MANUAL BOOK

RE:PC

Simulasi Rakit PC Berbasis Virtual Reality

Pendahuluan

RE:PC - Simulasi Rakit PC Berbasis *Virtual Reality* -, adalah aplikasi yang tertanam pada perangkat *smartphone* yang berfungsi untuk memainkan simulasi merakit komputer atau PC. Aplikasi ini bekerja dengan menggunakan VR, maka perangkat harus memiliki sensor *gyroscope* untuk menjalankan aplikasi ini. Aplikasi RE:PC menampilkan kepada pengguna bagaimana cara merakit PC dengan benar, sesuai dengan langkah-langkahnya. Aplikasi RE:PC diharapkan dapat membantu pengguna dalam merakit PC dengan benar. Aplikasi RE:PC menyasar pengguna pada kalangan masyarakat umum yang menginginkan merakit PC sendiri

KETENTUAN PENGGUNAAN APLIKASI

Berikut ini adalah ketentuan penggunaan dalam mengoperasikan aplikasi Re:PC / *Rebuilt the PC* -:

1. Memiliki spek *smartphone* yang tinggi

2. Kalau bisa harus memiliki Google Cardboard

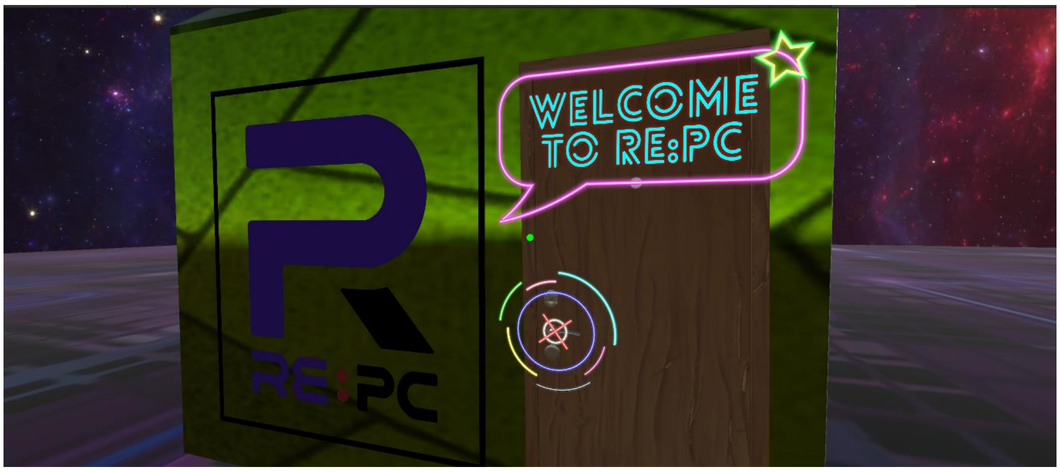
3. Tidak memiliki *motion sickness*

4. Memiliki kapasitas *storage* yang cukup, sekitar lebih dari 300 MB
5. Ada kapasitas ruang untuk bermain Re:PC

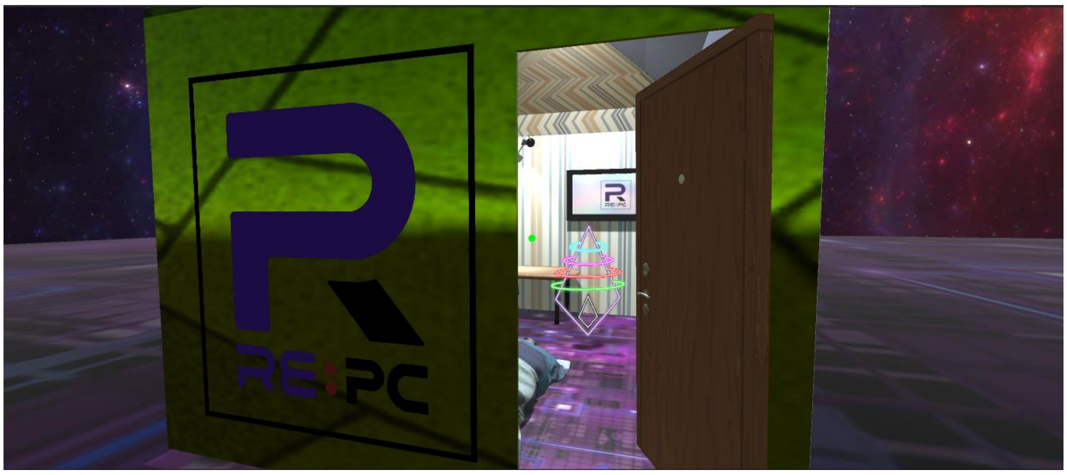
6. Jangan bermain di dapur atau dekat dengan benda yang tajam

User Interface

1. START MENU

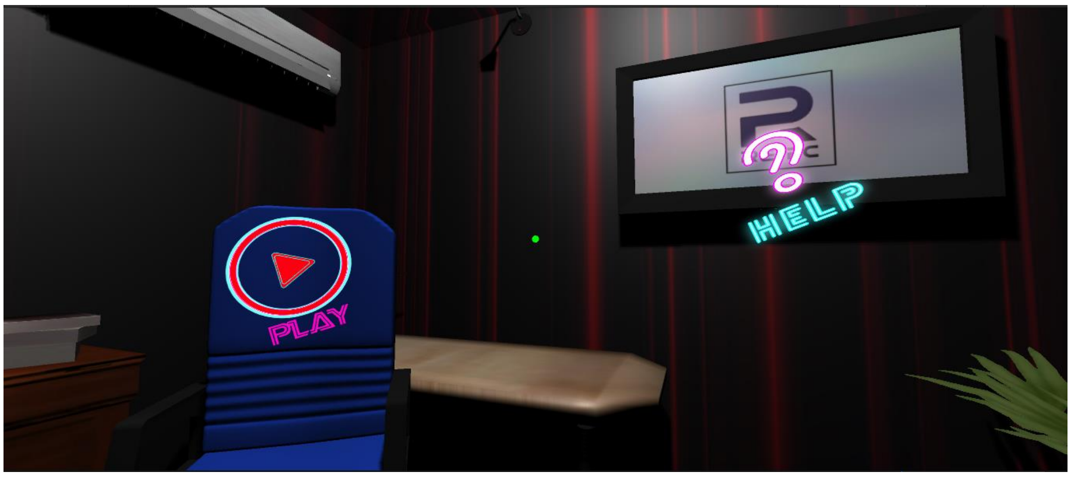


Pada saat memulai aplikasi, *user* diminta untuk mengarahkan *pointer* atau kursor ke animasi di pintu selama 2 detik untuk membuka pintu masuk.

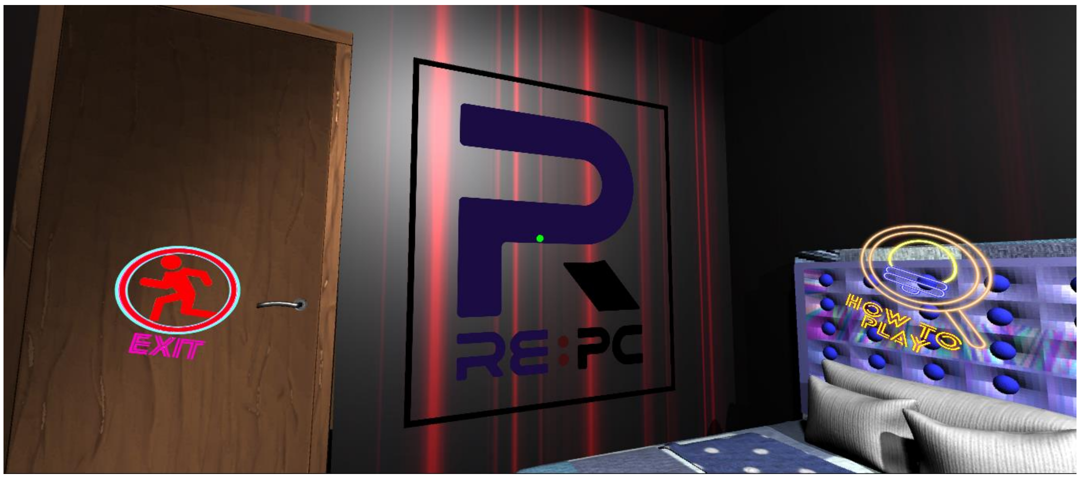


Arahkan pointer atau kursor selama 2 detik untuk memasuki *game* Re:PC.

