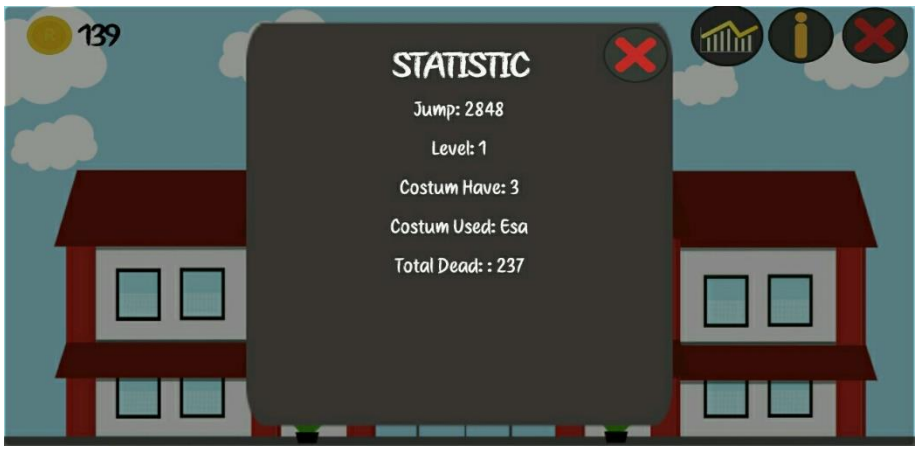
# LAMPIRAN

- Scene Splashcreen


Gambar 5.1 Scene Splashscreen

- Scene Statistic

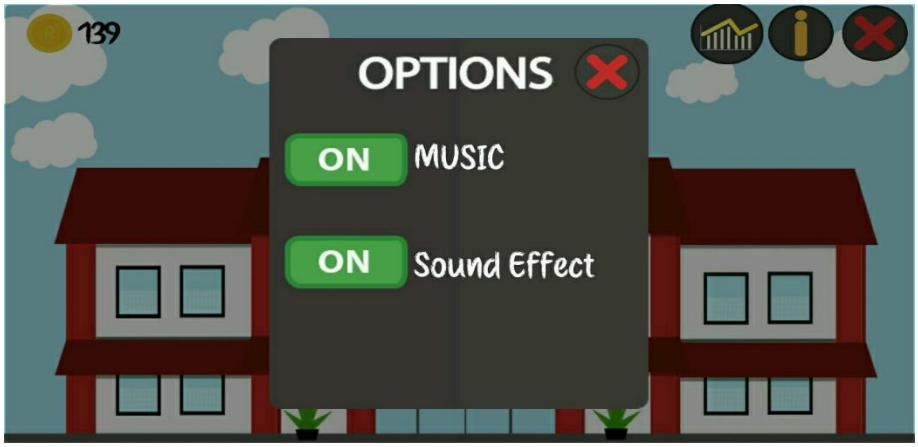
Scene ini menampilkan statistik permainan game Collage Runners




Gambar 5.2 Scene Statistic

- Scene Option

  Scene ini menampilkan pengaturan untuk mengatur musik




Gambar 5.3 Scene Option

- Scene Credit


Gambar 5.4 Scene Option

- Scene Loading

  Scene ini menampilkan loadingscreen sebelum masuk ke playgame


Gambar 5.5 Scene Loading

- Scene Game Over

  Scene ini menampilkan score dan koin yang didapat oleh pemain setelah karakter terjatuh


