit.

    // === AssetImporter === //
    AssetImporter.GetAtPath();                  // Gets the importer for a given path.

    // === TextureImporter === //
    textureImporter.textureType;                // Sets the texture import type.
    textureImporter.mipmapEnabled;              // Enables or disables mipmaps.
    textureImporter.SaveAndReimport();          // Applies and reimports changes.
}
```

UNITY CHEAT-SHEET

PART-28/50

PROFILER - OPTIMIZATION

```
public class UnityProfilerCheatSheet : MonoBehaviour
{
    // === Profiler === //
    Profiler.BeginSample();                          // Starts a custom sample block.
    Profiler.EndSample();                            // Ends the current sample.
    Profiler.GetTotalAllocatedMemoryLong();          // Gets total allocated memory.
    Profiler.GetTotalReservedMemoryLong();           // Gets total reserved memory.
    Profiler.GetTotalUnusedReservedMemoryLong();     // Gets unused reserved memory.
    Profiler.GetMonoHeapSizeLong();                  // Gets Mono heap size.
    Profiler.GetMonoUsedSizeLong();                  // Gets used Mono memory.
    Profiler.GetRuntimeMemorySizeLong();             // Gets runtime memory size of an object.
    Profiler.SetAreaEnabled();                       // Enables profiling for a specific area.
    Profiler.logFile;                                // Path to Profiler log file.
    Profiler.enableBinaryLog;                        // Enables binary logging.
    Profiler.enabled;                                // Enables or disables the Profiler.

    // === CustomSampler === //
    CustomSampler.Create();                          // Creates a custom sample marker.
    customSampler.Begin();                           // Begins a custom sampling section.
    customSampler.End();                             // Ends the custom sampling section.

    // === FrameTimingManager === //
    FrameTimingManager.CaptureFrameTimings();        // Captures current frame timings.
    FrameTimingManager.GetLatestTimings();           // Gets most recent frame timings.

    // === MemoryProfiler === //
    MemoryProfiler.TakeSnapshot();                   // Takes a memory snapshot.
    MemoryProfiler.SnapshotFinished += ();           // Callback when snapshot is complete.

    // === ProfilerDriver === //
    ProfilerDriver.GetAvailableProfilers();          // Lists available profilers.
    ProfilerDriver.connectedProfiler;                // ID of the connected profiler.

    // === Performance Optimization === //
    Profiler.maxUsedMemory;                          // Max memory used by Profiler.
    Profiler.maxNumberOfSamplesPerFrame;             // Max number of samples per frame.
}
```

UNITY CHEAT-SHEET

PART-29/50

POST PROCESSING

```
public class UnityPostProcessing : MonoBehaviour
{
    // === Bloom === //
    bloom.enabled.value;                                    // Enables Bloom effect.
    bloom.intensity.value;                                  // Intensity of Bloom.
    bloom.threshold.value;                                  // Brightness threshold for Bloom.
    bloom.softKnee.value;                                   // Softness of the threshold edge.

    // === ColorGrading === //
    colorGrading.enabled.value;                             // Enables Color Grading.
    colorGrading.postExposure.value;                        // Adjusts scene exposure.
    colorGrading.contrast.value;                            // Adjusts image contrast.
    colorGrading.saturation.value;                          // Adjusts color saturation.
    colorGrading.hueShift.value;                            // Shifts overall hue.

    // === DepthOfField === //
    depthOfField.enabled.value;                             // Enables Depth of Field.
    depthOfField.focusDistance.value;                       // Distance to focal plane.
    depthOfField.aperture.value;                            // Aperture (f-stop).
    depthOfField.focalLength.value;                         // Focal length in mm.

    // === Vignette === //
    vignette.enabled.value;                                 // Enables Vignette effect.
    vignette.intensity.value;                               // Intensity of vignette.
    vignette.smoothness.value;                              // Smoothness of vignette edges.

    // === ChromaticAberration === //
    chromaticAberration.enabled.value;                      // Enables Chromatic Aberration.
    chromaticAberration.intensity.value;                    // Intensity of the aberration.

    // === MotionBlur === //
    motionBlur.enabled.value;                               // Enables Motion Blur.
    motionBlur.shutterAngle.value;                          // Shutter angle in degrees.
    motionBlur.sampleCount.value;                           // Number of blur samples.

    // === LensDistortion === //
    lensDistortion.enabled.value;                           // Enables Lens Distortion.
    lensDistortion.intensity.value;                         // Distortion strength.
    lensDistortion.center.value;                            // Center point of distortion.

    // === AmbientOcclusion === //
    ambientOcclusion.enabled.value;                         // Enables Ambient Occlusion.
    ambientOcclusion.intensity.value;                       // Intensity of occlusion.
    ambientOcclusion.thicknessModifier.value;               // Occlusion thickness modifier.

    // === AutoExposure === //
    autoExposure.enabled.value;                             // Enables Auto Exposure.
    autoExposure.minLuminance.value;                        // Minimum luminance threshold.
    autoExposure.maxLuminance.value;                        // Maximum luminance threshold.

    // === ScreenSpaceReflection === //
    screenSpaceReflection.enabled.value;                    // Enables Screen Space Reflections.
    screenSpaceReflection.maximumMarchDistance.value;       // Max ray distance for reflection.
    screenSpaceReflection.thickness.value;                  // Thickness of reflective surfaces.
}
```

UNITY CHEAT-SHEET

PART-30/50

EDITOR SCRIPTING

```
public class UnityEditorScriptingCheatSheet : MonoBehaviour
{
    // === EditorWindow === //
    EditorWindow.GetWindow();                       // Creates and shows a custom editor window.
    editorWindow.Show();                            // Displays the custom editor window.

    // === EditorGUILayout === //
    EditorGUILayout.Toggle();                       // Adds a toggle field to the inspector.
    EditorGUILayout.IntField();                     // Adds an int field to the inspector.
    EditorGUILayout.FloatField();                   // Adds a float field to the inspector.
    EditorGUILayout.TextField();                    // Adds a text field to the inspector.
    EditorGUILayout.EnumPopup();                    // Adds an enum dropdown in the inspector.
    EditorGUILayout.ColorField();                   // Adds a color field to the inspector.
    EditorGUILayout.Vector3Field();                 // Adds a Vector3 field to the inspector.
    EditorGUILayout.BeginVertical();                // Starts a vertical layout group.
    EditorGUILayout.EndVertical();                  // Ends a vertical layout group.
    EditorGUILayout.BeginHorizontal();              // Starts a horizontal layout group.
    EditorGUILayout.EndHorizontal();                // Ends a horizontal layout group.
    EditorGUILayout.LabelField();                   // Displays a label inside a layout.

    // === EditorApplication === //
    EditorApplication.update += ();                 // Subscribes to the editor update loop.
    EditorApplication.isPlaying;                    // Checks if the editor is in Play mode.
    EditorApplication.isPaused;                     // Checks if the editor is paused.
    EditorApplication.Beep();                       // Plays a system beep.
    EditorApplication.RepaintHierarchyWindow();     // Repaints the hierarchy view.

    // === Handles === //
    Handles.DrawLine();                             // Draws a line in the Scene view.
    Handles.DrawWireDisc();                         // Draws a wire disc in the Scene view.
    Handles.DrawSolidDisc();                        // Draws a solid disc in the Scene view.
    Handles.PositionHandle();                       // Displays a position handle.
    Handles.RotationHandle();                       // Displays a rotation handle.

    // === EditorUtility === //
    EditorUtility.ClearConsole();                   // Clears the Unity console.
    EditorUtility.DisplayDialog();                  // Shows a dialog box.
    EditorUtility.DisplayProgressBar();             // Shows a progress bar.
    EditorUtility.ClearProgressBar();               // Clears the progress bar.
    EditorUtility.PingObject();                     // Pings the object in the Project window.