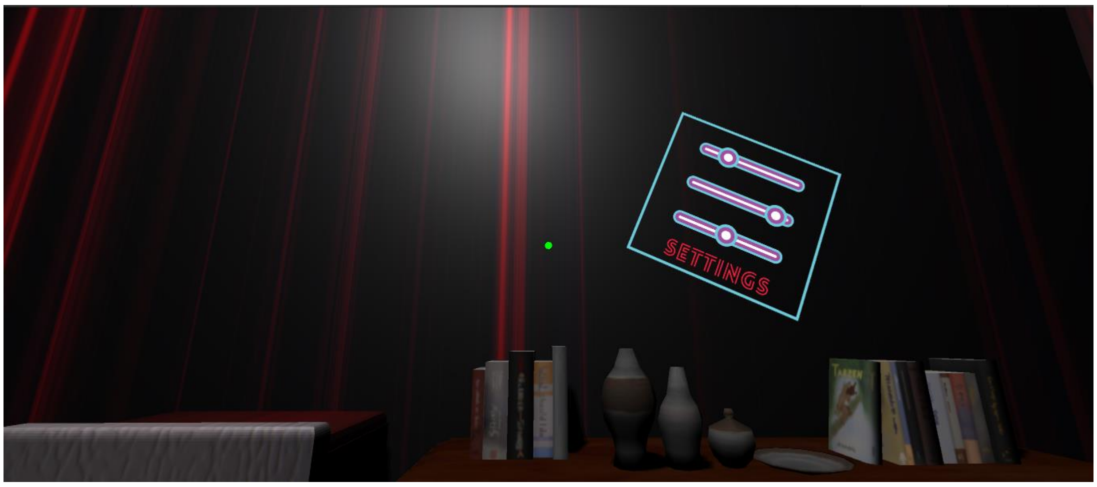
2. MAIN MENU



Di *main menu* terdapat *button play* untuk memulai *game* dan *button help* untuk menampilkan *about game* dan *credit.*

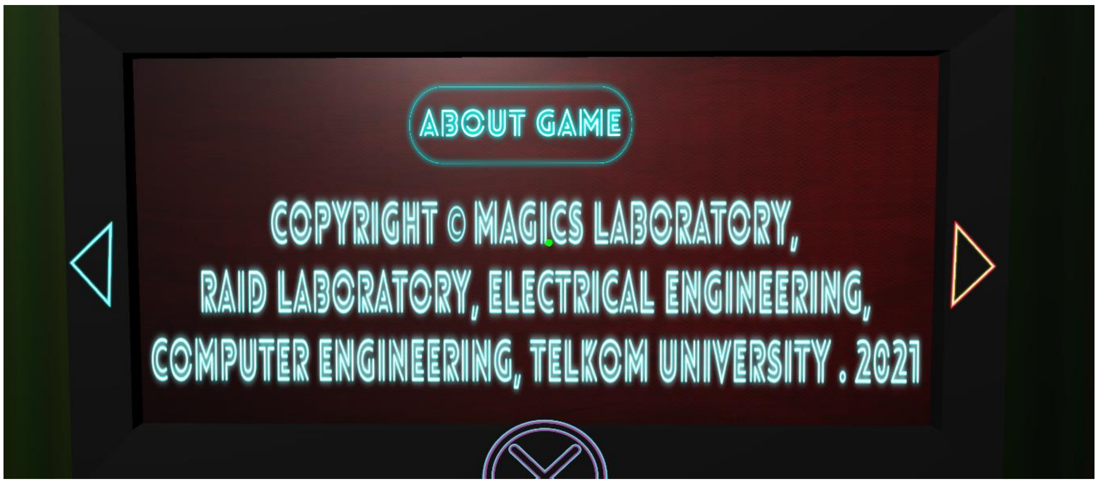


Button exit untuk keluar *game* dan *button how to play* untuk bagaimana penggunaan kursor atau button dalam game.



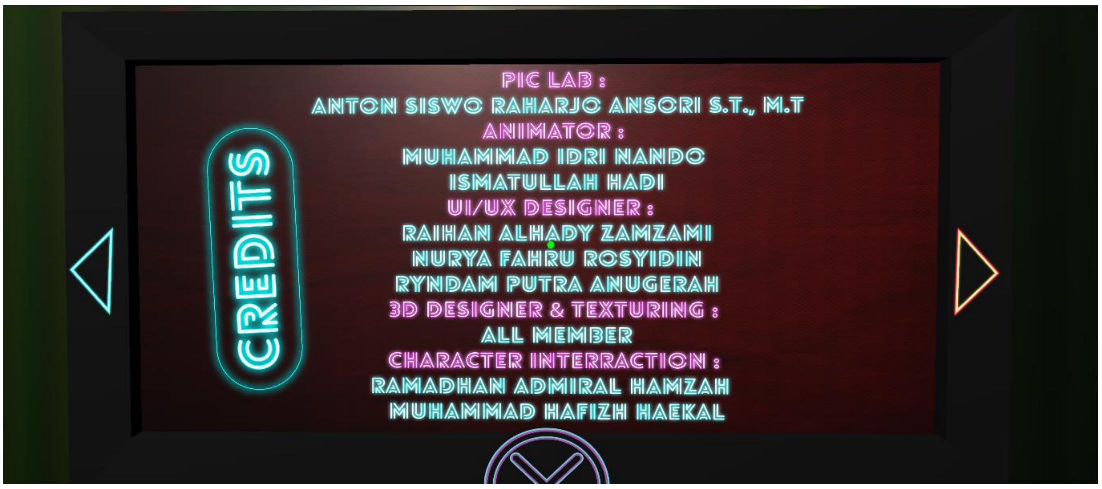
Button settings untuk pengaturan musik dan sfx.

3. HELP


ABOUT GAME

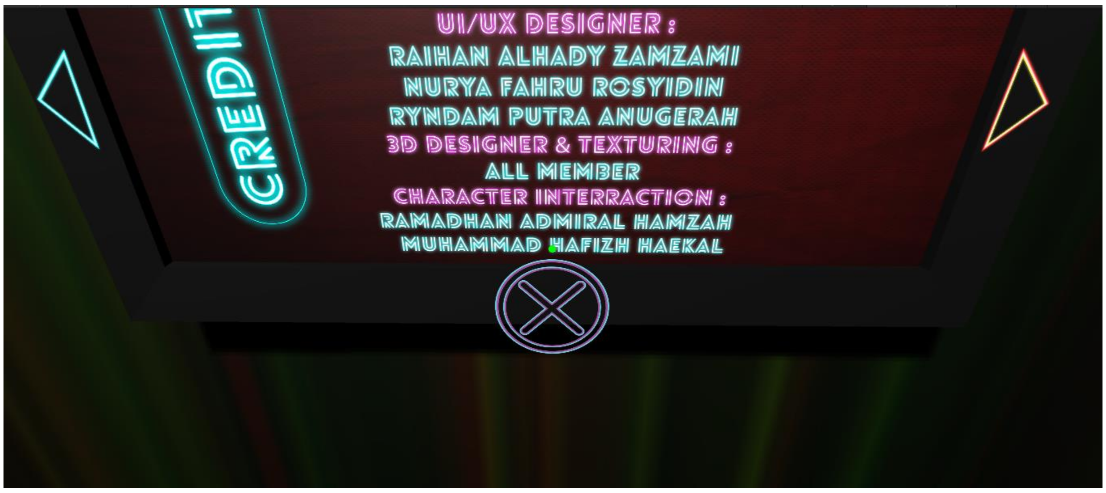
COPYRIGHT © MAGICS LABORATORY,
RAID LABORATORY, ELECTRICAL ENGINEERING,
COMPUTER ENGINEERING, TELKOM UNIVERSITY . 2021

Button tanda panah ke kanan untuk mengaktivasi perpindahan ke *credits*.