Gambar 5.6 Scene GameOver

- *Script Random* Balok

```
using System.Collections;

using System.Collections.Generic;
using UnityEngine;

public class PlatformGenerator : MonoBehaviour {

    public GameObject thePlatform;
    public Transform generationPoint;
    public float distanceBetween;

    private float platformWidth;

    public float distanceBetweenMin;
    public float distanceBetweenMax;

    //public GameObject[] thePlatforms;
    private int platformSelector;
    private float[] platformWidths;

    public ObjectPooler[] theObjectPools;

    private float minHeight;
    public Transform maxHeightPoint;
    private float maxHeight;
    public float maxHeightChange;
    private float heightChange;

    private CoinGenerator theCoinGenerator;
    public float randomCoinThreshold;

        // Use this for initialization
        void Start () {
    //platformWidth = thePlatform.GetComponent<BoxCollider2D>().size.x;

    platformWidths = new float[theObjectPools.Length];

    for (int i = 0; i < theObjectPools.Length; i++)
    {
       platformWidths[i] =
         theObjectPools[i].pooledObject.GetComponent<BoxCollider2D>().size.x;
    }

    minHeight = transform.position.y;
         if (PlayerPrefs.GetInt ("levelnow") == 1) {
                  maxHeightPoint.localPosition = new Vector3 (0, 0, 0);
         } else if (PlayerPrefs.GetInt ("levelnow") == 2) {
                  maxHeightPoint.localPosition = new Vector3 (0, 1, 0);
         }else if (PlayerPrefs.GetInt ("levelnow") == 3) {
                  maxHeightPoint.localPosition = new Vector3 (0, 2, 0);
         }else if (PlayerPrefs.GetInt ("levelnow") == 4) {
                  maxHeightPoint.localPosition = new Vector3 (0, 3, 0);
         }else if (PlayerPrefs.GetInt ("levelnow") == 5) {
```

```
        maxHeightPoint.localPosition = new Vector3 (0, 4, 0);
        }else if (PlayerPrefs.GetInt ("levelnow") == 6) {
                maxHeightPoint.localPosition = new Vector3 (0, 5, 0);
        }

        maxHeight = maxHeightPoint.position.y;

theCoinGenerator = FindObjectOfType<CoinGenerator>();
    }

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

if (transform.position.x < generationPoint.position.x)
{
   distanceBetween = Random.Range(distanceBetweenMin, distanceBetweenMax);

   platformSelector = Random.Range(0, theObjectPools.Length);

   heightChange = transform.position.y + Random.Range(maxHeightChange, -
     maxHeightChange);

   if (heightChange > maxHeight)
   {
      heightChange = maxHeight;
   }
   else if (heightChange < minHeight)
   {
      heightChange = minHeight;
   }

   transform.position = new Vector3(transform.position.x +
     (platformWidths[platformSelector] / 2) + distanceBetween,heightChange,
     transform.position.z)

   GameObject newPlatform = theObjectPools[platformSelector].GetPooledObject();

   newPlatform.transform.position = transform.position;
   newPlatform.transform.rotation = transform.rotation;
   newPlatform.SetActive (true);

   if (Random.Range(0f, 100f) < randomCoinThreshold)
   {
      theCoinGenerator.SpawnCoins(new Vector3(transform.position.x,
     transform.position.y + 1f, transform.position.z));
   }

   transform.position = new Vector3(transform.position.x +
     (platformWidths[platformSelector] / 2), transform.position.y,
     transform.position.z);

}
    }
    }
```

- *Script Scroll*

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

[RequireComponent(typeof(Image))]
[RequireComponent(typeof(Mask))]
[RequireComponent(typeof(ScrollRect))]
public class ScrollSnapRect : MonoBehaviour, IBeginDragHandler,
       IEndDragHandler, IDragHandler {

    [Tooltip("Set starting page index - starting from 0")]
    public int startingPage = 0;
    [Tooltip("Threshold time for fast swipe in seconds")]
    public float fastSwipeThresholdTime = 0.3f;
    [Tooltip("Threshold time for fast swipe in (unscaled) pixels")]
    public int fastSwipeThresholdDistance = 100;
    [Tooltip("How fast will page lerp to target position")]
    public float decelerationRate = 10f;
    [Tooltip("Button to go to the previous page (optional)")]
    public GameObject prevButton;
    [Tooltip("Button to go to the next page (optional)")]
    public GameObject nextButton;
    [Tooltip("Sprite for unselected page (optional)")]
    public Sprite unselectedPage;
    [Tooltip("Sprite for selected page (optional)")]
    public Sprite selectedPage;
    [Tooltip("Container with page images (optional)")]
    public Transform pageSelectionIcons;