    // === AssetDatabase === //
    AssetDatabase.CreateAsset();                    // Creates a new asset.
    AssetDatabase.DeleteAsset();                    // Deletes an asset at path.
    AssetDatabase.RenameAsset();                    // Renames an asset.
    AssetDatabase.CopyAsset();                      // Copies an asset to a new path.
    AssetDatabase.MoveAsset();                      // Moves an asset.
    AssetDatabase.LoadAssetAtPath();                // Loads an asset by path.
}
```

UNITY CHEAT-SHEET

PART-31/50

LOCALIZATION SYSTEM

```
public class UnityLocalizationCheatSheet : MonoBehaviour
{
    // === LocalizationSettings === //
    LocalizationSettings.SelectedLocale;                    // Active locale.
    LocalizationSettings.AvailableLocales.GetLocale();      // Retrieves a locale by identifier.
    LocalizationSettings.AvailableLocales.Locales.Count;    // Number of available locales.
    LocalizationSettings.AvailableLocales.GetLocales();     // Gets all available locales.
    LocalizationSettings.StringDatabase.GetTableAsync();    // Loads a string table asynchronously.
    LocalizationSettings.StringDatabase.GetLocalizedString(); // Gets localized string.
    LocalizationSettings.StringDatabase.GetTable();         // Retrieves a string table by name.
    LocalizationSettings.AssetDatabase.GetTableAsync();     // Loads asset table asynchronously.
    LocalizationSettings.AssetDatabase.GetLocalizedAsset(); // Retrieves localized asset.
    LocalizationSettings.InitializationOperation;           // Initialization async operation.
    LocalizationSettings.FallbackLocale;                    // Fallback locale.

    // === LocalizedString === //
    new LocalizedString();                                  // Creates a new LocalizedString instance.
    localizedString.TableReference;                         // Table to fetch string from.
    localizedString.TableEntryReference;                    // Entry key in the string table.
    localizedString.StringChanged += ();                    // Subscribes to updates.
    localizedString.GetLocalizedString();                   // Retrieves localized string synchronously.
    localizedString.GetLocalizedStringAsync();              // Retrieves localized string asynchronously.
    localizedString.PreferredLocale;                        // Preferred locale for the string.
    localizedString.IsEmpty;                                // Checks if the localized string is empty.
    localizedString.Clear();                                // Clears the current localized string.

    // === LocalizedAsset<T> === //
    new LocalizedAsset<Sprite>();                           // Creates a LocalizedAsset for a Sprite.
    localizedSprite.TableReference;                         // Table reference for asset.
    localizedSprite.TableEntryReference;                    // Entry reference for asset.
    localizedSprite.AssetChanged += ();                     // Subscribes to asset updates.
    localizedSprite.GetLocalizedAsset();                    // Retrieves localized asset synchronously.
    localizedSprite.IsEmpty;                                // Checks if the localized asset is empty.
    localizedSprite.Clear();                                // Clears the current localized asset.

    // === LocalizedAudioClip === //
    new LocalizedAudioClip();                               // Creates a new LocalizedAudioClip instance.
    localizedAudioClip.TableReference;                      // Table reference for audio.
    localizedAudioClip.TableEntryReference;                 // Entry reference for audio.
    localizedAudioClip.AssetChanged += ();                  // Subscribes to audio updates.

    // === Locale === //
    LocalizationSettings.SelectedLocale.Identifier.Code;    // Current locale code.
    Locale.Identifier;                                      // Locale identifier.
    Locale.LocaleName;                                      // Locale name.

    // === LocaleManager === //
    LocaleManager.AddLocale();                              // Adds a new locale.
    LocaleManager.RemoveLocale();                           // Removes a locale.
    LocaleManager.SetLocaleFallback();                      // Sets fallback locale.
}
```

UNITY CHEAT-SHEET

PART-32/50

MULTISCENE EDITING

```
public class UnityMultiSceneEditing : MonoBehaviour
{
    // === SceneManager === //
    SceneManager.LoadScene();                          // Loads a scene (single or additive).
    SceneManager.LoadSceneAsync();                     // Loads a scene asynchronously.
    SceneManager.UnloadSceneAsync();                   // Unloads a scene asynchronously.
    SceneManager.MergeScenes();                        // Merges one scene into another.
    SceneManager.MoveGameObjectToScene();              // Moves a GameObject to another scene.
    SceneManager.GetActiveScene();                     // Gets the currently active scene.
    SceneManager.GetSceneAt();                         // Gets the scene at the given index.
    SceneManager.GetSceneByName();                     // Gets a scene by its name.
    SceneManager.SetActiveScene();                     // Sets a specific scene as active.
    SceneManager.sceneCount;                           // Number of currently loaded scenes.
    SceneManager.sceneCountInBuildSettings;            // Number of scenes in build settings.
    SceneManager.sceneLoaded += ();                    // Triggered when a scene is loaded.
    SceneManager.sceneUnloaded += ();                  // Triggered when a scene is unloaded.
    SceneManager.activeSceneChanged += ();             // Triggered when active scene changes.

    // === Scene === //
    scene.IsValid();                                   // Checks if the scene is valid.
    scene.isLoaded;                                    // Checks if the scene is loaded.
    scene.isDirty;                                     // Checks if the scene has unsaved changes.
    scene.buildIndex;                                  // Gets the build index of the scene.
    scene.name;                                        // Gets the name of the scene.
    scene.path;                                        // Gets the path of the scene asset.
    scene.handle;                                      // Gets the internal handle of the scene.
    scene.GetRootGameObjects();                        // Gets all root GameObjects in the scene.
    scene.GetRootGameObjects();                        // Fills a list with root GameObjects.

    // === EditorSceneManager === //
    EditorSceneManager.OpenScene();                    // Opens a scene in the Editor.
    EditorSceneManager.SaveScene();                    // Saves a specific scene.
    EditorSceneManager.SaveOpenScenes();               // Saves all currently open scenes.
    EditorSceneManager.CloseScene();                   // Closes the specified scene.
    EditorSceneManager.NewScene();                     // Creates a new scene.
    EditorSceneManager.MarkSceneDirty();               // Marks a scene as dirty.
    EditorSceneManager.IsSceneOpen();                  // Checks if a scene is open in the editor.
    EditorSceneManager.IsSceneInPlayMode();            // Checks if a scene is part of the play mode setup.
    EditorSceneManager.sceneOpened += ();              // Triggered when a scene is opened in the editor.
    EditorSceneManager.sceneClosed += ();              // Triggered when a scene is closed in the editor.
    EditorSceneManager.sceneDirtied += ();             // Triggered when a scene is modified.
    EditorSceneManager.RestoreSceneManagerSetup();     // Restores the scene setup from stored data.
    EditorSceneManager.GetSceneManagerSetup();         // Gets the current scene setup for multi-scene editing.

    // === Multi-Scene Editing Utilities === //
    EditorSceneManager.PlayPreviewScene();             // Enters play preview mode for a scene.
    EditorSceneManager.CreatePreviewScene();           // Creates a temporary preview scene.
    EditorSceneManager.SetSceneCullingMask();          // Sets the culling mask for a specific scene view.
}
```

UNITY CHEAT-SHEET

PART-33/50

TILEMAPS

```
public class UnityTilemapSystem : MonoBehaviour
{
    // === Tilemap === //
    Tilemap.ResizeBounds();                         // Adjusts the tilemap's size to fit all tiles.
    tilemap.SetTile();                              // Places a tile at a specific position.
    tilemap.SetTileFlags();                         // Sets flags for a specific tile.
    tilemap.SetColor();                             // Changes the color of a specific tile.
    tilemap.GetTile();                              // Retrieves the tile at a given position.
    tilemap.GetTileFlags();                         // Gets flags of the tile at the specified position.
    tilemap.SwapTile();                             // Swaps two types of tiles throughout the tilemap.
    tilemap.ClearAllTiles();                        // Clears all tiles in the tilemap.
    tilemap.cellBounds;                             // Retrieves the tilemap’s bounds.
    tilemap.compressionType;                        // Retrieves the current compression type.

