LINK - https://grok.com/share/c2hhcmQtMg%3D%3D\_5f1628d0-bb8e-45f9-87ef-2a411c5adfd4  
  
  
# Subway Surfers-like Game Development Roadmap in Unity

This roadmap outlines the development process for creating a Subway Surfers-like endless runner game in Unity, focusing on Player Movement (automatic forward movement, lane switching, jumping, sliding), Main Menu (Play, Settings, Exit options), and Modular Level Design (endless track with obstacles and collectibles). It covers the use of GameObjects, Input systems, Prefabs, Collision handling, Scene Management, and UI Basics, with scripting, optimization, and testing steps.

---

## 1. Project Setup and Initial Configuration

### Objective

Set up the Unity project with necessary assets, scenes, and configurations to support development.

### Implementation Steps

1. \*\*Create a New Unity Project\*\*:

- Use Unity 2022.3 LTS (or later) for stability and modern features.

- Select 3D template for a 3D endless runner with perspective camera.

- Configure project settings: Set target platform to Android/iOS for mobile, or PC for testing.

2. \*\*Organize Project Structure\*\*:

- Create folders: `Scripts`, `Prefabs`, `Scenes`, `Materials`, `Models`, `Textures`, `UI`, `Audio`.

- Import free or purchased 3D assets (e.g., character model, track segments, obstacles, collectibles) from Unity Asset Store or create placeholders using Unity primitives (cubes, spheres).

3. \*\*Set Up Scenes\*\*:

- Create two scenes: `MainMenu` and `GameScene`.

- Save scenes in the `Scenes` folder and add them to Build Settings (`File > Build Settings`).

4. \*\*Install Necessary Packages\*\*:

- Install TextMeshPro via Package Manager for UI text rendering.

- Install Input System package for modern input handling (supports touch and keyboard).

5. \*\*Testing\*\*:

- Build and run the empty project on the target platform to ensure compatibility.

- Verify that scenes load correctly and no errors appear in the Console.

6. \*\*Optimization\*\*:

- Set initial quality settings (`Edit > Project Settings > Quality`) to optimize for mobile (e.g., Fastest or Simple preset).

- Disable unused features like Anti-Aliasing or Shadows for early prototyping.

---

## 2. Player Movement

### Objective

Implement player movement with automatic forward running, lane switching, jumping, and sliding, using GameObjects and the Input System.

### Implementation Steps

1. \*\*Set Up Player GameObject\*\*:

- Create a Player GameObject with a 3D character model or placeholder (e.g., capsule).

- Add a Rigidbody component (set to Kinematic for controlled movement) and a Collider (e.g., Capsule Collider).

- Position the player at `(0, 1, 0)` above the track.

2. \*\*Script Player Movement\*\*:

- Create a `PlayerController` script in the `Scripts` folder.

- Implement automatic forward movement using `Transform.Translate` or velocity.

- Define three lanes (e.g., x = -2, 0, 2) for lane switching.

- Use Unityâ€™s Input System for keyboard (WASD/Arrow keys) and touch input (swipes).

- Implement jumping using `Rigidbody.AddForce` or `Transform.Translate` with gravity simulation.

- Implement sliding by scaling down the playerâ€™s Collider and adjusting position.

```csharp

using UnityEngine;

using UnityEngine.InputSystem;

public class PlayerController : MonoBehaviour

{

private Rigidbody rb;

private float speed = 10f;

private float laneDistance = 2f;

private int currentLane = 0; // -1 (left), 0 (center), 1 (right)

private float jumpForce = 5f;

private bool isGrounded = true;

private bool isSliding = false;

private float slideDuration = 0.5f;

private float slideTimer = 0f;

private Vector3 targetPosition;

void Start()

{

rb = GetComponent<Rigidbody>();

targetPosition = transform.position;

}

void Update()

{

// Automatic forward movement

transform.Translate(Vector3.forward \* speed \* Time.deltaTime);

// Smooth lane transition

transform.position = Vector3.Lerp(transform.position, targetPosition, Time.deltaTime \* 10f);

// Handle sliding

if (isSliding)

{

slideTimer -= Time.deltaTime;

if (slideTimer <= 0)

{

isSliding = false;

transform.localScale = new Vector3(1f, 1f, 1f); // Reset scale

}

}

}

public void OnMove(InputAction.CallbackContext context)

{

if (context.performed)

{

Vector2 input = context.ReadValue<Vector2>();

if (input.x < 0 && currentLane > -1) // Left

{

currentLane--;

}

else if (input.x > 0 && currentLane < 1) // Right

{

currentLane++;

}

targetPosition = new Vector3(currentLane \* laneDistance, transform.position.y, transform.position.z);

}

}

public void OnJump(InputAction.CallbackContext context)

{

if (context.performed && isGrounded)

{

rb.AddForce(Vector3.up \* jumpForce, ForceMode.Impulse);

isGrounded = false;

}

}

public void OnSlide(InputAction.CallbackContext context)

{

if (context.performed && isGrounded && !isSliding)

{

isSliding = true;

slideTimer = slideDuration;

transform.localScale = new Vector3(1f, 0.5f, 1f); // Shrink for slide

}

}

void OnCollisionEnter(Collision collision)

{

if (collision.gameObject.CompareTag("Ground"))

{

isGrounded = true;

}

}

}