CREDITS

PIC LAB :
ANTON SISWO RAHARJO ANSORI S.T., M.T
ANIMATOR :
MUHAMMAD IDRI NANDO
ISMATULLAH HADI
UI/UX DESIGNER :
RAIHAN ALHADY ZAMZAMI
NURYA FAHRU ROSYIDIN
RYNDAM PUTRA ANUGERAH
3D DESIGNER & TEXTURING :
ALL MEMBER
CHARACTER INTERRACTION :
RAMADHAN ADMIRAL HAMZAH
MUHAMMAD HAFIZH HAEKAL

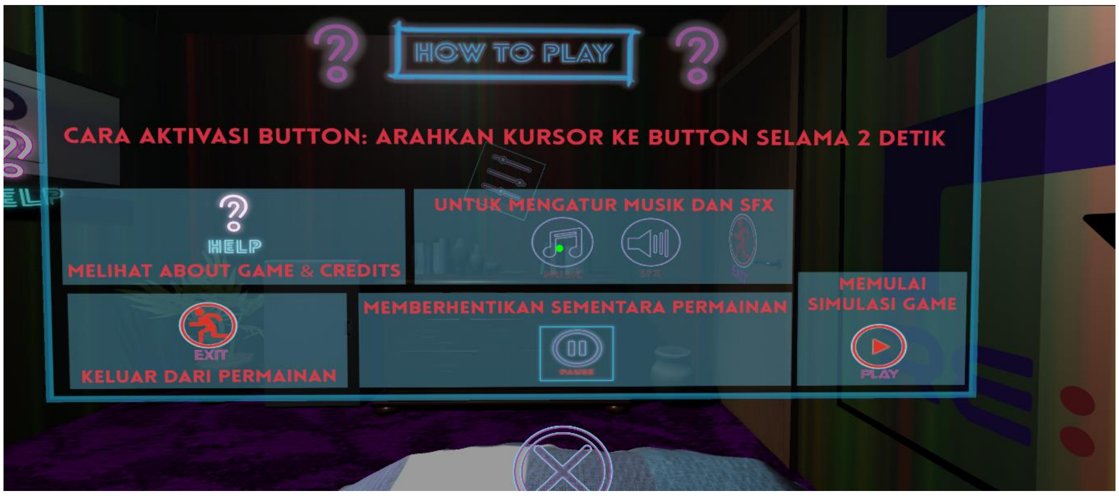
Button tanda panah ke kiri untuk aktivasi perpindahan ke *about game*.


CREDIT

UI/UX DESIGNER :
RAIHAN ALHADY ZAMZAMI
NURYA FAHRU ROSYIDIN
RYNDAM PUTRA ANUGERAH
3D DESIGNER & TEXTURING :
ALL MEMBER
CHARACTER INTERRACTION :
RAMADHAN ADMIRAL HAMZAH
MUHAMMAD HAFIZH HAEKAL

Button silang untuk aktivasi kembali ke *main menu*.

4. HOW TO PLAY


HOW TO PLAY

CARA AKTIVASI BUTTON: ARAHKAN KURSOR KE BUTTON SELAMA 2 DETIK

HELP
MELIHAT ABOUT GAME & CREDITS

UNTUK MENGATUR MUSIK DAN SFX

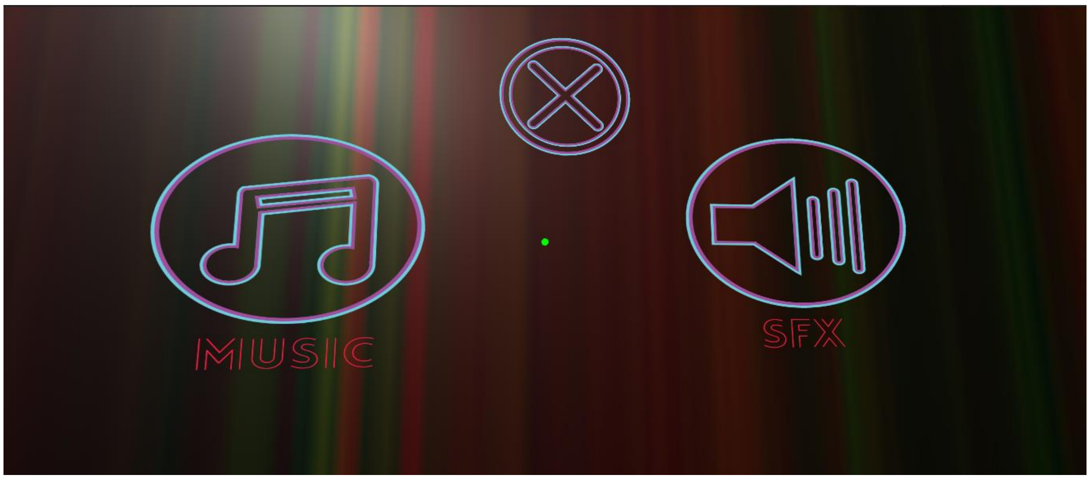
EXIT
KELUAR DARI PERMAINAN

MEMBERHENTIKAN SEMENTARA PERMAINAN

MEMULAI SIMULASI GAME
PLAY

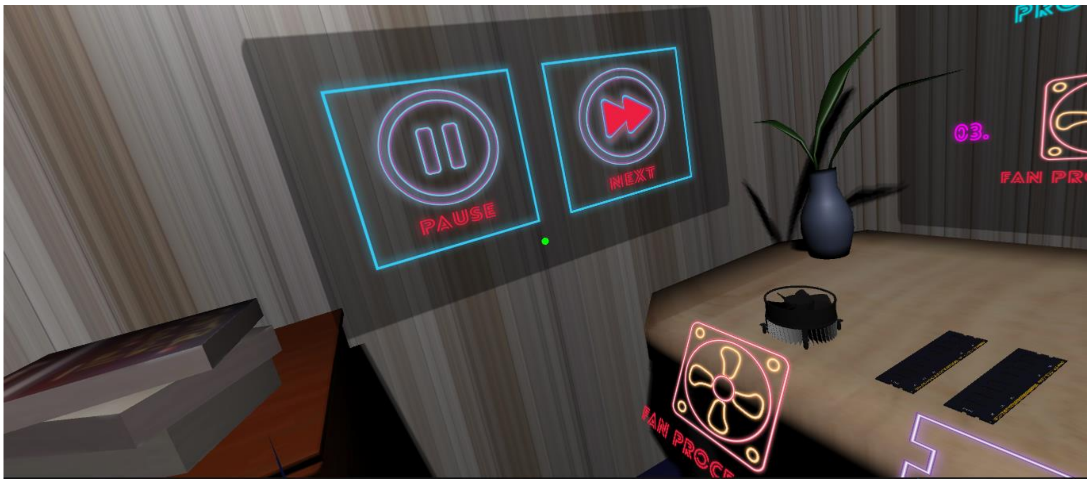
- Untuk melihat cara aktivasi *button* dan juga penjelasan tentang *button.*
- *Button* silang untuk aktivasi kembali ke *main menu.*

5. SETTINGS

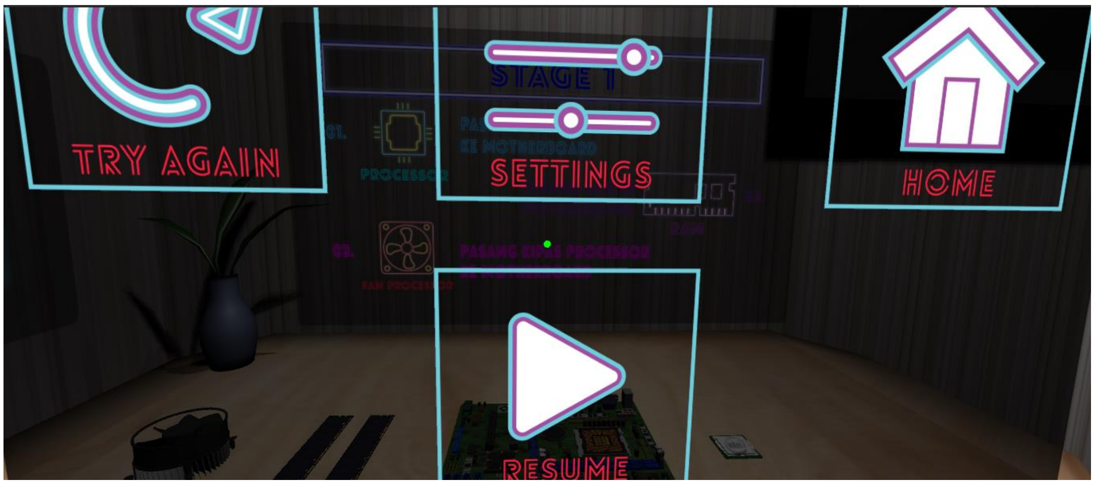


- *Button music* untuk mengaktifkan *on* dan *off music* di dalam aplikasi.
- *Button sfx* untuk mengaktifkan *on* dan *off* sfx di dalam aplikasi.
- *Button* silang untuk aktivasi kembali ke *main menu.*

