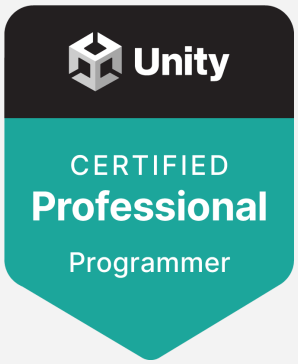

Exam Objectives

Unity Certified Professional: Programmer

The role

Unity Programmers develop interactive content using Unity. Together with the other members of the development team (for example, audio and art professionals), the Unity Programmer brings the vision for the application to life using the features of the Unity Editor, together with the visual and audio assets created by other members of the software development team. The Unity Programmer is a generalist, adept at solving difficult coding problems and responsible for contributing to a broad range of potential technical tasks including integrating art assets, coding the user interface, scripting user interactions and game system rules, implementing application state logic, simulating physics, debugging code, and optimizing performance.

Unity Certified Programmer is a professional certification for entry-level to mid-level programmers and graduating post-secondary students seeking programming roles in a variety of industries. This certification shows potential employers that the holder:

- Employs programming acumen within the context of professional software development processes to create and maintain applications using the Unity Editor
- Has an aptitude for technical processes; is logically oriented and resourceful
- Can be entrusted to handle routine to mid-level programming tasks independently and to work through complex technical challenges with more senior engineers

Job titles for this role

→ Gameplay Programmer
→ Unity Developer/Programmer
→ Software Engineer
→ Software Developer

Prerequisites

This certification was created for programmers who are recent college graduates in game programming, computer science, or related fields; independent learners who have completed two or more years of college-equivalent study or work experience in programming; or, early-career to mid-career professionals who have been using Unity in a work context. Candidates should come to the exam with previous hands-on experience programming interactive applications with Unity by themselves or as part of a cross-functional team, resulting in a completed prototype or technical demo.

Prerequisite experience:

- 2+ years practical experience in video game or 3D interactive programming using Unity
- 2+ years practical experience in computer programming including C#
- Experience in the full software development lifecycle, working from concept through completion
- Understanding of professional applications for software development with Unity including game development, interactive entertainment, and design visualization
- Basic understanding of the visual/3D asset and animation pipeline in Unity, including character and environment setups
- Understanding of professional team software development practices including unit testing and version control
- Knowledge of Unity Services for collaboration, monetization, live operations, and multiplayer
- Understanding of mathematics critical to 3D interactive development, including linear algebra and matrix operations

Note: This certification was developed for Unity 2020 LTS.

Core Skills

(Certification exam topics)

1. Programming core interactions

    1.1. Understand the functionality of MonoBehaviours and ScriptableObjects

    1.2. Implement and configure different methods of inputs as controls in various scenarios

    1.3. Understand how to configure Physics in Unity and the implications of using Rigidbodies, Colliders, and Constraints in different scenarios

    1.4. Understand the functions of Cameras in Unity and how they operate within a scene

    1.5. Understand mathematics critical to 3D interactive development

2. Working in the art pipeline

    2.1. Understand materials, textures, and shaders, and write scripts that interact with Unity's rendering API

    2.2. Understand how lighting functions in Unity and its impact on performance

    2.3. Understand 2D and 3D animation and write scripts that interact with Unity's animation API

    2.4. Understand the impact of art assets on memory usage and performance

2.5. Understand the implications of using different Render Pipelines in various scenarios

3. Developing application systems

3.1. Interpret and analyze scripts for application interface flow such as menu systems, UI navigation, and application settings

3.2. Interpret scripts that allow for user-controlled customization in applications such as configurators

3.3. Analyze and interpret scripts for saving and retrieving application and user data

3.4. Understand how scripts can be used for working with audio and art assets

4. Optimizing for performance and platforms

4.1. Evaluate errors and address performance issues using tools such as the Unity Profiler

4.2. Identify optimizations to address requirements for specific build platforms and/or hardware configurations

4.3. Identify methods for implementing GameObject instantiation, destruction, and management

4.4. Understand how Unity's canvas-based UI system works in relation to creating a performant UI layout

5. Engaging in professional software development teams

5.1. Understand strategies for parallel development within a team

5.2. Demonstrate knowledge of developer testing and its impact on the software development process

5.3. Recognize techniques for structuring scripts for modularity, readability, and reusability

6. Common C# programming constructs

6.1. Understand common C# programming constructs and data structures

6.2. Analyze and interpret examples of asynchronous code

6.3. Understand how delegates, callback systems, and UnityActions are used in code

Sample Questions

Question 1

A Programmer must implement a UI menu system. Each menu consists of a UI Panel and one or more UI Buttons, all of which are parented under a UI Canvas object. The entire UI menu system will be created in a separate scene that is loaded additively. The art style for the panels and buttons should be consistent (color, texture, button transition type, etc.), but the Art Director has not locked these decisions down yet. She would like to work on these settings concurrently with the Programmer's UI work. Her changes would take effect on all new and existing objects in the scene.