    // fast swipes should be fast and short. If too long, then it is
       not fast swipe
    private int _fastSwipeThresholdMaxLimit;

    private ScrollRect _scrollRectComponent;
    private RectTransform _scrollRectRect;
    private RectTransform _container;

    private bool _horizontal;

    // number of pages in container
    private int _pageCount;
    private int _currentPage;

    // whether lerping is in progress and target lerp position
    private bool _lerp;
    private Vector2 _lerpTo;

    // target position of every page
    private List<Vector2> _pagePositions = new List<Vector2>();

    // in draggging, when dragging started and where it started
    private bool _dragging;
```

```
private float _timeStamp;
private Vector2 _startPosition;

// for showing small page icons
private bool _showPageSelection;
private int _previousPageSelectionIndex;
// container with Image components - one Image for each page
private List<Image> _pageSelectionImages;

//------------------------------------------------------------------
   ---------
void Start() {
    _scrollRectComponent = GetComponent<ScrollRect>();
    _scrollRectRect = GetComponent<RectTransform>();
    _container = _scrollRectComponent.content;
    _pageCount = _container.childCount;

    // is it horizontal or vertical scrollrect
    if (_scrollRectComponent.horizontal
   && !_scrollRectComponent.vertical) {
        _horizontal = true;
    } else if (!_scrollRectComponent.horizontal &&
   _scrollRectComponent.vertical) {
        _horizontal = false;
    } else {
        Debug.LogWarning("Confusing setting of
   horizontal/vertical direction. Default set to horizontal.");
        _horizontal = true;
    }

    _lerp = false;

    // init
    SetPagePositions();
    SetPage(startingPage);
    InitPageSelection();
    SetPageSelection(startingPage);

    // prev and next buttons
    if (nextButton)
        nextButton.GetComponent<Button>().onClick.AddListener(()
   => { NextScreen(); });

    if (prevButton)
        prevButton.GetComponent<Button>().onClick.AddListener(()
   => { PreviousScreen(); });
   }

//------------------------------------------------------------------
   ---------
void Update() {
    // if moving to target position
    if (_lerp) {
        // prevent overshooting with values greater than 1
        float decelerate = Mathf.Min(decelerationRate *
   Time.deltaTime, 1f);
```

```
_container.anchoredPosition =
       Vector2.Lerp(_container.anchoredPosition, _lerpTo,
       decelerate);
            // time to stop lerping?
            if (Vector2.SqrMagnitude(_container.anchoredPosition -
       _lerpTo) < 0.25f) {
                // snap to target and stop lerping
                _container.anchoredPosition = _lerpTo;
                _lerp = false;
                // clear also any scrollrect move that may interfere
       with our lerping
                _scrollRectComponent.velocity = Vector2.zero;
            }

            // switches selection icon exactly to correct page
            if (_showPageSelection) {
                SetPageSelection(GetNearestPage());
            }
        }
    }

    //------------------------------------------------------------------
       ---------
    private void SetPagePositions() {
        int width = 0;
        int height = 0;
        int offsetX = 0;
        int offsetY = 0;
        int containerWidth = 0;
        int containerHeight = 0;

        if (_horizontal) {
            // screen width in pixels of scrollrect window
            width = (int)_scrollRectRect.rect.width;
            // center position of all pages
            offsetX = width / 2;
            // total width
            containerWidth = width * _pageCount;
            // limit fast swipe length - beyond this length it is
       fast swipe no more
            _fastSwipeThresholdMaxLimit = width;
        } else {
            height = (int)_scrollRectRect.rect.height;
            offsetY = height / 2;
            containerHeight = height * _pageCount;
            _fastSwipeThresholdMaxLimit = height;
        }

        // set width of container
        Vector2 newSize = new Vector2(containerWidth,
       containerHeight);
        _container.sizeDelta = newSize;
        Vector2 newPosition = new Vector2(containerWidth / 2,
       containerHeight / 2);
        _container.anchoredPosition = newPosition;
```