```

3. \*\*Configure Input System\*\*:

- In `Project Settings > Input System`, create an Input Action Asset (e.g., `PlayerControls`).

- Add actions: `Move` (Vector2 for left/right), `Jump` (Button), `Slide` (Button).

- Bind keyboard inputs (e.g., LeftArrow/RightArrow for Move, Space for Jump, Ctrl for Slide).

- For touch, map swipe gestures to actions (use `InputSystem.EnhancedTouch` for swipe detection).

4. \*\*Testing\*\*:

- Test movement in Play Mode: verify forward movement is smooth, lane switching is responsive, jumping has correct height, and sliding adjusts the collider.

- Test on target device (mobile/PC) to ensure input responsiveness.

- Debug edge cases (e.g., rapid lane switches, jump-slide combos).

5. \*\*Optimization\*\*:

- Use `Time.deltaTime` for frame-rate-independent movement.

- Cache components (e.g., `Rigidbody`) in `Start()` to avoid repeated `GetComponent` calls.

- Limit physics calculations by using Kinematic Rigidbody and manual collision checks where possible.

---

## 3. Main Menu

### Objective

Create a main menu with Play, Settings, and Exit buttons, using Unityâ€™s UI system and Scene Management for transitions.

### Implementation Steps

1. \*\*Set Up Main Menu Scene\*\*:

- In the `MainMenu` scene, add a Canvas (set to Scale With Screen Size).

- Add UI elements: Buttons (`PlayButton`, `SettingsButton`, `ExitButton`), TextMeshProUGUI for title.

- Position buttons in a vertical layout (e.g., centered, spaced 50 pixels apart).

2. \*\*Script Menu Functionality\*\*:

- Create a `MainMenuController` script in the `Scripts` folder.

- Use `SceneManager` to load the `GameScene` on Play button click.

- Implement Settings (e.g., toggle sound, adjust sensitivity) using PlayerPrefs.

- Implement Exit to quit the application (works in builds, not Editor).

```csharp

using UnityEngine;

using UnityEngine.SceneManagement;

using UnityEngine.UI;

public class MainMenuController : MonoBehaviour

{

public Button playButton;

public Button settingsButton;

public Button exitButton;

public Slider sensitivitySlider;

public Toggle soundToggle;

void Start()

{

playButton.onClick.AddListener(OnPlayButtonClicked);

settingsButton.onClick.AddListener(OnSettingsButtonClicked);

exitButton.onClick.AddListener(OnExitButtonClicked);

// Load saved settings

sensitivitySlider.value = PlayerPrefs.GetFloat("Sensitivity", 1f);

soundToggle.isOn = PlayerPrefs.GetInt("Sound", 1) == 1;

}

void OnPlayButtonClicked()

{

SceneManager.LoadScene("GameScene");

}

void OnSettingsButtonClicked()

{

// Toggle settings panel (optional implementation)

Debug.Log("Settings opened");

}

void OnExitButtonClicked()

{

Application.Quit();

}

public void OnSensitivityChanged(float value)

{

PlayerPrefs.SetFloat("Sensitivity", value);

}

public void OnSoundToggled(bool isOn)

{

PlayerPrefs.SetInt("Sound", isOn ? 1 : 0);

AudioListener.volume = isOn ? 1f : 0f;

}

}

```

3. \*\*Add UI Visuals\*\*:

- Use TextMeshProUGUI for clear, scalable text.