6. PAUSE

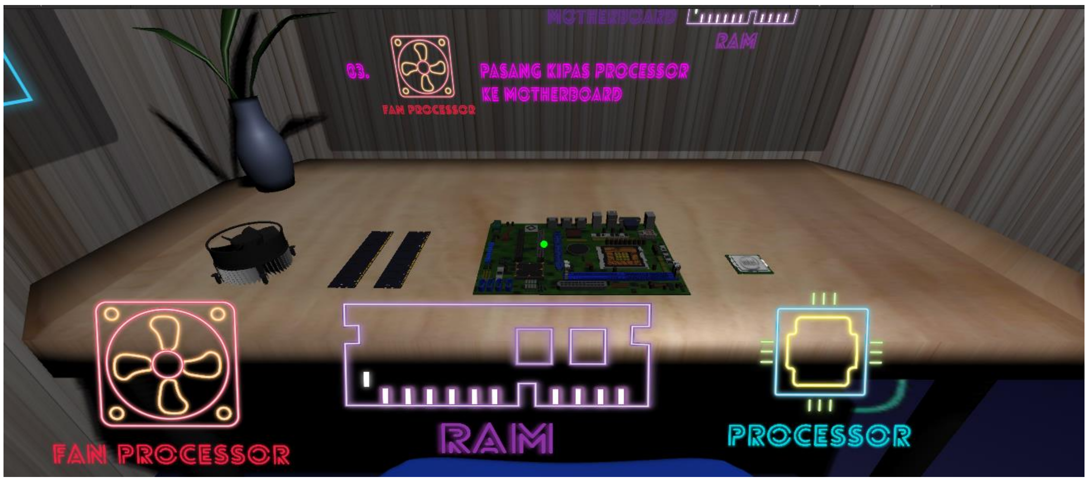


- *Button pause* untuk *pause* dalam bermain *game*.
- *Button next* untuk aktivasi ke *stage* berikutnya.



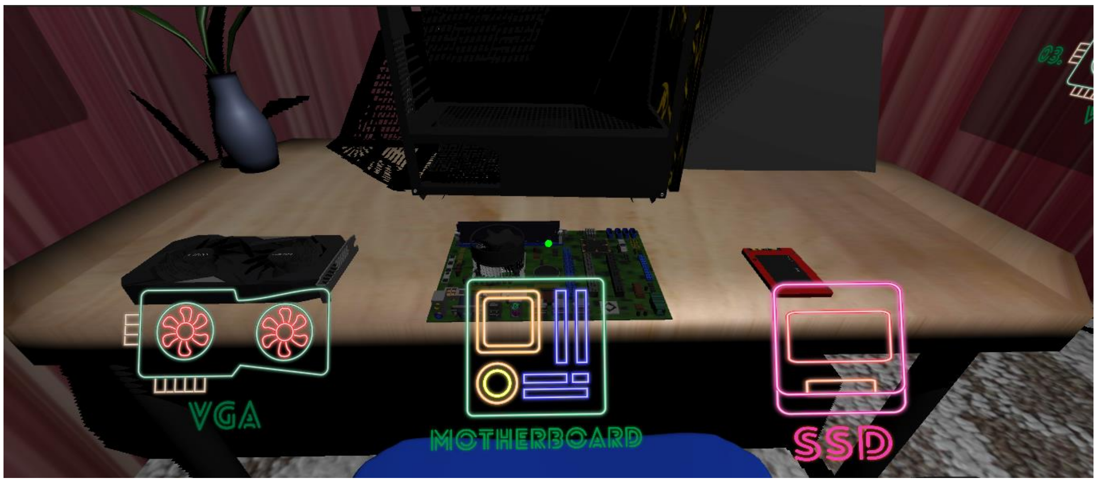
- *Button try again* untuk aktivasi mengulang kembali permainan di *stage* tersebut.
- *Button settings* untuk aktivasi ke *menu settings*.
- *Button home* untuk aktivasi ke *main menu*.
- *Button resume* untuk aktivasi kembali kedalam permainan.

7. PLAY STAGE 1



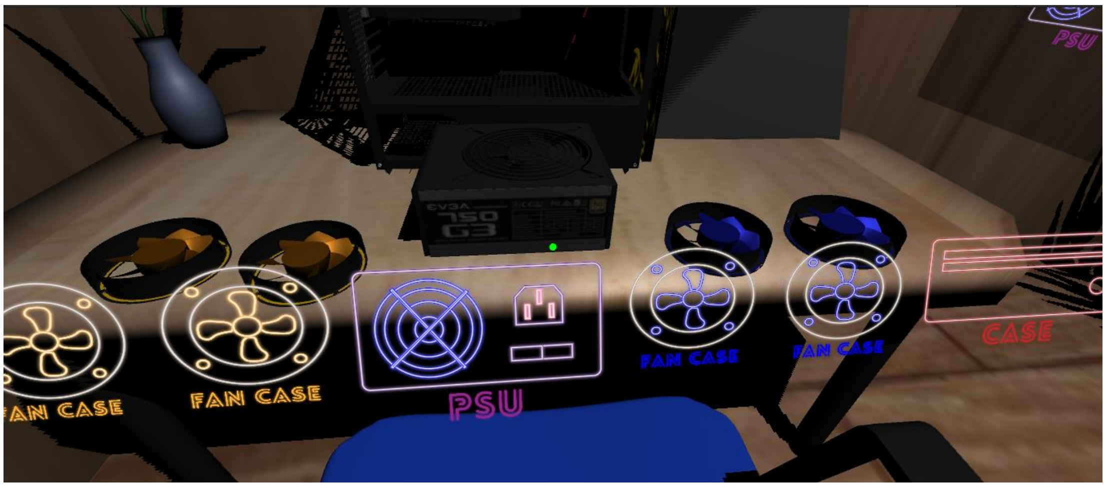
- *Button fan processor* untuk aktivasi animasi *fan* prosesor bergabung dengan *Motherboard.*
- *Button RAM* untuk aktivasi animasi *RAM* bergabung dengan *Motherboard.*
- *Button prosesor* untuk aktifasi animasi *prosesor* bergabung dengan *Motherboard.*

8. STAGE 2



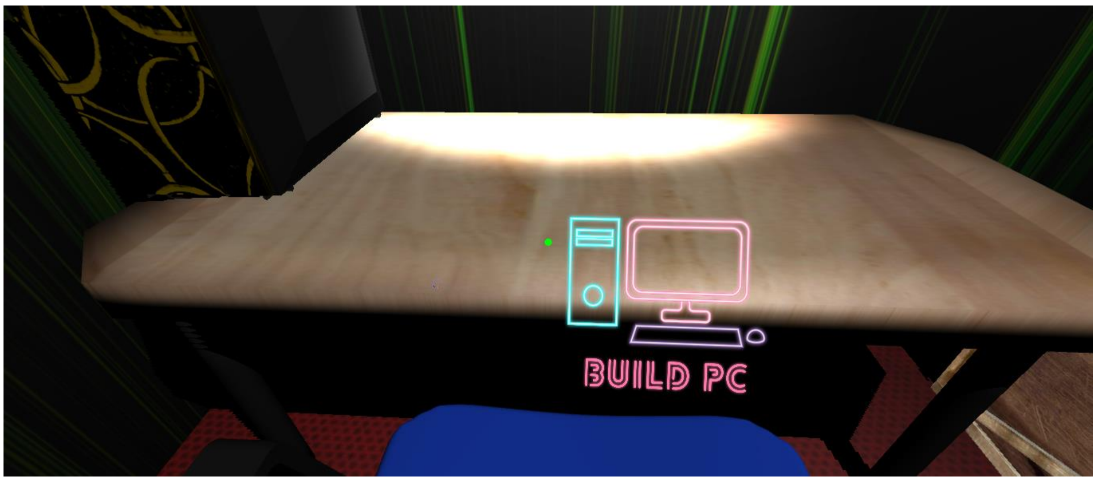
- Button *Motherboard* untuk aktivasi animasi *motherboard* masuk kedala case.
- Button *VGA* untuk aktivasi animasi *VGA* masuk kedala case.
- Button *SSD* untuk aktivasi animasi *SSD* masuk kedala case.

