Scripts

PlayerInput

- Using the new input system
- Detects if an input has been pressed
- Depending on how close the button was pressed to the beat indicator it gives and displays a certain grade

BeatMap

- Handles all the beats
- Moves the beats from the right of the screen to the left

BeatCountUpdater

- Updates beat count text
- Increase per how many beats has passed the beat indicator

BeatGradeUpdater

- Displays grade text when player hits a beat
- Displays on beat indicator

Game Object

- Player Controls
    - Contains BeatDetector which handles inputs
    - BeatDetector contains the PlayerInput script
- BeatMap
    - Parent Object of 16 beats
    - Has the BeatMap script to control the beats
- Canvas
    - Displays Beat Count & Beat Grade
- Background
    - A bunch of sprites that make up the background

Scene

- Only difference is the beat layout and background color
- The both have the same game objects

Scripts

BeatGradeUpdater.cs

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

public class BeatGradeUpdater : MonoBehaviour
{
	public static BeatGradeUpdater Instance { get; private set; }
	TextMeshProUGUI gradeText;

	void Awake()
	{
	if (Instance == null) { Instance = this; }
	else { Destroy(gameObject); }
	}

	private void Start()
	{
	gradeText = GetComponent<TextMeshProUGUI>();
	}

	public void UpdateText(string text)
	{
	gradeText.text = text;
	}
	public void HideText()
	{
	gradeText.enabled = false;
	}

	public void ShowText()
	{
	gradeText.enabled = true;
```

```
        }

        // returns gradeText.enabled
        public bool GetEnabled()
        {
        return gradeText.enabled;
        }

}
```

BeatCountUpdater.cs

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

public class BeatCountUpdater : MonoBehaviour
{
        public static BeatCountUpdater Instance { get; private set; }
        TextMeshProUGUI countText;

        private readonly string s = "Beat count: ";

        void Awake()
        {
        if (Instance == null) { Instance = this; }
        else { Destroy(gameObject); }
        }

        private void Start()
        {
        countText = GetComponent<TextMeshProUGUI>();
        }

        public void UpdateText(int i)
        {
        countText.text = s + i;
        }
}
```

PlayerInput.cs

```
using System.Collections;
using System.Collections.Generic;
using System.Security.Cryptography.X509Certificates;
using UnityEngine;
using UnityEngine.InputSystem;

public class PlayerInput : MonoBehaviour
{
	public static PlayerInput Instance { get; private set; }

	[SerializeField]
	InputAction input;
	[SerializeField]
	[Tooltip("The latest timeDif a beat can score a great")]
	float greatMargin = 0.5f;
	[SerializeField]
	[Tooltip("How far, in seconds, a beat can be detected. Must be greater than
greatMargin")]
	float inputTimeRange = 2f;
	[SerializeField]
	[Tooltip("How long the grade should stay on the screen")]
	float GradeDisplayLength = 1f;

	float GradeDisplayTimer = 0f;

	BeatGradeUpdater bguInstance;
	void OnEnable()
	{
	input.Enable();
	}

	void OnDisable()
	{
	input.Disable();
	}

	private void Awake()
	{
	if (Instance == null) { Instance = this; }
	else { Destroy(gameObject); }
	}
```

```
	private void Start()
	{
	bguInstance = BeatGradeUpdater.Instance;
	if (inputTimeRange < greatMargin) { Debug.LogError("inputTimeRange is smaller than
greatMargin"); }
	}

	void Update()
	{
	if (bguInstance.GetEnabled())
	{
	if (GradeDisplayTimer >= GradeDisplayLength)
	{
		HideZeroGradeDisplayTimer();
	}
	else
	{
		GradeDisplayTimer += Time.deltaTime;
	}
	}
	if (!input.WasPressedThisFrame()) { return; }
	float timeDifferent = BeatMap.Instance.GetTimeDifference();
	//Debug.Log("Time Difference: " + timeDifferent);
	if (timeDifferent > inputTimeRange) { return; }
	GradeHit(timeDifferent);
	}

	private void GradeHit(float timeDifferent)
	{
	Debug.Log("Time Difference:" + timeDifferent);
	if(bguInstance.GetEnabled()){ bguInstance.HideText(); }
	if (timeDifferent > inputTimeRange) { bguInstance.UpdateText("Miss"); }
	else if(GetInput() != BeatMap.Instance.CurrentBeat.direction) {
bguInstance.UpdateText("Wrong"); }
	else if (timeDifferent < greatMargin) { bguInstance.UpdateText("Great");}
	else { bguInstance.UpdateText("Nice"); }
	bguInstance.ShowText();
	BeatMap.Instance.IncrementCurrentBeat();
	}

	BeatMap.Direction GetInput()
	{
	Vector2 v2 = input.ReadValue<Vector2>();
	if (v2.y >= 1) { return BeatMap.Direction.Up; }
```

```
        else if (v2.y <= -1) { return BeatMap.Direction.Down; }
        else if (v2.x <= -1) { return BeatMap.Direction.Left; }
        else { return BeatMap.Direction.Right; }
        }

        public void HideZeroGradeDisplayTimer()
        {
        bguInstance.HideText();
        GradeDisplayTimer = 0f;
        }

}
```