- Add a background image or gradient to the Canvas for visual appeal.

- Ensure buttons are large enough for touch input (minimum 100x50 pixels).

4. \*\*Testing\*\*:

- Test button clicks in Play Mode to ensure scene transitions, settings save/load, and exit functionality work.

- Test on mobile to verify touch input and UI scaling.

- Check PlayerPrefs persistence across sessions.

5. \*\*Optimization\*\*:

- Use a single Canvas to reduce draw calls.

- Optimize UI textures (compress to 512x512 or lower, use sprite atlases).

- Cache UI component references in `Start()`.

---

## 4. Modular Level Design

### Objective

Create an endless track using Prefabs for reusable track segments, with obstacles and collectibles, and handle collisions.

### Implementation Steps

1. \*\*Create Track Segment Prefabs\*\*:

- Design a track segment (e.g., 50 units long) using GameObjects (e.g., cube for ground, walls, obstacles).

- Add obstacles (e.g., barriers, trains) and collectibles (e.g., coins) as child GameObjects.

- Tag obstacles as `Obstacle` and collectibles as `Collectible`.

- Save the segment as a Prefab in the `Prefabs` folder.

- Create 3â€“5 variations of track segments with different obstacle/collectible layouts.

2. \*\*Script Endless Track Generation\*\*:

- Create a `TrackManager` script to spawn and despawn track segments.

- Use a pool of Prefabs to avoid instantiating/destroying at runtime.

- Spawn segments ahead of the player and despawn behind to maintain performance.

```csharp

using UnityEngine;

using System.Collections.Generic;

public class TrackManager : MonoBehaviour

{

public GameObject[] trackPrefabs;

public float segmentLength = 50f;

public int segmentsToKeep = 3;

public Transform player;

private List<GameObject> activeSegments = new List<GameObject>();

private float spawnZ = 0f;

private Queue<GameObject> segmentPool = new Queue<GameObject>();

void Start()

{

// Initialize pool with segments

for (int i = 0; i < segmentsToKeep \* 2; i++)

{

GameObject segment = Instantiate(trackPrefabs[Random.Range(0, trackPrefabs.Length)]);

segment.SetActive(false);

segmentPool.Enqueue(segment);

}

// Spawn initial segments

for (int i = 0; i < segmentsToKeep; i++)

{

SpawnSegment();

}

}

void Update()

{

// Spawn new segment when player approaches end

if (player.position.z > spawnZ - segmentsToKeep \* segmentLength)

{

SpawnSegment();

DespawnSegment();

}

}

void SpawnSegment()

{

GameObject segment;

if (segmentPool.Count > 0)

{

segment = segmentPool.Dequeue();

segment.SetActive(true);

}

else

{

segment = Instantiate(trackPrefabs[Random.Range(0, trackPrefabs.Length)]);

}

segment.transform.position = Vector3.forward \* spawnZ;

activeSegments.Add(segment);

spawnZ += segmentLength;

}

void DespawnSegment()

{

if (activeSegments.Count > segmentsToKeep)

{

GameObject segment = activeSegments[0];

activeSegments.RemoveAt(0);

segment.SetActive(false);

segmentPool.Enqueue(segment);

}

}

}

```

3. \*\*Handle Collisions\*\*:

- In `PlayerController`, add collision detection for obstacles and collectibles.

- On obstacle collision, trigger game over (e.g., reload `GameScene` or show game over UI).

- On collectible collision, increment score and despawn the collectible.

```csharp

void OnCollisionEnter(Collision collision)

{

if (collision.gameObject.CompareTag("Obstacle"))

{

SceneManager.LoadScene("GameScene"); // Game over

}

else if (collision.gameObject.CompareTag("Collectible"))

{

GameManager.Instance.AddScore(10);

collision.gameObject.SetActive(false);

}

}

```

4. \*\*Script GameManager for Score\*\*:

- Create a `GameManager` singleton to track score and game state.

```csharp

using UnityEngine;

public class GameManager : MonoBehaviour

{

public static GameManager Instance;

private int score = 0;

void Awake()

{

if (Instance == null)

{

Instance = this;

DontDestroyOnLoad(gameObject);

}

else

{

Destroy(gameObject);

}

}

public void AddScore(int points)

{

score += points;

// Update HUD (see Section 5)

}

}

```

5. \*\*Testing\*\*:

- Test track spawning: ensure segments appear seamlessly and despawn correctly.