9. STAGE 3

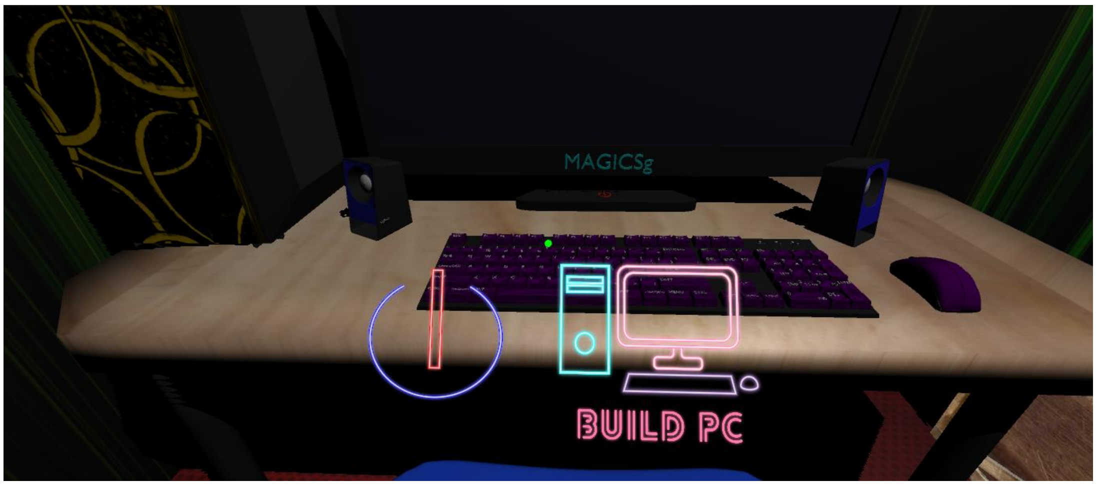


- Button *Motherboard* untuk aktivasi animasi *motherboard* masuk kedala case.
- Button *Fan Case* untuk aktivasi animasi *Fan Case* masuk kedala case.
- Button *Penutup Case* untuk aktivasi animasi *Penutup Case* masuk kedala case.

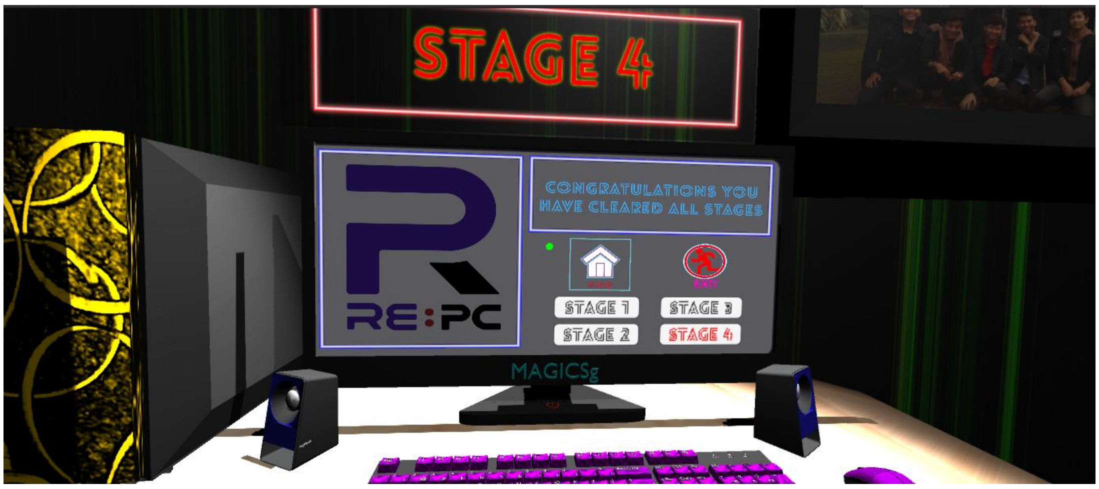
10. STAGE 4



- Button *Build PC* untuk aktivasi animasi pengabungan seluruh komponen seperti Monitor, Mouse, Keyboard, dan Speaker.



- Button On/Of untuk aktivasi animasi menyalanya computer.



- *Button home* untuk aktivasi kembali ke *main menu*.
- *Button exit* untuk aktifasi keluar *game*.
- *Button stage* 1 untuk aktivasi ke *stage* 1.
- *Button stage* 2 untuk aktivasi ke *stage* 2.
- *Button stage* 3 untuk aktivasi ke *stage* 3.
- *Button stage* 4 untuk aktivasi ke *stage* 4.

Source Code (C#)

1. GvrEditorEmulator

```
//-----------------------------------------------------------------------
// <copyright file="GvrEditorEmulator.cs" company="Google Inc.">
// Copyright 2017 Google Inc. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
// See the License for the specific language governing permissions and
// limitations under the License.
// </copyright>
//-----------------------------------------------------------------------

using System;
using System.Collections.Generic;
using Gvr.Internal;
using UnityEngine;

/// <summary>Provides mouse-controlled head tracking emulation in the Unity
editor.</summary>
[HelpURL("https://developers.google.com/vr/unity/reference/class/GvrEditorEmulator")]
public class GvrEditorEmulator : MonoBehaviour
{
    // GvrEditorEmulator should only be compiled in the Editor.
    //
    // Otherwise, it will override the camera pose every frame on device which causes the
    // following behaviour:
    //
    // The rendered camera pose will still be correct because the VR.InputTracking pose
    // gets applied after LateUpdate has occured. However, any functionality that
    // queries the camera pose during Update or LateUpdate after GvrEditorEmulator has been
    // updated will get the wrong value applied by GvrEditorEmulator intsead.
#if UNITY_EDITOR
    private const string AXIS_MOUSE_X = "Mouse X";
private const string AXIS_MOUSE_Y = "Mouse Y";

// Simulated neck model.  Vector from the neck pivot point to the point between the eyes.
private static readonly Vector3 NECK_OFFSET = new Vector3(0, 0.075f, 0.08f);

private static GvrEditorEmulator instance;
private static bool instanceSearchedFor = false;

// Allocate an initial capacity; this will be resized if needed.
private static Camera[] allCameras = new Camera[32];

// Use mouse to emulate head in the editor.
// These variables must be static so that head pose is maintained between scene changes,
// as it is on device.
private float mouseX = 0;
private float mouseY = 0;
private float mouseZ = 0;

/// <summary>Gets the instance for this singleton class.</summary>
/// <value>The instance for this singleton class.</value>
public static GvrEditorEmulator Instance
{
   get
   {
      if (instance == null && !instanceSearchedFor)
      {
         instance = FindObjectOfType<GvrEditorEmulator>();
         instanceSearchedFor = true;
      }

      return instance;
   }
}

/// <summary>Gets the emulated head position.</summary>
/// <value>The emulated head position.</value>
public Vector3 HeadPosition { get; private set; }

/// <summary>Gets the emulated head rotation.</summary>
/// <value>The emulated head rotation.</value>
public Quaternion HeadRotation { get; private set; }

/// <summary>Recenters the emulated headset.</summary>
public void Recenter()
```

```
{
   mouseX = mouseZ = 0;  // Do not reset pitch, which is how it works on the phone.
   UpdateHeadPositionAndRotation();
   ApplyHeadOrientationToVRCameras();
}

/// <summary>Single-frame updates for this module.</summary>
/// <remarks>Should be called in one MonoBehavior's `Update` method.</remarks>
public void UpdateEditorEmulation()
{
   if (InstantPreview.IsActive)
   {
      return;
   }

   if (GvrControllerInput.Recentered)
   {
      Recenter();
   }

   bool rolled = false;
   if (CanChangeYawPitch())
   {
      GvrCursorHelper.HeadEmulationActive = true;
      mouseX += Input.GetAxis(AXIS_MOUSE_X) * 5;
      if (mouseX <= -180)
      {
         mouseX += 360;
      }
      else if (mouseX > 180)
      {
         mouseX -= 360;
      }

      mouseY -= Input.GetAxis(AXIS_MOUSE_Y) * 2.4f;