    // === TilemapRenderer === //
    tilemapRenderer.sortOrder;                      // Sets tile rendering order.
    tilemapRenderer.maskInteraction;                // Configures how the tilemap interacts with sprite masks.
    tilemapRenderer.detectChunkCullingBounds;       // Enables chunk culling for performance.
    tilemapRenderer.chunkSize;                      // Defines the size of rendering chunks.
    tilemapRenderer.sortingLayerID;                 // Gets or sets the sorting layer ID for rendering.
    tilemapRenderer.material;                       // Gets or sets the material used for rendering tiles.

    // === TilemapCollider2D === //
    tilemapCollider2D.usedByComposite;              // Enables use with CompositeCollider2D.
    tilemapCollider2D.extrusionFactor;              // Defines collider extrusion depth.
    tilemapCollider2D.isTrigger;                    // Determines if the collider is a trigger.
    tilemapCollider2D.offset;                       // Sets the collider's offset.
    tilemapCollider2D.sharedMaterial;               // Gets or sets the material used by the collider.

    // === Grid === //
    grid.cellSize;                                  // Defines the size of each grid cell.
    grid.cellGap;                                   // Sets the gap between grid cells.
    grid.cellLayout;                                // Sets the grid layout type.
    grid.cellSwizzle;                               // Sets the swizzling order of the grid cells.
    grid.cellRotation;                              // Sets the rotation of the grid cells.

    // === AnimatedTile === //
    animatedTile.m_AnimatedSprites;                 // Assigns animation sprites to animated tile.
    animatedTile.m_MinSpeed;                        // Sets the minimum animation speed for the tile.
    animatedTile.m_MaxSpeed;                        // Sets the maximum animation speed for the tile.
    animatedTile.animationSpeed;                    // Gets or sets the speed of animation.
    animatedTile.loop;                              // Gets or sets if the animation should loop.

    // === Tilemap Queries === //
    tilemap.WorldToCell();                          // Converts world position to tilemap coordinates.
    tilemap.CellToWorld();                          // Converts tilemap coordinates to world position.
    tilemap.GetTileCount();                         // Gets the count of tiles in the map.
    tilemap.GetTileAtPosition();                    // Gets the tile at the specified position.
}
```

UNITY CHEAT-SHEET

PART-34/50

VFX GRAPH

```
public class UnityVFXGraph : MonoBehaviour
{
    // === VisualEffect Control === //
    visualEffect.Play();                        // Starts the VFX effect.
    visualEffect.Stop();                        // Stops the VFX effect.
    visualEffect.Reinit();                      // Reinitializes the VFX component.
    visualEffect.pause;                         // Pauses the VFX effect.
    visualEffect.initialEventName;              // Sets the initial event to trigger on start.
    visualEffect.ResetEffect();                 // Resets and restarts the VFX effect.
    visualEffect.SendEvent();                   // Sends a custom event to trigger VFX.

    // === VisualEffect Property Setters === //
    visualEffect.SetFloat();                    // Sets a float property in VFX.
    visualEffect.SetInt();                      // Sets an integer property in VFX.
    visualEffect.SetUInt();                     // Defines instance count for GPU instancing.
    visualEffect.SetVector3();                  // Sets a Vector3 property in VFX.
    visualEffect.SetVector4();                  // Sets a Vector4 (RGBA) property.
    visualEffect.SetMatrix4x4();                // Sets a matrix property in VFX.
    visualEffect.SetTexture();                  // Assigns a texture property in VFX.
    visualEffect.SetMesh();                     // Assigns a mesh to the VFX graph.
    visualEffect.SetBool();                     // Toggles a boolean property in VFX.

    // === VisualEffect Property Access === //
    visualEffectAsset.HasFloat();               // Checks if a float property exists.
    visualEffect.HasVector3();                  // Checks if a Vector3 property exists.
    visualEffect.HasMatrix4x4();                // Checks if a Matrix property exists.
    visualEffect.HasTexture();                  // Checks if a texture property exists.
    visualEffect.HasMesh();                     // Checks if a mesh property exists.
    visualEffect.GetFloat();                    // Retrieves a float value from VFX Graph.
    visualEffect.GetInt();                      // Retrieves an int value from VFX Graph.
    visualEffect.GetVector3();                  // Retrieves a Vector3 value.
    visualEffect.GetVector4();                  // Retrieves a Vector4 (RGBA) value.

    // === VFX Spawner State === //
    vfxSpawnerState.playing;                    // Starts or stops the spawner.
    vfxSpawnerState.spawnCount;                 // Sets the number of particles spawned.
    vfxSpawnerState.delay;                      // Adds a delay before spawning particles.
    vfxSpawnerState.loopState;                  // Sets the spawner to trigger only once.
}
```

UNITY CHEAT-SHEET

PART-35/50

POST PROCESSING PART 2

```
public class UnityPostProcessingPart2 : MonoBehaviour
{
    // === Film Grain === //
    filmGrain.enabled.value;                        // Enables Film Grain effect.
    filmGrain.type.value;                           // Selects grain texture type.
    filmGrain.intensity.value;                      // Amount of grain.
    filmGrain.response.value;                       // Sensitivity to luminance.

    // === Panini Projection === //
    paniniProjection.enabled.value;                 // Enables Panini Projection.
    paniniProjection.distance.value;                // Controls perspective flattening.
    paniniProjection.cropToFit.value;               // Crops image to preserve framing.

    // === Lift, Gamma, Gain === //
    liftGammaGain.enabled.value;                    // Enables Lift/Gamma/Gain controls.
    liftGammaGain.lift.value;                       // Adjusts shadows.
    liftGammaGain.gamma.value;                      // Adjusts midtones.
    liftGammaGain.gain.value;                       // Adjusts highlights.

    // === Split Toning === //
    splitToning.enabled.value;                      // Enables Split Toning.
    splitToning.shadows.value;                      // Tint color for shadows.
    splitToning.highlights.value;                   // Tint color for highlights.
    splitToning.balance.value;                      // Balance between shadows and highlights.

    // === White Balance === //
    whiteBalance.enabled.value;                     // Enables White Balance.
    whiteBalance.temperature.value;                 // Adjusts color temperature.
    whiteBalance.tint.value;                        // Adjusts color tint (green-magenta).

    // === Tonemapping === //
    tonemapping.enabled.value;                      // Enables Tonemapping.
    tonemapping.mode;                               // Tonemapping algorithm (e.g., ACES, Neutral).

    // === Shadows, Midtones, Highlights === //
    shadowsMidtonesHighlights.enabled.value;        // Enables SMH control.
    shadowsMidtonesHighlights.shadows.value;        // Adjusts shadows color.
    shadowsMidtonesHighlights.midtones.value;       // Adjusts midtones color.
    shadowsMidtonesHighlights.highlights.value;     // Adjusts highlights color.
    shadowsMidtonesHighlights.shadowsStart.value;   // Start threshold for shadows.
    shadowsMidtonesHighlights.highlightsEnd.value;  // End threshold for highlights.

