Wreck-It Kaiju User Manual

Welcome to Wreck-It Kaiju, a thrilling virtual reality simulation where players take on the role of a towering monster, navigating an urban landscape and causing mayhem. In this game, players can explore breakable objects, interact with a detailed cityscape, and experience engaging physics-based interactions. The game's mechanics allow for a variety of ways to interact with the world, including smashing, grabbing, and launching atomic attacks, providing a realistic and immersive experience.

Developed by Christian Bueno on the Meta Quest 2

Assets used:

- SimplePoly City - Low Poly Assets

Special assets created: Custom broken structures/objects

- **Breakable Structures:** These assets represent buildings or structures designed to break apart in response to specific triggers or interactions, such as collisions or explosions.
- **Physics-Driven Fractures:** The breaking mechanism involves Unity's physics engine, which handles the simulation of how pieces interact, fall, or collide post-breakage, creating realistic animations and responses.
- **Cell Fracture:** To generate the broken pieces, Blender's cell fracture technique is used. This method divides a structure into smaller, irregular pieces, simulating realistic damage and destruction patterns. It creates varied and visually interesting results, enhancing the breakable asset's impact.
- **Prefab-Based Design:** The assets incorporate prefabs or references to prefabs for broken pieces, ensuring consistency and ease of replication across different buildings.
- **Scripts and References:** The SimpleBreakable controls the behavior of these assets, including how and when they break, how pieces are instantiated, and how they interact with each other and their surroundings.

Game Controls

Navigation:

- Use the right Quest controller to point where you want to go, and press forward on the joystick to teleport there.

Player Actions:

- **Grab:** squeeze either grip to pick up an interactable object.
- **Throw:** Release the trigger to drop or throw the object; throwing velocity is based on controller velocity.
- **Break:** Press the trigger button while holding an interactable object to break it into pieces.
- **Atomic Attack:** Press the A or X button to shoot atomic missiles.

Scene Interaction:

- **Laser Pointer:** Use the laser pointer to navigate and interact with UI elements.
- **Start Game:** Click the red button on the start panel to begin gameplay.

Script Functions and Interactions

SceneHandler:

- **Game Initialization:** Pauses gameplay and shows a start panel (a UI button and its parent) when the scene starts. The panel resumes gameplay when clicked.
- **Pointer Events:** Handles UI and object interactions through a laser pointer system.
- **Game Resumption:** Clicking the start button resumes gameplay by resetting the time scale to 1 and hiding the start panel.
- **Object Interaction:** Processes interactions with breakable objects using the SimpleBreakable script, allowing players to break items or get feedback on non-interactable objects.

SimpleBreakable:

- **Breaking Logic:** Handles the breaking of objects into smaller pieces, spawning broken pieces and applying physics-based explosion forces to simulate the break.
- **Tag-Based Collisions:** Breaks the object on collision with objects of specific tags, allowing for controlled interaction.
- **Deactivation:** Deactivates the original object after breaking, managing scene complexity and reducing potential performance impact.

Teleporter:

- **Teleportation:** Allows the player to instantly move to a new location by pointing and clicking with the VR controller.
- **Pointer Update:** Continuously updates the teleportation pointer to indicate valid target locations, hiding the pointer when a valid location isn't found.
- **Teleport Activation:** Teleports the player when the teleport action button is pressed and released, moving the camera rig to the targeted location.
- **Visual Feedback:** Implements a fade effect to smooth the transition between teleportation points, reducing disorientation.

Hand Script:

- **Object Interaction:** Manages interactions with objects in the game world, allowing the player to grab, throw, and break objects.
- **Grabbing:** Finds the nearest interactable object and attaches it to the controller, making it available for further actions.
- **Throwing:** Releases the object, applying velocity based on the controller's movement to simulate throwing.
- **Breaking:** Calls the Break() method on the current interactable object, causing it to break into pieces.

- **Collision Tracking:** Maintains a list of interactable objects within range, enabling seamless interaction with nearby items.

Interactable:

- **General Function:** Marks the GameObject as interactable, allowing it to be picked up, manipulated, and interacted with by other scripts.
- **Active Hand Tracking:** Stores a reference to the Hand currently interacting with the object, enabling smooth handling and interaction transitions.
- **Rigidbody Integration:** Ensures the object has a Rigidbody component, making it compatible with physics-based interactions and movements.

Projectile:

- Shooting Mechanics: Manages the creation and launching of projectiles, triggered by a SteamVR input action.
- Projectile Targeting: Uses a raycast from the controller to determine the target point, aiming projectiles in the direction of the ray's endpoint.
- Instantiation and Launch: Instantiates a new projectile at the controller's position, sets its direction to the target, and applies force to shoot it towards its destination.
- Shot Delay: Implements a delay between shots, using a coroutine to manage the timing and ensure balanced gameplay.
- Projectile Lifespan: Automatically destroys the projectile after a specified time, maintaining scene complexity and performance.

ProjectileHandler:

- Collision-Based Destruction: Handles destruction of projectile objects upon collision. The projectile destroys itself when it collides with any object except those tagged "Road," managing which objects it interacts with.
- GameObject Management: Helps manage scene complexity by removing projectiles that are no longer needed, freeing up resources and maintaining a smooth gameplay experience.