      mouseY = Mathf.Clamp(mouseY, -85, 85);
   }
   else if (CanChangeRoll())
   {
      GvrCursorHelper.HeadEmulationActive = true;
      rolled = true;
      mouseZ += Input.GetAxis(AXIS_MOUSE_X) * 5;
      mouseZ = Mathf.Clamp(mouseZ, -85, 85);
   }
```

```
        else
        {
            GvrCursorHelper.HeadEmulationActive = false;
        }

        if (!rolled)
        {
            // People don't usually leave their heads tilted to one side for long.
            mouseZ = Mathf.Lerp(mouseZ, 0, Time.deltaTime / (Time.deltaTime + 0.1f));
        }

        UpdateHeadPositionAndRotation();
        ApplyHeadOrientationToVRCameras();
    }

    private void Awake()
    {
        if (Instance == null)
        {
            instance = this;
        }
        else if (Instance != this)
        {
            Debug.LogError("More than one active GvrEditorEmulator instance was found in
your " +
                           "scene.  Ensure that there is only one active GvrEditorEmulator.");
            this.enabled = false;
            return;
        }
    }

    private void Start()
    {
        UpdateAllCameras();
        for (int i = 0; i < Camera.allCamerasCount; ++i)
        {
            Camera cam = allCameras[i];

            // Only check camera if it is an enabled VR Camera.
            if (cam && cam.enabled && cam.stereoTargetEye != StereoTargetEyeMask.None)
            {
                if (cam.nearClipPlane > 0.1
                    && GvrSettings.ViewerPlatform ==
GvrSettings.ViewerPlatformType.Daydream)
```

```
            {
                Debug.LogWarningFormat(
                    "Camera \"{0}\" has Near clipping plane set to {1} meters, which might " +
                    "cause the rendering of the Daydream controller to clip unexpectedly.\n" +
                    "Suggest using a lower value, 0.1 meters or less.",
                    cam.name, cam.nearClipPlane);
            }
        }
    }
}

private void Update()
{
    // GvrControllerInput automatically updates GvrEditorEmulator.
    // This guarantees that GvrEditorEmulator is updated before anything else responds to
    // controller input, which ensures that re-centering works correctly in the editor.
    // If GvrControllerInput is not available, then fallback to using Update().
    if (GvrControllerInput.ApiStatus != GvrControllerApiStatus.Error)
    {
        return;
    }

    UpdateEditorEmulation();
}

private bool CanChangeYawPitch()
{
    // If the MouseControllerProvider is currently active, then don't move the camera.
    if (MouseControllerProvider.IsActivateButtonPressed)
    {
        return false;
    }

    return Input.GetKey(KeyCode.LeftAlt) || Input.GetKey(KeyCode.RightAlt);
}

private bool CanChangeRoll()
{
    // If the MouseControllerProvider is currently active, then don't move the camera.
    if (MouseControllerProvider.IsActivateButtonPressed)
    {
        return false;
    }
    return Input.GetKey(KeyCode.LeftControl) || Input.GetKey(KeyCode.RightControl);
}

private void UpdateHeadPositionAndRotation()
{
    HeadRotation = Quaternion.Euler(mouseY, mouseX, mouseZ);
    HeadPosition = (HeadRotation * NECK_OFFSET) - (NECK_OFFSET.y * Vector3.up);
}

private void ApplyHeadOrientationToVRCameras()
{
    UpdateAllCameras();

    // Update all VR cameras using Head position and rotation information.
    for (int i = 0; i < Camera.allCamerasCount; ++i)
    {
        Camera cam = allCameras[i];

        // Check if the Camera is a valid VR Camera, and if so update it to track head motion.
        if (cam && cam.enabled && cam.stereoTargetEye != StereoTargetEyeMask.None)
        {
            cam.transform.localPosition = HeadPosition * cam.transform.lossyScale.y;
            cam.transform.localRotation = HeadRotation;
        }
    }
}

// Avoids per-frame allocations. Allocates only when allCameras array is resized.
private void UpdateAllCameras()
{
    // Get all Cameras in the scene using persistent data structures.
    if (Camera.allCamerasCount > allCameras.Length)
    {
        int newAllCamerasSize = Camera.allCamerasCount;
        while (Camera.allCamerasCount > newAllCamerasSize)
        {
            newAllCamerasSize *= 2;
        }

        allCameras = new Camera[newAllCamerasSize];
    }

    // The GetAllCameras method doesn't allocate memory (Camera.allCameras does).
    Camera.GetAllCameras(allCameras);
    }

#endif  // UNITY_EDITOR
}
```