    // === Color Adjustments === //
    colorAdjustments.enabled.value;                 // Enables color adjustment controls.
    colorAdjustments.colorFilter.value;             // Applies a global color filter.
    colorAdjustments.postExposure.value;            // Additional exposure adjustment.

    // === Shadows – URP Specific === //
    screenSpaceShadows.enabled.value;               // Enables screen space shadows (URP only).
    screenSpaceShadows.blend.value;                 // Blending factor for shadows.

    // === Global Volume === //
    volume.weight;                                  // Blending weight of the volume.
    volume.isGlobal;                                // Sets volume to affect whole scene.
    volume.priority;                                // Determines which volume overrides others.
}
```

UNITY CHEAT-SHEET

PART-36/50

NETWORK OPTIMIZATION

```
public class UnityNetworkingOptimization : MonoBehaviour
{
    // === NetworkManager === //
    networkManager.NetworkConfig.TickRate;                // Sets the network tick rate to balance performance.
    networkManager.NetworkConfig.ConnectionApproval;      // Enables connection approval process.
    networkManager.ShutdownInProgress();                  // Checks if the network is shutting down.
    networkManager.LogLevel;                              // Enables detailed logging for debugging.
    networkManager.DontDestroy;                           // Prevents network manager from being destroyed.

    // === NetworkTime === //
    networkManager.NetworkTime.ServerTime;                // Retrieves the current server time.
    networkManager.NetworkTime.Time;                      // Retrieves the local client time.
    networkManager.NetworkConfig.TimeTicksPerSecond;      // Adjusts tick rate for better network synchronization.

    // === NetworkTransform === //
    networkObject.NetworkTransform.Interpolate;           // Enables interpolation for smooth movement.
    networkObject.NetworkTransform.TeleportThreshold;     // Defines teleport threshold to correct lag spikes.
    networkObject.NetworkTransform.InLocalSpace;          // Ensures transformation updates occur in local space.

    // === Network Bandwidth === //
    networkManager.EnableNetworkVariableCompression;      // Enables compression to reduce bandwidth usage.
    networkManager.EnableSceneManagement;                 // Disables automatic scene synchronization.
    networkManager.MaxReceiveEventsPerTickRate;           // Limits incoming network events per tick.
    networkManager.MaxSendQueuesPerTickRate;              // Limits outgoing packets per tick.

    // === Messaging === //
    networkManager.NetworkConfig.EnsureMessageOrder;      // Disables strict message ordering to reduce lag.
    networkManager.NetworkConfig.MaxConnections;          // Limits the number of active network connections.
    networkManager.NetworkConfig.ReceiveTimeoutMS;        // Sets the timeout for network disconnection handling.

    // === Interpolation & Sync === //
    networkObject.NetworkTransform.InterpolationDelay;    // Adds interpolation delay for smoother movement sync.
    networkObject.NetworkTransform.SynchronizePhysics;    // Ensures physics calculations remain in sync.

    // === Network Callbacks === //
    networkManager.OnClientDisconnectCallback;            // Attempts to reconnect automatically after disconnect.
    networkManager.OnClientConnectedCallback;             // Handles successful client connection.
    networkManager.OnTransportFailure;                    // Listens for transport failures and responds.

    // === Transport Layer Optimization === //
    networkManager.NetworkConfig.NetworkTransport;        // Sets custom transport for performance tuning.
    networkManager.NetworkConfig.ProtocolVersion;         // Ensures clients use compatible versions.

    // === Connection Handling === //
    networkManager.ConnectionApprovalCallback;            // Custom logic for connection approval.
    networkManager.OnServerStarted;                       // Triggered when the server has started.

    // === Custom Tick Settings === //
    networkManager.NetworkConfig.ClientBufferTimeout;     // Timeout before dropping slow clients.
    networkManager.NetworkConfig.SecondsHistory;          // Time buffer for network history.
}
```

UNITY CHEAT-SHEET

PART-37/50

BUILD PIPELINE

Problem: To automate key player settings

Solution:

```
PlayerSettings.productName = "My Game";
PlayerSettings.companyName = "My Company";
PlayerSettings.bundleVersion = "1.0.0";
PlayerSettings.fullScreenMode =
    FullScreenMode.FullScreenWindow;
PlayerSettings.runInBackground = true;
```

Problem: To include all scenes from build settings

Solution:

```
buildOptions.scenes = EditorBuildSettings.scenes
    .Where(scene => scene.enabled)
    .Select(scene => scene.path)
    .ToArray();
```

Problem: To configure the build type

Solution:

```
EditorUserBuildSettings.development = false;
EditorUserBuildSettings.allowDebugging = false;
```

Problem: To execute the build process

Solution:

```
BuildPipeline.BuildPlayer(buildOptions);
```

Problem: To log detailed Build Statistics

Solution:

```
BuildReport report = BuildPipeline.BuildPlayer(buildOptions);
BuildSummary summary = report.summary;
Debug.Log($"Build Result: {summary.result}");
```

Problem: To configure android specific build settings

Solution:

```
PlayerSettings.Android.bundleVersionCode += 1;
buildOptions.target = BuildTarget.Android;
buildOptions.options = BuildOptions.CompressWithLz4;
EditorUserBuildSettings.buildAppBundle = true;
```

UNITY CHEAT-SHEET

PART-38/50

AI NPC

Problem: To calculate and set a navigation path

Solution:

```
if (NavMesh.SamplePosition(target.position, out hit, 2.0f, NavMesh.AllAreas))
{
    bool pathSuccess = NavMesh.CalculatePath(navMeshAgent.transform.position,
                                             hit.position, NavMesh.AllAreas, navMeshPath);
    if (pathSuccess)
        navMeshAgent.SetPath(navMeshPath);
}
```

Problem: To start the movement along the calculated path

Solution:

```
navMeshAgent.isStopped = false;
navMeshAgent.speed = 3.5f;
navMeshAgent.acceleration = 8f;
navMeshAgent.angularSpeed = 120f;
navMeshAgent.stoppingDistance = 1.5f;
```

Problem: To navigate through Gaps & Ledges

Solution:

```
navMeshAgent.autoTraverseOffMeshLink = true;
if (navMeshAgent.isOnOffMeshLink)
    navMeshAgent.CompleteOffMeshLink();
```

Problem: To setup obstacle avoidance

Solution:

```
navMeshObstacle.carving = true;
navMeshObstacle.carvingMoveThreshold = 0.1f;
navMeshObstacle.carvingTimeToStationary = 0.5f;
navMeshObstacle.carveOnlyStationary = false;
```

Problem: To visualize the navigation path

Solution:

```
Debug.DrawLine(navMeshAgent.transform.position,
               navMeshAgent.pathEndPosition, Color.red);
for (int i = 0; i < navMeshAgent.path.corners.Length - 1; i++)
    Debug.DrawLine(navMeshAgent.path.corners[i],
                   navMeshAgent.path.corners[i + 1], Color.green);
```

UNITY CHEAT-SHEET

PART-39/50

ADS: REWARDED VIDEO ADS

Problem: To initialize the unity ads sdk with platform detection

Solution:

```
string gameId;
#if UNITY_IOS
    gameId = "YOUR_IOS_GAME_ID";
#elif UNITY_ANDROID
    gameId = "YOUR_ANDROID_GAME_ID";
#else
    gameId = null;
    Debug.LogError("Unsupported platform for Unity Ads");
    return;
#endif

bool testMode = true;
Advertisement.Initialize(gameId, testMode);
```

Problem: To load a rewarded video ad

Solution:

```
// Must be called from a class implementing IUnityAdsLoadListener
public void LoadRewardedAd() => Advertisement.Load("rewardedVideo", this);
```

Problem: To implement ads load success and failure callbacks

Solution:

```
// Required callbacks for IUnityAdsLoadListener
public void OnUnityAdsAdLoaded(string placementId) =>
    Debug.Log($"Ad loaded: {placementId}"); // Enable ad button

public void OnUnityAdsFailedToLoad(string placementId,
            UnityAdsLoadError error, string message) =>
    Debug.LogError($"Ad load failed: {error}"); // Handle failure
```

Problem: To display a rewarded video ad

Solution:

```
public void ShowRewardedAd()
{
    if (Advertisement.IsReady("rewardedVideo"))
    {
        Advertisement.Show("rewardedVideo", new ShowOptions
        {
            resultCallback = HandleAdResult
        });
    }
}
```

Problem: To handle rewarded ad completion

Solution:

```
void HandleAdResult(ShowResult result)
{
    switch (result)
    {
        case ShowResult.Finished:
            Debug.Log("Player watched the full ad - Reward them!");
            break;
        case ShowResult.Skipped:
            Debug.Log("Ad was skipped - No reward!");
            break;
        case ShowResult.Failed:
            Debug.LogError("Ad failed to play!");
            break;
    }
}
```

UNITY CHEAT-SHEET

PART-40/50

MATHEMATICS

Problem: Calculating Angles and Rotation

Solution:

```
Mathf.Sin(Mathf.PI / 2);   // Computes the sine of an angle (radians).
Mathf.Cos(0);              // Computes the cosine of an angle.
Mathf.Tan(Mathf.PI / 4);   // Computes the tangent of an angle.
Mathf.Asin(0.5f);          // Returns the arcsine (inverse sine).
Mathf.Acos(0.5f);          // Returns the arccosine (inverse cosine).
Mathf.Atan(1f);            // Returns the arctangent (inverse tangent).
Mathf.Atan2(1f, 2f);       // Returns the angle between x and y
```

Problem: Rounding, Scaling, and Logarithmic Computations

Solution:

```
Mathf.Round(4.7f);
Mathf.Ceil(4.2f);
Mathf.Floor(4.8f);
Mathf.Pow(2, 3);
Mathf.Sqrt(16f);
Mathf.Log(10f);
Mathf.Log10(100f);
Mathf.Sign(-5f);
```

Problem: Performing Vector and Matrix Calculations

Solution:

```
float3 a = new float3(1f, 2f, 3f);
float3 b = new float3(4f, 5f, 6f);
math.dot(a, b);
math.cross(a, b);
math.normalize(a);
math.length(a);
math.distance(a, b);

float4x4 matrix = math.mul(math.float4x4.identity,
    math.float4x4.identity);
float4 transformedVector = math.mul(matrix,
    new float4(1f, 2f, 3f, 1f));
```

Special Math Functions

```
Mathf.Abs(-10f);                  // Returns absolute value.
Mathf.Min(3.5f, 7.2f);            // Returns the smaller of two values.
Mathf.Max(3.5f, 7.2f);            // Returns the larger of two values.
Mathf.Clamp(15f, 0f, 10f);        // Clamps a value within a specified range.
Mathf.Clamp01(1.5f);              // Clamps a value between 0 and 1.
Mathf.Lerp(0f, 10f, 0.5f);        // Linearly interpolates between two values.
Mathf.MoveTowards(5f, 10f, 2f);   // Moves a value towards a target.
```

UNITY CHEAT-SHEET

PART-41/50

TAGS & LAYERS

Problem: Assigning and Checking Tags

Solution:

```
gameObject.tag = "Player";
bool isTagged = gameObject.CompareTag("Enemy");
```

Problem: Finding Objects by Tag

Solution:

```
GameObject foundObject = GameObject.FindWithTag("Player");
GameObject[] enemies = GameObject.FindGameObjectsWithTag("Enemy");
```

Problem: Ignoring Collisions Between Specific Layers

Solution:

```
public const int ENEMY_LAYER = 8;
Physics.IgnoreLayerCollision(ENEMY_LAYER, ENEMY_LAYER, true);
bool collisionState = Physics.GetIgnoreLayerCollision(ENEMY_LAYER, ENEMY_LAYER);
```

Problem: Controlling Camera Rendering with Layers

Solution:

```
// Makes the camera render only specified layers
Camera.main.cullingMask = LayerMask.GetMask("Default", "UI");
// Excludes specific layers from rendering
Camera.main.cullingMask &= ~LayerMask.GetMask("UI");
```

Problem: Finding Objects by Layer in a Specific Area

Solution:

```
// Creates a mask for enemy layer
int enemyLayerMask = 1 << ENEMY_LAYER;

// Gets all colliders within a sphere that match specific layers
Collider[] nearbyEnemies = Physics.OverlapSphere(
    transform.position,
    10f, // Radius
    enemyLayerMask
);
```

UNITY CHEAT-SHEET

PART-42/50

PHYSICS PART 3

Problem: Applying forces to objects

Solution:

```
Rigidbody rb = GetComponent<Rigidbody>();
rb.AddForce(Vector3.forward * 10f, ForceMode.Force);
rb.AddForce(Vector3.up * 5f, ForceMode.Impulse);
rb.AddTorque(Vector3.up * 5f, ForceMode.Impulse);
rb.AddExplosionForce(10f, explosionCenter, 5f);
```

Problem: Creating bouncy surfaces

Solution:

```
PhysicMaterial bouncyMaterial = new PhysicMaterial("Bouncy");
bouncyMaterial.bounciness = 0.8f;
bouncyMaterial.frictionCombine = PhysicMaterialCombine.Average;
bouncyMaterial.bounceCombine = PhysicMaterialCombine.Maximum;
GetComponent<Collider>().material = bouncyMaterial;
```

Problem: Detecting objects with raycasts

Solution:

```
Ray ray = new Ray(transform.position, transform.forward);
Physics.Raycast(ray, out RaycastHit rayHit, 100f);
```

Problem: Finding objects in an area

Solution:

```
Collider[] colliders = Physics.OverlapSphere(transform.position, 5f); // Radius check
foreach (Collider col in colliders) {
    float distance = Vector3.Distance(transform.position, col.transform.position);
    if (col.CompareTag("Enemy")) {
        // Process each found enemy
    }
}

// Box-shaped check
Collider[] boxResults = Physics.OverlapBox(
    center: transform.position,
    halfExtents: new Vector3(2f, 1f, 2f),
    orientation: transform.rotation
);
```

UNITY CHEAT-SHEET

PART-43/50

PROJECT SETTINGS

```
public class ProjectSettings : MonoBehaviour
{
    // ═══ PlayerSettings ═══ //
    PlayerSettings.GetScriptingBackend();                // Gets scripting backend.
    PlayerSettings.SetScriptingBackend();                // Sets scripting backend.
    PlayerSettings.GetArchitecture();                    // Gets platform architecture.
    PlayerSettings.SetArchitecture();                    // Sets platform architecture.
    PlayerSettings.stripEngineCode;                      // Strips unused engine code.
    PlayerSettings.Android.forceInternetPermission;      // Forces internet permission on Android.
    PlayerSettings.Android.useAPKExpansionFiles;         // Enables APK expansion files.
    PlayerSettings.iOS.requiresPersistentWiFi;           // Requires persistent WiFi on iOS.
    PlayerSettings.iOS.allowHTTPDownload;                // Allows/disallows HTTP downloads on iOS.