What would be the best way for the Programmer to use Unity's functionality to easily create a functional menu system while meeting the requirement of allowing the Art Director to work concurrently (and independently) on the look and feel?

A. Create subclasses for UI.Button and UI.Panel, and set the look and feel values programmatically.

B. Create new button and panel materials, and assign them to all buttons and panels in the scene.

C. Use prefabs for the button and panel, and have the art director modify the prefabs.

D. Write a script to search/replace values in the scene file based on the art director's input.

Question 2

A 3D endless runner game is set along multiple parallel train tracks in a rail yard. The player is always running forward on the tracks, and must avoid oncoming trains by jumping over them or onto an adjacent track. Each new train added to the track is added behind all other trains on that track. However, because trains move toward the player at varying speeds, or not at all, trains occasionally overlap each other, which needs to be fixed.

What is the most performant way to prevent new trains from overlapping with trains already on the same track?

A. When spawning a train on the track, determine a spawn position that will avoid the problem by using the speeds of the new train and the train last placed on that track as well as the point at which the train last placed on the track will de-spawn as it passes the player.

B. When a train is moving, raycast forward from the front of the train, and push any train hit by the raycast forward with the faster train's speed.

C. When spawning a train on the track, add a Rigidbody to it, and then use forces to move trains.

D. When spawning a train on the track, use a BoxCast with a length proportional to the train's speed to ensure that it will not collide with another train until it is behind the Camera.

Question 3

A Programmer is working on a dark and moody room and must create a flickering torch that casts a dancing, eerie shadow over the walls, floor, and ceiling. The Programmer writes these functions on a MonoBehaviour attached to the torch:

```
void Start ( ) {
        Light light = GetComponent<Light>();
        light.lightMapBakeType = LightMapBakeType.Mixed;
        light.type = LightType.Area;
        light.shadows = LightShadows.Soft;
        light.range = 5f;
}
void Update ( ) {
        GetComponent<Light>().intensity =Mathf.PerlinNoise(Time.time, 0);
        }
```

The torch does not cast any light or shadow during runtime. The light is set to the default values in the Unity Editor.

What should the Programmer change for this code to work as required?

A. Set light.lightBakeType to LightmapBakeType.Realtime

B. Set light.range to 10

C. Set light.shadows to LightShadows.Hard

D. Set light.type to LightType.Point

Question 4

A Programmer is developing a mining simulation game where the Player can dig through the ground in search of minerals. In one of the sites, the player can create a tunnel that intersects an existing cave system. The Design Document specifies that any audio that occurs in both the current caves and the new tunnels should have some reverb. The Programmer needs to make sure the user is consistently in the closest cave's ReverbZone.

How should the Programmer manipulate the AudioReverbZone properties to meet these requirements?

A. Increase the reflections to fit the new area.

B. Increase the maxDistance of both ReverbZones so that they touch within the new connecting area.

C. Increase the reverb to accommodate the new area.

D. Increase the decayTime on the new area.

Question 5

While writing a loading function, a Programmer receives a compile error:

error CS1624: The body of `CustomAnalytics.LevelLoading()' cannot be an iterator block because `void' is not an iterator interface type

```
void LevelLoading ( ) {
        AsyncOperation async = SceneManager.LoadSceneAsync( “Level_01” );
        while (!async.isDone) {
                yield return null;
                }
        }
```

What should the Programmer do to fix this error?

A. Change yield return null to yield return WaitForSeconds(0)

B. Change void LevelLoading() to IEnumerator LevelLoading

C. Change SceneManager.LoadSceneAsync(“Level_01”) to Application.LoadLevelAdditiveAsync(“Level_01”)

D. Change while (!async.isDone) to while (!async.allowSceneActivation)

Question 6

A driving game's input system is mapped so that the horizontal input axis controls steering. During testing, it is discovered that some joystick devices register steering input even when the stick is centered.

Which change should be made to the axis in the input system to resolve this issue?

A. Increase Gravity

B. Set Snap to true

C. Increase Deadzone

D. Decrease Sensitivity

Question 7

In an adventure game set on an alien planet, the player must exterminate various life forms. The Player's score increases for each kill. The Design Document states that the score must be linked to a player's account in order to make it retrievable later, even if the player is on a different play session or a different device.

What is the most reliable method for the Programmer to store the score data?

A. Use DontDestroyOnLoad() on the GameObject holding the score data and upload data to a server right before application quits.

B. Save score into PlayerPrefs every time it gets updated and uploaded to a server right before the application quits.

C. Use a static value to store the score data so it will be available in the next play session .

D. Use data serialization to persistently store the score data and upload it to a server.

Correct Answers: C, A, D, B, B, C, D