BeatMap.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;

public class BeatMap : MonoBehaviour
{
        public static BeatMap Instance { get; private set; }
        public enum Direction { Up, Down, Left, Right, None };

        [System.Serializable]
        public struct Beat {
        [Min(0)]
        [SerializeField]
        public float timeOccursAt;// { get; private set };

        [SerializeField]
        public Direction direction;// { get; private set};
        }

        [Header("BeatTypes")]
        [SerializeField]
        Sprite Up;
        [SerializeField]
        Sprite Down;
        [SerializeField]
        Sprite Left;
```

```
        [SerializeField]
        Sprite Right;
        [Space(8)]
        [SerializeField]
        private List<Beat> beats;
        List<Transform> beatObjects;
        [Header("Settings")]
        [SerializeField]
        private Vector3 spawnPos = new Vector3(6, 0, 0);
        [SerializeField]
        private Vector3 endPos = new Vector3(-6, 0, 0);
        [SerializeField]
        private Vector3 detectorPos = new Vector3(-4, 0, 0);
        [SerializeField][Min(0)][Tooltip("How long it should take the beat to move from spawn to
beatdetector")]
        private float timeOffset;
        [SerializeField][Min(0)][Tooltip("Delay between game start and when the song starts
playing")]
        private float startDelay;

        public float TimeSinceStart { get; private set; } = 0f;
        public int CurrentBeatIndex { get; private set; } = 0;

        private int LatestBeat = 0;

        public Beat CurrentBeat { get; private set; }

        void Awake()
        {
        if(Instance == null) { Instance = this; }
        else { Destroy(gameObject); }
        }

        void Start()
        {
        InitializeBeatsAndBeatObjects();
        }

        private void Update()
        {
        CurrentBeat = beats[CurrentBeatIndex];
        }

        void FixedUpdate()
```

```
{
TimeSinceStart += Time.fixedDeltaTime;
while (LatestBeat < beats.Count && beats[LatestBeat].timeOccursAt <= TimeSinceStart)
{
LatestBeat++;
}
for (int i = CurrentBeatIndex; i < LatestBeat; i++)
{
if (!beatObjects[i].gameObject.activeSelf) { beatObjects[i].gameObject.SetActive(true); }
MoveBeat(beatObjects[i]);
}
//Debug.Log("Time: " + TimeSinceStart + "\n");
//Debug.Log("CurBeat: " + CurrentBeatIndex + "\n" + "LatestBeat: " + LatestBeat + "\n");
}

private void InitializeBeatsAndBeatObjects()
{
if (beats == null) { Debug.LogError("No Beats"); return; }
beatObjects = new List<Transform>();
GetComponentsInChildren<Transform>(true, beatObjects);
beatObjects.RemoveAt(0);
foreach (Transform t in beatObjects) { t.gameObject.SetActive(false); }
if (beats.Count > beatObjects.Count) { Debug.LogError("Not enough beats"); return; }
beatObjects.RemoveRange(beats.Count, beatObjects.Count - beats.Count);
for (int i = 0; i < beats.Count; i++)
{
ChangeSprite(beatObjects[i].GetComponent<SpriteRenderer>(), beats[i].direction);
}
}
void ChangeSprite(SpriteRenderer sr, Direction d)
{
if(d == Direction.Up)
{
sr.sprite = Up;
}
else if(d == Direction.Down)
{
sr.sprite = Down;
}
else if(d == Direction.Left)
{
sr.sprite = Left;
}
else
```

```
        {
        sr.sprite = Right;
        }
        }

        void MoveBeat(Transform t){
        float speed = (spawnPos.x - detectorPos.x) / timeOffset;
        t.Translate(speed * Time.deltaTime * Vector2.left);
        if (t.position.x <= detectorPos.x)
        {
        IncrementCurrentBeat();
        PlayerInput.Instance.HideZeroGradeDisplayTimer();
        BeatGradeUpdater.Instance.UpdateText("Miss");
        BeatGradeUpdater.Instance.ShowText();
        }
        }

        public void IncrementCurrentBeat()
        {
        beatObjects[CurrentBeatIndex].gameObject.SetActive(false);
        if(CurrentBeatIndex + 1 >= beatObjects.Count) { return; }
        CurrentBeatIndex++;
        BeatCountUpdater.Instance.UpdateText(CurrentBeatIndex);
        CurrentBeat = beats[CurrentBeatIndex];
        }

        public float GetTimeDifference()
        {
        return CurrentBeat.timeOccursAt + timeOffset - TimeSinceStart;
        }

        private void OnDrawGizmos()
        {
        Gizmos.color = Color.green;
        Gizmos.DrawCube(spawnPos, Vector3.one * 0.1f);
        Gizmos.color = Color.red;
        Gizmos.DrawCube(endPos, Vector3.one * 0.1f);
        Gizmos.color = Color.yellow;
        Gizmos.DrawCube(Vector3.zero, Vector3.one * 0.1f);
        }
}
```