    // ═══ Physics & Simulation ═══ //
    Physics.gravity;                                     // Sets global gravity.
    Physics.defaultSolverIterations;                     // Controls solver accuracy.
    Physics.defaultSolverVelocityIterations;             // Controls velocity solver accuracy.
    Physics.IgnoreLayerCollision();                      // Ignores collisions between layers.
    Physics.queriesHitTriggers;                          // Toggles raycast detection on triggers.
    Physics.autoSimulation;                              // Enables/disables automatic physics updates.
    Physics.defaultContactOffset;                        // Sets default collision offset.
    Physics.bounceThreshold;                             // Defines velocity threshold for bouncing.
    Physics.sleepThreshold;                              // Defines velocity threshold for sleep mode.
    Physics.reuseCollisionCallbacks;                     // Optimizes physics callbacks.

    // ═══ Application ═══ //
    Application.version;                                 // Gets app version.
    Application.productName;                             // Gets product name.
    Application.companyName;                             // Gets company name.
    Application.unityVersion;                            // Gets Unity version.
    Application.platform;                                // Gets current platform.
    Application.targetFrameRate;                         // Sets target frame rate.
    Application.internetReachability;                    // Checks internet status.
    Application.RequestUserAuthorization();              // Requests user permissions.
    Application.SetStackTraceLogType();                  // Configures stack trace logging.

    // ═══ QualitySettings (Performance) ═══ //
    QualitySettings.vSyncCount;                          // Toggles VSync.
    QualitySettings.antiAliasing;                        // Sets anti-aliasing level.
    QualitySettings.shadowDistance;                      // Sets max shadow distance.
    QualitySettings.lodBias;                             // Adjusts LOD scaling.
    QualitySettings.masterTextureLimit;                  // Sets texture quality.
    QualitySettings.realtimeReflectionProbes;            // Toggles real-time reflections.
    QualitySettings.pixelLightCount;                     // Sets max pixel lights.
    QualitySettings.softParticles;                       // Toggles soft particles.

    // ═══ AudioSettings (Audio & DSP) ═══ //
    AudioSettings.outputSampleRate;                      // Gets audio sample rate.
    AudioSettings.dspTime;                               // Gets DSP time.
    AudioSettings.GetConfiguration();                    // Gets audio config.
    AudioSettings.GetSpatializerPluginName();            // Gets spatializer plugin.
    AudioSettings.SetSpatializerPluginName();            // Sets spatializer plugin.
    AudioSettings.speakerMode;                           // Gets/sets speaker mode.

    // ═══ Input System ═══ //
    Input.simulateMouseWithTouches;                      // Simulates mouse with touch.
    Input.multiTouchEnabled;                             // Enables multi-touch.
    Input.touchPressureSupported;                        // Checks touch pressure support.
}
```

UNITY CHEAT-SHEET

PART-44/50

PROBUILDER: MESH MODELING

```
public class ProBuilder : MonoBehaviour
{
    // == ProBuilderMesh == //
    ProBuilderMesh.Create();                    // Creates a new ProBuilder mesh instance.
    pbMesh.ToMesh();                            // Applies ProBuilder changes to the Unity Mesh.
    pbMesh.Refresh();                           // Refreshes mesh attributes like UVs and normals.
    pbMesh.Rebuild();                           // Rebuilds mesh topology.
    pbMesh.Extrude();                           // Extrudes selected faces along their normals.
    pbMesh.SubdivideFaces();                    // Subdivides selected faces into smaller faces.
    pbMesh.DeleteVertices();                    // Deletes vertices by their indices.
    pbMesh.MergeVertices();                     // Merges vertices that share the same position.
    pbMesh.FlipFaces();                         // Flips the normals of the mesh.

    // == ShapeGenerator == //
    ShapeGenerator.CreateShape();               // Generates a specified shape (e.g., Cube, Sphere).
    ShapeGenerator.GeneratePlane();             // Generates a plane with specified parameters.
    ShapeGenerator.GenerateStair();             // Generates a stair mesh with specified parameters.
    ShapeGenerator.GenerateArch();              // Generates an arch mesh with specified parameters.
    ShapeGenerator.GenerateDoor();              // Generates a door mesh with specified parameters.
    ShapeGenerator.GeneratePipe();              // Generates a pipe mesh with specified parameters.
    ShapeGenerator.GenerateCone();              // Generates a cone mesh with specified parameters.

    // == AutoUnwrapSettings == //
    AutoUnwrapSettings.useWorldSpace;           // Determines if UVs are in world space.
    AutoUnwrapSettings.flipU;                   // Flips UVs horizontally.
    AutoUnwrapSettings.flipV;                   // Flips UVs vertically.
    AutoUnwrapSettings.fill;                    // Sets the fill mode for UVs.
    AutoUnwrapSettings.anchor;                  // Sets the anchor point for UVs.
    AutoUnwrapSettings.scale;                   // Sets the scale for UVs.
    AutoUnwrapSettings.offset;                  // Sets the offset for UVs.

    // == BuiltinMaterials == //
    BuiltinMaterials.defaultMaterial;           // Accesses ProBuilder's default material.
    BuiltinMaterials.facePickerMaterial;        // Material used for face picking.
    BuiltinMaterials.vertexPickerMaterial;      // Material used for vertex picking.
    BuiltinMaterials.edgePickerMaterial;        // Material used for edge picking.
    BuiltinMaterials.triggerMaterial;           // Material used for triggers.
    BuiltinMaterials.colliderMaterial;          // Material used for colliders.
    BuiltinMaterials.noDrawMaterial;            // Material that is not drawn.

    // == Normals == //
    Normals.CalculateNormals();                 // Recalculates normals for the mesh.
    Normals.CalculateTangents();                // Recalculates tangents for the mesh.

    // == Projection == //
    Projection.PlanarProject();                 // Applies planar UV projection to faces.
    Projection.BoxProject();                    // Applies box UV projection to faces.

    // == MeshUtility == //
    MeshUtility.Optimize();                     // Optimizes the mesh for performance.
    MeshUtility.CollapseSharedVertices();       // Collapses vertices that share the same position.

    // == HandleUtility == //
    HandleUtility.FaceRaycast();                // Raycasts to detect a face under the cursor.
    HandleUtility.PickVertices();               // Picks vertices from a raycast.
}
```

UNITY CHEAT-SHEET

PART-45/50

DOTS

```
public class UnityDOTS : MonoBehaviour
{
    // === World === //
    World.DefaultGameObjectInjectionWorld;  // The default World instance.
    World.CreateSystem();                   // Creates a new system.
    World.GetOrCreateSystem();              // Retrieves or creates a system.
    World.DestroySystem();                  // Destroys a system.

    // === EntityManager === //
    entityManager.CreateEntity();           // Creates a new Entity.
    entityManager.DestroyEntity();          // Destroys an Entity.
    entityManager.AddComponent();           // Adds a component to an Entity.
    entityManager.RemoveComponent();        // Removes a component from an Entity.
    entityManager.HasComponent();           // Checks if an Entity has a component.
    entityManager.GetComponentData();       // Gets component data from an Entity.
    entityManager.SetComponentData();       // Sets component data for an Entity.

    // === EntityQuery === //
    entityQuery.CalculateEntityCount();     // Gets the number of entities matching the query.
    entityQuery.ToEntityArray();            // Retrieves an array of entities from the query.
    entityQuery.SetChangedVersionFilter();  // Filters entities with changed component data.

    // === SystemBase === //
    systemBase.Enabled;                     // Checks if the system is enabled.
    systemBase.OnCreate();                  // Called when the system is created.
    systemBase.OnUpdate();                  // Called on each frame update.
    systemBase.OnDestroy();                 // Called when the system is destroyed.

    // === JobHandle === //
    jobHandle.Complete();                   // Ensures the job has completed.
    jobHandle.IsCompleted;                  // Checks if the job is completed.

    // === BurstCompiler === //
    BurstCompiler.CompileFunctionPointer()  // Compiles a function for Burst.
    BurstCompiler.IsEnabled;                // Checks if Burst is enabled.