2. GvrEventSystem

```
//-----------------------------------------------------------------------
// <copyright file="GvrPointerInputModule.cs" company="Google Inc.">
// Copyright 2016 Google Inc. All rights reserved.
//
// Licensed under the MIT License, you may not use this file except in
// compliance with the License. You may obtain a copy of the License at
//
//     http://www.opensource.org/licenses/mit-license.php
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
// See the License for the specific language governing permissions and
// limitations under the License.
// </copyright>
//-----------------------------------------------------------------------

using System.Collections.Generic;
using Gvr.Internal;
using UnityEngine;
using UnityEngine.EventSystems;

/// <summary>This script provides an implemention of Unity's `BaseInputModule`
class.</summary>
/// <remarks><para>
/// Exists so that Canvas-based (`uGUI`) UI elements and 3D scene objects can be interacted
with in
/// a Gvr Application.
/// </para><para>
/// This script is intended for use with either a 3D Pointer with the Daydream Controller
/// (Recommended for Daydream), or a Gaze-based-Pointer (Recommended for Cardboard).
/// </para><para>
/// To use, attach to the scene's **EventSystem** object.  Be sure to move it above the
/// other modules, such as `TouchInputModule` and `StandaloneInputModule`, in order
/// for the Pointer to take priority in the event system.
/// </para><para>
/// If you are using a **Canvas**, set the `Render Mode` to **World Space**, and add the
/// `GvrPointerGraphicRaycaster` script to the object.
/// </para><para>
/// If you'd like pointers to work with 3D scene objects, add a `GvrPointerPhysicsRaycaster`
to the
/// main camera, and add a component that implements one of the `Event` interfaces
(`EventTrigger`
/// will work nicely) to an object with a collider.
/// </para><para>
/// `GvrPointerInputModule` emits the following events: `Enter`, `Exit`, `Down`, `Up`,
`Click`,
/// `Select`, `Deselect`, `UpdateSelected`, and `GvrPointerHover`.  Scroll, move, and
submit/cancel
/// events are not emitted.
/// </para><para>
/// To use a 3D Pointer with the Daydream Controller:
///   - Add the prefab GoogleVR/Prefabs/UI/GvrControllerPointer to your scene.
///   - Set the parent of `GvrControllerPointer` to the same parent as the main camera
///     (With a local position of 0,0,0).
/// </para><para>
/// To use a Gaze-based-pointer:
///   - Add the prefab GoogleVR/Prefabs/UI/GvrReticlePointer to your scene.
///   - Set the parent of `GvrReticlePointer` to the main camera.
/// </para></remarks>
[AddComponentMenu("GoogleVR/GvrPointerInputModule")]
[HelpURL("https://developers.google.com/vr/unity/reference/class/GvrPointerInputModule")
]
public class GvrPointerInputModule : BaseInputModule, IGvrInputModuleController
{
    /// <summary>
    /// If `true`, pointer input is active in VR Mode only.
    /// If `false`, pointer input is active all of the time.
    /// </summary>
    /// <remarks>
    /// Set to false if you plan to use direct screen taps or other input when not in VR Mode.
    /// </remarks>
    [Tooltip("Whether Pointer input is active in VR Mode only (true), or all the time (false).")]
    public bool vrModeOnly = false;

    /// <summary>Manages scroll events for the input module.</summary>
    [Tooltip("Manages scroll events for the input module.")]
    public GvrPointerScrollInput scrollInput = new GvrPointerScrollInput();

    /// <summary>Gets or sets the static reference to the `GvrBasePointer`.</summary>
    /// <value>The static reference to the `GvrBasePointer`.</value>
```

```
public static GvrBasePointer Pointer
{
   get
   {
      GvrPointerInputModule module = FindInputModule();
      if (module == null || module.Impl == null)
      {
         return null;
      }

      return module.Impl.Pointer;
   }

   set
   {
      GvrPointerInputModule module = FindInputModule();
      if (module == null || module.Impl == null)
      {
         return;
      }

      module.Impl.Pointer = value;
   }
}

/// <summary>Gets the current `RaycastResult`.</summary>
/// <value>The current `RaycastResult`.</value>
public static RaycastResult CurrentRaycastResult
{
   get
   {
      GvrPointerInputModule inputModule = GvrPointerInputModule.FindInputModule();
      if (inputModule == null)
      {
         return new RaycastResult();
      }

      if (inputModule.Impl == null)
      {
         return new RaycastResult();
      }

      if (inputModule.Impl.CurrentEventData == null)
      {
            return new RaycastResult();
        }

        return inputModule.Impl.CurrentEventData.pointerCurrentRaycast;
    }
}

/// <summary>Gets the implementation object of this module.</summary>
/// <value>The implementation object of this module.</value>
public GvrPointerInputModuleImpl Impl { get; private set; }

/// <summary>Gets the executor this module uses to process events.</summary>
/// <value>The executor this module uses to process events.</value>
public GvrEventExecutor EventExecutor { get; private set; }

/// <summary>Gets the event system reference.</summary>
/// <value>The event system reference.</value>
[System.Diagnostics.CodeAnalysis.SuppressMessage(
    "UnityRules.LegacyGvrStyleRules",
    "VR1001:AccessibleNonConstantPropertiesMustBeUpperCamelCase",
    Justification = "Legacy Public API.")]
public new EventSystem eventSystem
{
    get
    {
        return base.eventSystem;
    }
}

/// <summary>Gets the list of raycast results used as a cache.</summary>
/// <value>The list of raycast results used as a cache.</value>
public List<RaycastResult> RaycastResultCache
{
    get
    {
        return m_RaycastResultCache;
    }
}

/// <summary>The `GvrBasePointer` calls this when it is created.</summary>
/// <remarks>
/// If a pointer hasn't already been assigned, it will assign the newly created one by default.
/// This simplifies the common case of having only one `GvrBasePointer` so it can be
/// automatically hooked up to the manager.  If multiple `GvrBasePointers` are in the scene,
/// the app has to take responsibility for setting which one is active.
/// </remarks>
/// <param name="createdPointer">The pointer whose creation triggered this call.</param>
public static void OnPointerCreated(GvrBasePointer createdPointer)
{
   GvrPointerInputModule module = FindInputModule();
   if (module == null || module.Impl == null)
   {
      return;
   }

   if (module.Impl.Pointer == null)
   {
      module.Impl.Pointer = createdPointer;
   }
}

/// <summary>
/// Helper function to find the Event executor that is part of the input module if one exists
/// in the scene.
/// </summary>
/// <returns>A found GvrEventExecutor or null.</returns>
public static GvrEventExecutor FindEventExecutor()
{
   GvrPointerInputModule gvrInputModule = FindInputModule();
   if (gvrInputModule == null)
   {
      return null;
   }

   return gvrInputModule.EventExecutor;
}

/// <summary>
/// Helper function to find the input module if one exists in the scene and it is the active
/// module.
/// </summary>
/// <returns>A found `GvrPointerInputModule` or null.</returns>
public static GvrPointerInputModule FindInputModule()
{
   if (EventSystem.current == null)
   {
      return null;
   }
```

```
    EventSystem eventSystem = EventSystem.current;
    if (eventSystem == null)
    {
        return null;
    }

    GvrPointerInputModule gvrInputModule =
        eventSystem.GetComponent<GvrPointerInputModule>();

    return gvrInputModule;
}

/// <inheritdoc/>
[SuppressMemoryAllocationError(IsWarning = true, Reason = "Pending documentation.")]
public override bool ShouldActivateModule()
{
    return Impl.ShouldActivateModule();
}

/// <inheritdoc/>
[SuppressMemoryAllocationError(IsWarning = true, Reason = "Pending documentation.")]
public override void DeactivateModule()
{
    Impl.DeactivateModule();
}

/// <inheritdoc/>
public override bool IsPointerOverGameObject(int pointerId)
{
    return Impl.IsPointerOverGameObject(pointerId);
}

/// <inheritdoc/>
[SuppressMemoryAllocationError(IsWarning = true, Reason = "Pending documentation.")]
public override void Process()
{
    UpdateImplProperties();
    Impl.Process();
}

/// <summary>Whether the module should be activated.</summary>
/// <returns>Returns `true` if this module should be activated, `false` otherwise.</returns>
[SuppressMemoryAllocationError(IsWarning = true, Reason = "Pending documentation.")]
```

```
public bool ShouldActivate()
{
    return base.ShouldActivateModule();
}

/// <summary>Deactivate this instance.</summary>
public void Deactivate()
{
    base.DeactivateModule();
}

/// <summary>Finds the common root between two `GameObject`s.</summary>
/// <returns>The common root.</returns>
/// <param name="g1">The first `GameObject`.</param>
/// <param name="g2">The second `GameObject`.</param>
[SuppressMemoryAllocationError(IsWarning = true, Reason = "Pending documentation.")]
public new GameObject FindCommonRoot(GameObject g1, GameObject g2)
{
    return BaseInputModule.FindCommonRoot(g1, g2);
}

/// <summary>Gets the base event data.</summary>
/// <returns>The base event data.</returns>
[SuppressMemoryAllocationError(IsWarning = true, Reason = "Pending documentation.")]
public new BaseEventData GetBaseEventData()
{
    return base.GetBaseEventData();
}

/// <summary>Finds the first raycast.</summary>
/// <returns>The first raycast.</returns>
/// <param name="candidates">
/// The list of `RaycastResult`s to search for the first Raycast.
/// </param>
public new RaycastResult FindFirstRaycast(List<RaycastResult> candidates)
{
    return BaseInputModule.FindFirstRaycast(candidates);
}

/// @cond
/// <inheritdoc/>
protected override void Awake()
{
    base.Awake();
```

```
        Impl = new GvrPointerInputModuleImpl();
        EventExecutor = new GvrEventExecutor();
        UpdateImplProperties();
    }

    /// @endcond
    /// <summary>Update implementation properties.</summary>
    private void UpdateImplProperties()
    {
        if (Impl == null)
        {
            return;
        }

        Impl.ScrollInput = scrollInput;
        Impl.VrModeOnly = vrModeOnly;
        Impl.ModuleController = this;
        Impl.EventExecutor = EventExecutor;
    }
}
```