```
        // delete any previous settings
        _pagePositions.Clear();

        // iterate through all container childern and set their
    positions
        for (int i = 0; i < _pageCount; i++) {
            RectTransform child =
    _container.GetChild(i).GetComponent<RectTransform>();
            Vector2 childPosition;
            if (_horizontal) {
                childPosition = new Vector2(i * width -
    containerWidth / 2 + offsetX, 0f);
            } else {
                childPosition = new Vector2(0f, -(i * height -
    containerHeight / 2 + offsetY));
            }
            child.anchoredPosition = childPosition;
            _pagePositions.Add(-childPosition);
        }
    }

    //------------------------------------------------------------------
        ---------
    private void SetPage(int aPageIndex) {
        aPageIndex = Mathf.Clamp(aPageIndex, 0, _pageCount - 1);
        _container.anchoredPosition = _pagePositions[aPageIndex];
        _currentPage = aPageIndex;
    }

    //------------------------------------------------------------------
        ---------
    private void LerpToPage(int aPageIndex) {
        aPageIndex = Mathf.Clamp(aPageIndex, 0, _pageCount - 1);
        _lerpTo = _pagePositions[aPageIndex];
        _lerp = true;
        _currentPage = aPageIndex;
    }

    //------------------------------------------------------------------
        ---------
    private void InitPageSelection() {
        // page selection - only if defined sprites for selection
    icons
        _showPageSelection = unselectedPage != null &&
    selectedPage != null;
        if (_showPageSelection) {
            // also container with selection images must be defined
    and must have exatly the same amount of items as pages
    container
            if (pageSelectionIcons == null ||
    pageSelectionIcons.childCount != _pageCount) {
                Debug.LogWarning("Different count of pages and
    selection icons - will not show page selection");
                _showPageSelection = false;
            } else {
                _previousPageSelectionIndex = -1;
```

```
_pageSelectionImages = new List<Image>();

                // cache all Image components into list
                for (int i = 0; i < pageSelectionIcons.childCount;
      i++) {
                    Image image =
      pageSelectionIcons.GetChild(i).GetComponent<Image>();
                    if (image == null) {
                        Debug.LogWarning("Page selection icon at
      position " + i + " is missing Image component");
                    }
                    _pageSelectionImages.Add(image);
                }
            }
        }
    }

    //-----------------------------------------------------------------
       ---------
    private void SetPageSelection(int aPageIndex) {
        // nothing to change
        if (_previousPageSelectionIndex == aPageIndex) {
            return;
        }

        // unselect old
        if (_previousPageSelectionIndex >= 0) {
            _pageSelectionImages[_previousPageSelectionIndex].sprite
      = unselectedPage;

      _pageSelectionImages[_previousPageSelectionIndex].SetNativeSiz
      e();
        }

        // select new
        _pageSelectionImages[aPageIndex].sprite = selectedPage;
        _pageSelectionImages[aPageIndex].SetNativeSize();

        _previousPageSelectionIndex = aPageIndex;
    }

    //-----------------------------------------------------------------
       ---------
    private void NextScreen() {
        LerpToPage(_currentPage + 1);
    }

    //-----------------------------------------------------------------
       ---------
    private void PreviousScreen() {
        LerpToPage(_currentPage - 1);
    }

    //-----------------------------------------------------------------
       ---------
    private int GetNearestPage() {
```

```
        // based on distance from current position, find nearest page
        Vector2 currentPosition = _container.anchoredPosition;

        float distance = float.MaxValue;
        int nearestPage = _currentPage;

        for (int i = 0; i < _pagePositions.Count; i++) {
            float testDist = Vector2.SqrMagnitude(currentPosition -
    _pagePositions[i]);
            if (testDist < distance) {
                distance = testDist;
                nearestPage = i;
            }
        }

        return nearestPage;
    }

    //------------------------------------------------------------------
       ---------
    public void OnBeginDrag(PointerEventData aEventData) {
        // if currently lerping, then stop it as user is draging
        _lerp = false;
        // not dragging yet
        _dragging = false;
    }