    // === NativeArray === //
    nativeArray.Length;                     // Gets the length of the native array.
    nativeArray.Dispose();                  // Disposes of the native array.
    nativeArray.IsCreated;                  // Checks if the native array is created.

    // === NativeQueue === //
    nativeQueue.Enqueue();                  // Adds an item to the native queue.
    nativeQueue.Dequeue();                  // Removes an item from the native queue.
    nativeQueue.IsEmpty;                    // Checks if the native queue is empty.
}
```

UNITY CHEAT-SHEET

PART-46/50

XR: PART 1

```
public class UnityXR : MonoBehaviour
{
    // === XRGeneralSettings === //
    XRGeneralSettings.Instance.Manager.activeLoader;        // Gets the active XR loader.
    XRGeneralSettings.Instance.Manager.activeLoader.name;   // Retrieves the current XR device name.

    // === XRDisplaySubsystem === //
    xrDisplaySubsystem.running;                             // Checks if XR is currently active.
    xrDisplaySubsystem.eyeTextureWidth;                     // Gets the width of the XR eye texture.
    xrDisplaySubsystem.eyeTextureHeight;                    // Gets the height of the XR eye texture.
    xrDisplaySubsystem.scaleOfAllViewports;                 // Adjusts the render resolution scale.
    xrDisplaySubsystem.renderViewportScale;                 // Adjusts the XR viewport scaling.
    xrDisplaySubsystem.stereoRenderingMode;                 // Defines the stereo rendering method.
    xrDisplaySubsystem.TryGetAppGPUTimeLastFrame();         // Retrieves GPU time for the last frame.
    xrDisplaySubsystem.TryGetDisplayRefreshRate();          // Gets display refresh rate.
    xrDisplaySubsystem.TryGetRenderPass();                  // Retrieves active render passes.
    xrDisplaySubsystem.TryGetFramePresentCount();           // Gets the number of presented frames.

    // === XRInputSubsystem === //
    xrInputSubsystem.TryRecenter();                         // Resets tracking origin.
    xrInputSubsystem.TrySetTrackingOriginMode();            // Sets tracking origin mode.
    xrInputSubsystem.GetTrackingOriginMode();               // Gets the current tracking origin mode.
    xrInputSubsystem.GetSupportedTrackingOriginModes();     // Lists available tracking origins.
    xrInputSubsystem.TryGetBoundaryPoints();                // Gets boundary points of the play area.
    xrInputSubsystem.trackingOriginUpdated;                 // Event when tracking origin is updated.
    xrInputSubsystem.boundaryChanged;                       // Event when boundary changes.

    // === InputDevices === //
    InputDevices.GetDevicesWithCharacteristics();           // Gets XR input devices by characteristics.
    InputDevices.GetDeviceAtXRNode();                       // Gets a specific XR device.
    InputDevices.deviceConnected;                           // Event when an input device connects.
    InputDevices.deviceDisconnected;                        // Event when an input device disconnects.
    InputDevices.deviceConfigChanged;                       // Event when an input device config changes.
    InputDevices.GetDevices();                              // Retrieves all connected XR devices.
    InputDevices.SendHapticImpulse();                       // Sends haptic feedback.
    InputDevices.TryGetFeatureValue();                      // Retrieves input feature values.

    // === XRStats === //
    XRStats.TryGetGPUTimeLastFrame();                       // Gets GPU time for the last frame.
    XRStats.TryGetFramePresentCount();                      // Retrieves number of presented frames.
    XRStats.TryGetDroppedFrameCount();                      // Retrieves number of dropped frames.
}
```

UNITY CHEAT-SHEET

PART-47/50

XR: PART 2

```
public class UnityXRPart2 : MonoBehaviour
{
    // === ARTrackedImage === //
    arTrackedImage.referenceImage;                  // Gets detected reference image.
    arTrackedImage.size;                            // Gets detected image size.
    arTrackedImage.trackingState;                   // Gets tracking state of the image.
    arTrackedImage.transform;                       // Gets transform of detected image.

    // === ARFaceManager === //
    arFaceManager.enabled;                          // Enables/disables face tracking.
    arFaceManager.trackables;                       // Lists detected faces.
    arFaceManager.facePrefab;                       // Sets prefab for tracked faces.
    arFaceManager.maximumFaceCount;                 // Limits simultaneous face tracking.
    arFaceManager.supportedFaceCount;               // Gets max supported face tracking.

    // === ARFace === //
    arFace.vertices;                                // Gets face mesh vertices.
    arFace.normals;                                 // Gets face mesh normals.
    arFace.indices;                                 // Gets face mesh indices.
    arFace.uvs;                                     // Gets face mesh UVs.
    arFace.leftEye;                                 // Gets left eye transform.
    arFace.rightEye;                                // Gets right eye transform.

    // === ARSession === //
    ARSession.Reset();                              // Resets the AR session.
    arSession.enabled;                              // Enables/disables AR tracking.
    ARSession.state;                                // Gets the current AR session state.
    ARSession.notTrackingReason;                    // Gets the reason for lost tracking.
    arSession.matchFrameRate;                       // Matches AR frame rate to display.

    // === ARPlaneManager === //
    arPlaneManager.enabled;                         // Enables/disables plane detection.
    arPlaneManager.requestedDetectionMode;          // Sets plane detection mode.
    arPlaneManager.trackables;                      // Lists detected planes.
    arPlaneManager.planePrefab;                     // Sets the plane prefab.
    arPlaneManager.currentDetectionMode;            // Gets active plane detection mode.

    // === ARPlane === //
    arPlane.boundary;                               // Gets plane boundary points.
    arPlane.alignment;                              // Gets plane alignment (horizontal/vertical).
    arPlane.center;                                 // Gets plane center position.
    arPlane.extents;                                // Gets plane extents (half-size).
    arPlane.size;                                   // Gets plane width and height.

    // === ARTrackedImageManager === //
    arTrackedImageManager.referenceLibrary;         // Sets tracked image library.
    arTrackedImageManager.enabled;                  // Enables/disables image tracking.
    arTrackedImageManager.trackables;               // Lists detected images.
    arTrackedImageManager.maxNumberOfMovingImages;  // Limits tracked moving images.
    arTrackedImageManager.trackedImagePrefab;       // Sets prefab for tracked images.
}
```

UNITY CHEAT-SHEET

PART-48/50

MULTIPLAYER NETCODE: PART - 2

```
public class UnityNetcode : MonoBehaviour
{
    // === NetworkManager === //
    NetworkManager.Singleton.StartHost();                  // Starts the server and client (host mode).
    NetworkManager.Singleton.StartServer();                // Starts the server.
    NetworkManager.Singleton.StartClient();                // Starts the client and connects to a server.
    NetworkManager.Singleton.Shutdown();                   // Shuts down the network session.
    NetworkManager.Singleton.ConnectedClients;             // Retrieves connected clients.
    NetworkManager.Singleton.IsHost;                       // Checks if running as host.
    NetworkManager.Singleton.IsServer;                     // Checks if running as server.
    NetworkManager.Singleton.IsClient;                     // Checks if running as client.
    NetworkManager.Singleton.ConnectionApprovalCallback; // Sets the approval callback for connections.
    NetworkManager.Singleton.OnClientConnectedCallback;  // Event triggered when a client connects.
    NetworkManager.Singleton.OnClientDisconnectCallback; // Event triggered when a client disconnects.