- Test collisions: verify obstacles end the game and collectibles add to the score.

- Test on device to ensure smooth transitions and no lag during spawning.

6. \*\*Optimization\*\*:

- Use object pooling to reuse track segments, obstacles, and collectibles.

- Combine meshes in track segments to reduce draw calls (use `StaticBatchingUtility.Combine`).

- Limit the number of active GameObjects (e.g., keep only 3â€“4 segments active).

---

## 5. In-Game HUD

### Objective

Create a HUD to display score and basic game information, integrated with the GameManager.

### Implementation Steps

1. \*\*Set Up HUD in GameScene\*\*:

- Add a Canvas to the `GameScene` (set to Scale With Screen Size).

- Add TextMeshProUGUI for score display (e.g., top-left corner).

- Add a Pause button (top-right corner).

2. \*\*Script HUD\*\*:

- Create a `HUDController` script to update the score display and handle pause functionality.

```csharp

using UnityEngine;

using TMPro;

using UnityEngine.UI;

public class HUDController : MonoBehaviour

{

public TextMeshProUGUI scoreText;

public Button pauseButton;

void Start()

{

pauseButton.onClick.AddListener(OnPauseButtonClicked);

}

void Update()

{

scoreText.text = $"Score: {GameManager.Instance.GetScore()}";

}

void OnPauseButtonClicked()

{

Time.timeScale = 0f; // Pause game

// Show pause menu (optional)

}

}

```

3. \*\*Testing\*\*:

- Test score updates in Play Mode when collecting items.

- Verify pause button functionality and UI scaling on different resolutions.

- Test on mobile to ensure touch input works for the pause button.

4. \*\*Optimization\*\*:

- Update UI only when necessary (e.g., on score change) using events from `GameManager`.

- Use a single Canvas for HUD to minimize draw calls.

- Compress UI textures and use sprite atlases.

---

## 6. Final Integration and Polish

### Objective

Integrate all components, polish the game, and prepare for deployment.

### Implementation Steps

1. \*\*Integrate Systems\*\*:

- Ensure `PlayerController`, `TrackManager`, `GameManager`, and `HUDController` work together.

- Verify scene transitions from `MainMenu` to `GameScene` and back on game over.

- Add audio: background music for the menu and game, sound effects for jumps, slides, and collisions.

2. \*\*Polish Visuals and Animations\*\*:

- Add animations to the player (e.g., run, jump, slide) using Unityâ€™s Animator component.

- Use particle effects for collectibles and collisions.

- Apply materials/textures to track segments and obstacles for visual appeal.

3. \*\*Testing\*\*:

- Conduct playtesting sessions to balance speed, lane switching, and obstacle frequency.

- Test on multiple devices (Android, iOS, PC) to ensure compatibility and performance.

- Use Unity Profiler to identify and fix performance bottlenecks (e.g., CPU spikes during spawning).

4. \*\*Optimization\*\*:

- Enable batching (static and dynamic) to reduce draw calls.

- Use occlusion culling to hide off-screen track segments.

- Optimize scripts by minimizing `Update` calls and using coroutines for timed events.

5. \*\*Build and Deploy\*\*:

- Build the game for the target platform (`File > Build Settings`).

- Test the final build on devices to ensure no runtime errors.

- Optionally, export to Android/iOS stores or as a standalone executable.

---

## 7. Future Enhancements (Optional)

- \*\*Power-Ups\*\*: Add temporary boosts (e.g., speed, invincibility) as collectibles.

- \*\*Leaderboards\*\*: Integrate with a backend (e.g., Firebase) for online scores.

- \*\*Level Progression\*\*: Increase speed or obstacle density over time.

- \*\*Settings Menu\*\*: Expand with graphics quality options or control remapping.

---

## Tools and Resources

- \*\*Unity Documentation\*\*: Reference for Input System, SceneManager, and UI.

- \*\*Asset Store\*\*: Free/low-cost assets for characters, tracks, and effects.

- \*\*Tutorials\*\*: Brackeys or Unity Learn for endless runner and UI tutorials.

- \*\*Testing Devices\*\*: Use physical devices or emulators (e.g., Android Studio emulator) for mobile testing.