    //------------------------------------------------------------------
       ---------
    public void OnEndDrag(PointerEventData aEventData) {
        // how much was container's content dragged
        float difference;
        if (_horizontal) {
            difference = _startPosition.x -
    _container.anchoredPosition.x;
        } else {
            difference = - (_startPosition.y -
    _container.anchoredPosition.y);
        }

        // test for fast swipe - swipe that moves only +/-1 item
        if (Time.unscaledTime - _timeStamp < fastSwipeThresholdTime
    &&
            Mathf.Abs(difference) > fastSwipeThresholdDistance &&
            Mathf.Abs(difference) < _fastSwipeThresholdMaxLimit) {
            if (difference > 0) {
                NextScreen();
            } else {
                PreviousScreen();
            }
        } else {
            // if not fast time, look to which page we got to
            LerpToPage(GetNearestPage());
        }

        _dragging = false;
```

```
        }

        //-------------------------------------------------------------
           ---------
        public void OnDrag(PointerEventData aEventData) {
            if (!_dragging) {
                // dragging started
                _dragging = true;
                // save time - unscaled so pausing with Time.scale should
           not affect it
                _timeStamp = Time.unscaledTime;
                // save current position of cointainer
                _startPosition = _container.anchoredPosition;
            } else {
                if (_showPageSelection) {
                    SetPageSelection(GetNearestPage());
                }
            }
        }
    }
```

- *Script AdMob*

```
using GoogleMobileAds.Api;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;

public class AdScript : MonoBehaviour {
	private RewardBasedVideoAd adVid;
	public InterstitialAd inAd;
	public BannerView bannerAd;
	// Use this for initialization
	void Start () {

	}
	public AdRequest CreateAdRequest()
	{
	return new AdRequest.Builder()
		.AddTestDevice(AdRequest.TestDeviceSimulator)
		.AddTestDevice("0123456789ABCDEF0123456789ABCDEF")
		.AddKeyword("game")
		.SetGender(Gender.Male)
		.SetBirthday(new DateTime(1985, 1, 1))
		.TagForChildDirectedTreatment(false)
		.AddExtra("color_bg", "9B30FF")
		.Build();
	}
    public void showBannerAd()
    {
	string adID = "ca-app-pub-6256592444826149/4622364155";
	if (this.bannerAd != null)
	{
		this.bannerAd.Destroy();
	}
	 //***For Production When Submit App***
	 //AdRequest request = new AdRequest.Builder().Build();

	 this.bannerAd = new BannerView(adID, AdSize.SmartBanner,
	AdPosition.Bottom);
	bannerAd.LoadAd(this.CreateAdRequest());
    }
	public void showIns(){
	string adUnitId = "ca-app-pub-6256592444826149/5336324800";
	if (this.inAd != null)
	{
		this.inAd.Destroy();
	}
	this.inAd = new InterstitialAd(adUnitId);
	this.inAd.LoadAd(this.CreateAdRequest());
	}
	public void ShowInterstitial()
	{
	if (inAd.IsLoaded())
	{
		inAd.Show();
```

```
            }
            else
            {
                  MonoBehaviour.print("Interstitial is not ready yet");
            }
            }
            public void DestroyBanner(){
            this.bannerAd.Destroy();
            }
            /*private void showVideo(){
            string adIDvid = "ca-app-pub-6256592444826149/5275425683";

            //***For Testing in the Device***
            AdRequest request = new AdRequest.Builder()
                  .AddTestDevice(AdRequest.TestDeviceSimulator)       //
            Simulator.
                  .AddTestDevice("2077ef9a63d2b398840261c8221a0c9b")  //
            My test device.
                  .Build();
            //***For Production When Submit App***
            //AdRequest request = new AdRequest.Builder().Build();
            RewardBasedVideoAd video = new RewardBasedVideoAd(adIDvid);
            }*/
        // Update is called once per frame
        void Update () {

            }
}
```