    // === NetworkBehaviour === //
    networkBehaviour.IsOwner;                              // Checks if this client owns the object.
    networkBehaviour.IsServer;                             // Checks if the object is on the server.
    networkBehaviour.NetworkObject;                        // Retrieves the NetworkObject component.
    networkBehaviour.NetworkManager;                       // Accesses the NetworkManager instance.
    networkBehaviour.OwnerClientId;                        // Gets the client ID of the object owner.
    networkBehaviour.NetworkVariable;                      // Networked variable for syncing state.

    // === NetworkObject === //
    networkObject.Spawn();                                 // Spawns the network object.
    networkObject.Despawn();                               // Despawns the network object.
    networkObject.IsSpawned;                               // Checks if the object is spawned.
    networkObject.ChangeOwnership();                       // Changes object ownership to a client.
    networkObject.IsOwner;                                 // Checks if this client owns the object.

    // === RPCs (Remote Procedure Calls) === //
    networkBehaviour.SendServerRpc();                      // Sends an RPC from client to server.
    networkBehaviour.SendClientRpc();                      // Sends an RPC from server to all clients.
    networkBehaviour.SendTargetRpc();                      // Sends an RPC to a specific client.

    // === NetworkVariable === //
    networkVariable.Value;                                 // Gets or sets the value of the variable.
    networkVariable.OnValueChanged;                        // Event triggered when value changes.
    networkVariable.Settings;                              // Configures the variable settings.
    networkVariable.ReadPerm;                              // Defines read permissions.
    networkVariable.WritePerm;                             // Defines write permissions.

    // === Scene Management === //
    NetworkManager.Singleton.SceneManager.LoadScene();   // Loads a scene across the network.
    NetworkManager.Singleton.SceneManager.OnSceneEvent;  // Event triggered for network scene events.
}
```

UNITY CHEAT-SHEET

PART-49/50

RECORDER

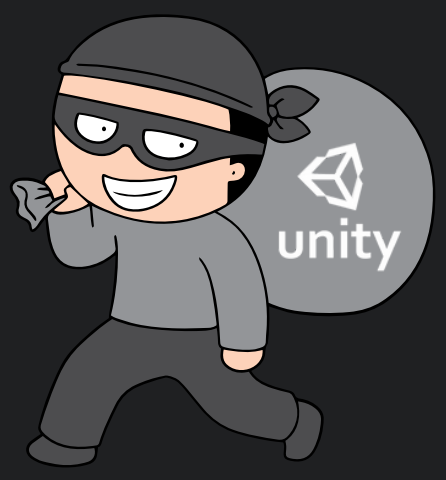
```
public class UnityRecorder : MonoBehaviour
{
    // === RecorderController === //
    recorderController.StartRecording();                    // Starts recording.
    recorderController.StopRecording();                     // Stops recording.
    recorderController.IsRecording();                       // Checks if recording is in progress.
    recorderController.PrepareRecording();                  // Prepares the recorder before starting.

    // === RecorderControllerSettings === //
    recorderControllerSettings.AddRecorderSettings();   // Adds a recorder to the session.
    recorderControllerSettings.SetRecordModeToManual(); // Enables manual control over recording.

    // === RecorderSettings === //
    recorderSettings.FrameRate;                             // Sets the recording frame rate.
    recorderSettings.OutputFile;                            // Specifies the output file path.
    recorderSettings.CaptureEveryNthFrame;                  // Defines frame capture interval.
    recorderSettings.ImageWidth;                            // Sets output image width.
    recorderSettings.ImageHeight;                           // Sets output image height.

    // === MovieRecorderSettings === //
    movieRecorderSettings.OutputFormat;                     // Defines the video format (MP4, WebM, etc.).
    movieRecorderSettings.VideoBitrateMode;                 // Controls video bitrate settings.
    movieRecorderSettings.AudioInputSettings;               // Configures audio recording options.

    // === ImageRecorderSettings === //
    imageRecorderSettings.OutputFormat;                     // Sets image format (PNG, JPEG, EXR, etc.).
    imageRecorderSettings.CaptureAlpha;                     // Enables alpha channel capture.
    imageRecorderSettings.ColorSpace;                       // Sets the color space (sRGB, Linear).

    // === GifRecorderSettings === //
    gifRecorderSettings.FrameRate;                          // Sets the GIF frame rate.
    gifRecorderSettings.Quality;                            // Defines GIF compression quality.

    // === AudioRecorderSettings === //
    audioRecorderSettings.OutputFormat;                     // Defines audio format (WAV, MP3, etc.).
    audioRecorderSettings.AudioInputSettings;               // Configures microphone or in-game audio input.

    // === Recorder Inputs === //
    gameViewInputSettings.CaptureGameView();                // Captures Game View for recording.
    cameraInputSettings.CaptureCamera();                    // Captures a specific camera's output.
    renderTextureInputSettings.CaptureTexture();            // Captures RenderTexture for recording.

    // === Options === //
    Options.showLegacyRecorders;                            // Hides or shows legacy recorders.
    Options.showRecorderGameObject;                         // Displays the Recorder GameObject in Hierarchy.
    Options.verboseMode;                                    // Enables verbose logging for debugging.
}
```

UNITY CHEAT-SHEET

PART-50/50

ANIMATION RIGGING

```
public class UnityAnimationRigging : MonoBehaviour
{
    // === RigBuilder === //
    rigBuilder.layers;                                  // Retrieves the layers in the RigBuilder.
    rigBuilder.Build();                                 // Builds the rig at runtime.
    rigBuilder.Clear();                                 // Clears the rig data.
    rigBuilder.enabled;                                 // Enables or disables RigBuilder.

    // === Rig === //
    rig.weight;                                         // Controls the overall weight of the rig.
    rig.layers;                                         // Retrieves layers applied to the rig.
    rig.SetWeight();                                    // Dynamically adjusts rig weight.

    // === TwoBoneIKConstraint === //
    twoBoneIK.target;                                   // Sets the target Transform for IK.
    twoBoneIK.hint;                                     // Sets the hint Transform for guiding IK.
    twoBoneIK.weight;                                   // Adjusts the overall influence of IK.
    twoBoneIK.targetPositionWeight;                     // Sets the influence of the target position.
    twoBoneIK.targetRotationWeight;                     // Sets the influence of the target rotation.
    twoBoneIK.hintWeight;                               // Controls how much the hint affects IK.

    // === MultiAimConstraint === //
    multiAimConstraint.data.constrainedObject;          // The object being constrained.
    multiAimConstraint.data.sourceObjects;              // List of sources influencing aim.
    multiAimConstraint.data.aimAxis;                    // Defines the aim direction.
    multiAimConstraint.data.offset;                     // Adds an offset to the aim direction.
    multiAimConstraint.data.rotationWeight;             // Adjusts influence of the constraint.

    // === MultiParentConstraint === //
    multiParentConstraint.data.constrainedObject;       // The object being constrained.
    multiParentConstraint.data.sourceObjects;           // List of parent sources.
    multiParentConstraint.data.positionWeight;          // Weight for position blending.
    multiParentConstraint.data.rotationWeight;          // Weight for rotation blending.

    // === MultiReferentialConstraint === //
    multiReferentialConstraint.data.constrainedObject;  // The object being controlled.
    multiReferentialConstraint.data.sourceObjects;      // List of references.
    multiReferentialConstraint.data.switchWeight;       // Blends between references.

    // === OverrideTransform === //
    overrideTransform.data.constrainedObject;           // The object being overridden.
    overrideTransform.data.sourceObject;                // The override source.
    overrideTransform.data.positionWeight;              // Influence of position override.
    overrideTransform.data.rotationWeight;              // Influence of rotation override.
}
```



LEARN

UNITY

with

50 CHEAT SHEETS

BUILD 20 GAME PROJECTS AT OUTSCAL

CLICK HERE

OUTSCAL