3. GvrReticlePointer

```
//-----------------------------------------------------------------------
// <copyright file="GvrReticlePointer.cs" company="Google Inc.">
// Copyright 2017 Google Inc. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
// See the License for the specific language governing permissions and
// limitations under the License.
// </copyright>
//-----------------------------------------------------------------------

using UnityEngine;
using UnityEngine.EventSystems;
```

```
/// <summary>Draws a circular reticle in front of any object that the user points
at.</summary>
/// <remarks>The circle dilates if the object is clickable.</remarks>
[HelpURL("https://developers.google.com/vr/unity/reference/class/GvrReticlePointer")]
public class GvrReticlePointer : GvrBasePointer
{
    /// <summary>
    /// The constants below are expsed for testing. Minimum inner angle of the reticle (in
degrees).
    /// </summary>
    public const float RETICLE_MIN_INNER_ANGLE = 0.0f;

    /// <summary>Minimum outer angle of the reticle (in degrees).</summary>
    public const float RETICLE_MIN_OUTER_ANGLE = 0.5f;

    /// <summary>
    /// Angle at which to expand the reticle when intersecting with an object (in degrees).
    /// </summary>
    public const float RETICLE_GROWTH_ANGLE = 1.5f;

    /// <summary>Minimum distance of the reticle (in meters).</summary>
    public const float RETICLE_DISTANCE_MIN = 0.45f;

    /// <summary>Maximum distance of the reticle (in meters).</summary>
    public float maxReticleDistance = 20.0f;

    /// <summary>Number of segments making the reticle circle.</summary>
    public int reticleSegments = 20;

    /// <summary>Growth speed multiplier for the reticle.</summary>
    public float reticleGrowthSpeed = 8.0f;

    /// <summary>Sorting order to use for the reticle's renderer.</summary>
    /// <remarks><para>
    /// Range values come from https://docs.unity3d.com/ScriptReference/Renderer-
sortingOrder.html.
    /// </para><para>
    /// Default value 32767 ensures gaze reticle is always rendered on top.
    /// </para></remarks>
    [Range(-32767, 32767)]
    public int reticleSortingOrder = 32767;

    /// <summary>Gets or sets the material used to render the reticle.</summary>
    /// <value>The material used to render the reticle.</value>
    public Material MaterialComp { private get; set; }

    /// <summary>Gets the current inner angle of the reticle (in degrees).</summary>
    /// <remarks>Exposed for testing.</remarks>
    /// <value>The current inner angle of the reticle (in degrees).</value>
    public float ReticleInnerAngle { get; private set; }

    /// <summary>Gets the current outer angle of the reticle (in degrees).</summary>
    /// <remarks>Exposed for testing.</remarks>
    /// <value>The current outer angle of the reticle (in degrees).</value>
    public float ReticleOuterAngle { get; private set; }

    /// <summary>Gets the current distance of the reticle (in meters).</summary>
    /// <remarks>Getter exposed for testing.</remarks>
    /// <value>The current distance of the reticle (in meters).</value>
    public float ReticleDistanceInMeters { get; private set; }

    /// <summary>
    /// Gets the current inner and outer diameters of the reticle, before distance multiplication.
    /// </summary>
    /// <remarks>Getters exposed for testing.</remarks>
    /// <value>
    /// The current inner and outer diameters of the reticle, before distance multiplication.
    /// </value>
    public float ReticleInnerDiameter { get; private set; }

    /// <summary>Gets the current outer diameter of the reticle (in meters).</summary>
    /// <value>The current outer diameter of the reticle (in meters).</value>
    public float ReticleOuterDiameter { get; private set; }

    /// <inheritdoc/>
    public override float MaxPointerDistance
    {
        get { return maxReticleDistance; }
    }

    /// <inheritdoc/>
    public override void OnPointerEnter(RaycastResult raycastResultResult, bool
isInteractive)
    {
        SetPointerTarget(raycastResultResult.worldPosition, isInteractive);
    }

    /// <inheritdoc/>
    public override void OnPointerHover(RaycastResult raycastResultResult, bool
isInteractive)
    {
        SetPointerTarget(raycastResultResult.worldPosition, isInteractive);
    }

    /// <inheritdoc/>
    public override void OnPointerExit(GameObject previousObject)
    {
        ReticleDistanceInMeters = maxReticleDistance;
        ReticleInnerAngle = RETICLE_MIN_INNER_ANGLE;
        ReticleOuterAngle = RETICLE_MIN_OUTER_ANGLE;
    }

    /// <inheritdoc/>
    public override void OnPointerClickDown()
    {
    }

    /// <inheritdoc/>
    public override void OnPointerClickUp()
    {
    }

    /// <inheritdoc/>
    public override void GetPointerRadius(out float enterRadius, out float exitRadius)
    {
        float min_inner_angle_radians = Mathf.Deg2Rad * RETICLE_MIN_INNER_ANGLE;

        float max_inner_angle_radians =
            Mathf.Deg2Rad * (RETICLE_MIN_INNER_ANGLE +
RETICLE_GROWTH_ANGLE);

        enterRadius = 2.0f * Mathf.Tan(min_inner_angle_radians);
        exitRadius = 2.0f * Mathf.Tan(max_inner_angle_radians);
    }

    /// <summary>Updates the material based on the reticle properties.</summary>
    public void UpdateDiameters()
    {
        ReticleDistanceInMeters =
      Mathf.Clamp(ReticleDistanceInMeters, RETICLE_DISTANCE_MIN,
maxReticleDistance);
```

```
        if (ReticleInnerAngle < RETICLE_MIN_INNER_ANGLE)
        {
            ReticleInnerAngle = RETICLE_MIN_INNER_ANGLE;
        }

        if (ReticleOuterAngle < RETICLE_MIN_OUTER_ANGLE)
        {
            ReticleOuterAngle = RETICLE_MIN_OUTER_ANGLE;
        }

        float inner_half_angle_radians = Mathf.Deg2Rad * ReticleInnerAngle * 0.5f;
        float outer_half_angle_radians = Mathf.Deg2Rad * ReticleOuterAngle * 0.5f;

        float inner_diameter = 2.0f * Mathf.Tan(inner_half_angle_radians);
        float outer_diameter = 2.0f * Mathf.Tan(outer_half_angle_radians);

        ReticleInnerDiameter =
      Mathf.Lerp(ReticleInnerDiameter, inner_diameter, Time.unscaledDeltaTime *
reticleGrowthSpeed);
        ReticleOuterDiameter =
      Mathf.Lerp(ReticleOuterDiameter, outer_diameter, Time.unscaledDeltaTime *
reticleGrowthSpeed);

        MaterialComp.SetFloat("_InnerDiameter", ReticleInnerDiameter *
ReticleDistanceInMeters);
        MaterialComp.SetFloat("_OuterDiameter", ReticleOuterDiameter *
ReticleDistanceInMeters);
        MaterialComp.SetFloat("_DistanceInMeters", ReticleDistanceInMeters);
    }

    /// @cond
    /// <inheritdoc/>
    protected override void Start()
    {
        base.Start();

        Renderer rendererComponent = GetComponent<Renderer>();
        rendererComponent.sortingOrder = reticleSortingOrder;

        MaterialComp = rendererComponent.material;

        CreateReticleVertices();
    }
    /// @endcond
    /// <summary>This MonoBehavior's Awake behavior.</summary>
    private void Awake()
    {
        ReticleInnerAngle = RETICLE_MIN_INNER_ANGLE;
        ReticleOuterAngle = RETICLE_MIN_OUTER_ANGLE;
    }

    /// @cond
    /// <summary>This MonoBehavior's `Update` method.</summary>
    private void Update()
    {
        UpdateDiameters();
    }

    /// @endcond
    /// <summary>Sets the reticle pointer's target.</summary>
    /// <param name="target">The target location.</param>
    /// <param name="interactive">Whether the pointer is pointing at an interactive
object.</param>
    /// <returns>Returns `true` if the target is set successfully.</returns>
    private bool SetPointerTarget(Vector3 target, bool interactive)
    {
        if (PointerTransform == null)
        {
            Debug.LogWarning("Cannot operate on a null pointer transform");
            return false;
        }

        Vector3 targetLocalPosition = PointerTransform.InverseTransformPoint(target);

        ReticleDistanceInMeters = Mathf.Clamp(targetLocalPosition.z,
                                              RETICLE_DISTANCE_MIN,
                                              maxReticleDistance);
        if (interactive)
        {
            ReticleInnerAngle = RETICLE_MIN_INNER_ANGLE +
RETICLE_GROWTH_ANGLE;
            ReticleOuterAngle = RETICLE_MIN_OUTER_ANGLE +
RETICLE_GROWTH_ANGLE;
        }
        else
        {
            ReticleInnerAngle = RETICLE_MIN_INNER_ANGLE;
            ReticleOuterAngle = RETICLE_MIN_OUTER_ANGLE;
        }

        return true;
    }

    private void CreateReticleVertices()
    {
        Mesh mesh = new Mesh();
        gameObject.AddComponent<MeshFilter>();
        GetComponent<MeshFilter>().mesh = mesh;

        int segments_count = reticleSegments;
        int vertex_count = (segments_count + 1) * 2;

#region Vertices

        Vector3[] vertices = new Vector3[vertex_count];

        const float kTwoPi = Mathf.PI * 2.0f;
        int vi = 0;
        for (int si = 0; si <= segments_count; ++si)
        {
            // Add two vertices for every circle segment: one at the beginning of the
            // prism, and one at the end of the prism.
            float angle = (float)si / (float)segments_count * kTwoPi;

            float x = Mathf.Sin(angle);
            float y = Mathf.Cos(angle);

            vertices[vi++] = new Vector3(x, y, 0.0f); // Outer vertex.
            vertices[vi++] = new Vector3(x, y, 1.0f); // Inner vertex.
        }
#endregion

#region Triangles
        int indices_count = (segments_count + 1) * 3 * 2;
        int[] indices = new int[indices_count];

        int vert = 0;
        int idx = 0;
        for (int si = 0; si < segments_count; ++si)
        {
            indices[idx++] = vert + 1;
            indices[idx++] = vert;
            indices[idx++] = vert + 2;

            indices[idx++] = vert + 1;
            indices[idx++] = vert + 2;
            indices[idx++] = vert + 3;

            vert += 2;
        }
#endregion

        mesh.vertices = vertices;
        mesh.triangles = indices;
        mesh.RecalculateBounds();
#if !UNITY_5_5_OR_NEWER
        // Optimize() is deprecated as of Unity 5.5.0p1.
        mesh.Optimize();
#endif  // !UNITY_5_5_OR_NEWER
    }
}
```

4. GVRbutton (Gaze UI)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.Events;

public class GVRButton : MonoBehaviour
{
    public Image imgCircle;
    public UnityEvent GVRClick;
    public float totalTime = 2;
    bool gvrStatus;
    public float gvrTimer;

    // Start is called before the first frame update
    void Update()
    {
        if (gvrStatus)
        {
            gvrTimer += Time.deltaTime;
            imgCircle.fillAmount = gvrTimer / totalTime;
        }
        if(gvrTimer > totalTime)
        {
            GVRClick.Invoke();
        }

    }

    // Update is called once per frame
    public void GvrOn()
    {
        gvrStatus = true;
    }

    public void GvrOff()
    {
        gvrStatus = false;
        gvrTimer = 0;
        imgCircle.fillAmount = 0;
    }
}
```

5. Scene Changer

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class SceneChanger : MonoBehaviour
{
    // Start is called before the first frame update
    public void LoadScene(string scenee2)
    {
       SceneManager.LoadScene(scenee2);
    }

    // Update is called once per frame
    void Update()
    {

    }
}
```