1.) **A Programmer is working on an Infinite Platformer prototype that focuses on navigating 2D platforms with increasingly more difficult jumps while avoiding enemy NPCs.**
The NPCs will look similar, but not identical. Different colors or slight variance in shape will help the player figure out how to avoid or use them. The player can jump on NPCs to kill them, but more important, the player will bounce when jumping on them, allowing the player extra distance to be able to reach platforms that are further away. At the most difficult levels, the player will have to carefully time their jumping and bouncing off multiple NPCs if they want to make it to the next platform.

The primary goal of the game is the distance traveled. If the player falls off the screen because they miss a platform, they immediately wrap to the top of the screen and continue falling back into the level while losing a life. The player will have enough lives to die a few times this way, but does not die from colliding with enemies. A future iteration of the prototype may explore the idea of earning extra lives if they kill enough enemies. However the main purpose of the enemy is to help the player get further in the level. The player cannot go back, only forward, to the right, but they can stop if they need to.
The player is awarded points for how far they get, and bonus points for enemies killed. Lives and score need to be shown to the player at all times, but not be distracting. The player's movements are going to be physics based, but precise control is required while jumping.

The levels need to be randomly procedurally generated from several different sections. The same section cannot appear next to itself. Enemies should randomly appear on these platforms, with flying enemies filling gaps. Gaps farther than the player can jump will always have a flying enemy for them to bounce off of.

Given these requirements, which of the following game mechanics and features are Must Haves? (Choose all that apply.)

Answers:

1. Physics based rendering

2. Player must always be moving and cannot move backwards

3. Randomly generated levels

4. Points awarded for how long the player stays alive

5. Enemies kill the player by touch

6. Multiple enemy types

7. Player re-spawns where they die

8. Precise air control

Correct Answers:

3. Randomly generated levels

6. Multiple enemy types

8. Precise air control

Explanation:

Explanation Though appearing similar, different colors or slight variance in shape will identify the different types of enemies. Identifying enemies by color or shape is a core game mechanic. It is stated that the player requires precise control while jumping even though their movement is physics based when not jumping. Precise air control is a core game mechanic. The levels need to be randomly procedurally generated from several different sections. This is a core game mechanic. The requirement clearly states that the player does not die from colliding with the enemies. Enemies killing the player by touch is not a core game mechanic. The player earns points by the distance they travel and bonus points for the number of enemies killed. Points are not awarded for how long they stay alive. Points awarded for how long the player stays alive is not a core mechanic. The player's movements are physics based, and nothing was mentioned about the rendering system. Physics based rendering is not a core game mechanic. The requirement states that the player cannot go back, only forward, to the right, but they can stop if they need to. Players having to always be moving and unable to move backwards is not a core game mechanic. The player wraps to the top of the screen and continues falling back into the level while losing a life. Re-spawning the player where they die is not a core game mechanic. References 2D infinite generating level in all direction Platformer Microgame 2D Physics

2.) A Programmer is working on a game prototype of a 2D top down shooter game for mobile platforms. The player controls an alien spaceship and is defending their home world from a human invasion. The movement is not based in physics but transform position updates directly from user input.

The enemy (human) ships shoot two types of projectiles: lasers and missiles. The Player's ship has only lasers. All can have their shots easily detected using a trigger. Scoring is based on the number of human ships destroyed, for which there needs to be a HUD element showing the current count of casualties.

A future iteration of this prototype may introduce small planetoids for the player to avoid and use strategically to hide behind to attack the human enemy.

Which two components will be required to properly prototype this game? (Choose two.)

Answers:

1. RigidBody

2. Canvas

3. Mesh Renderer

4. Particle System

5. Sphere Collider set to Trigger

6. Sprite Renderer

Correct Answers:

2. Canvas

6. Sprite Renderer

Explanation:

Explanation A canvas will be required for the player HUD system to display the accurate count of human ships destroyed. Sprite renderers would be required to display the player and enemy ships as well as the projectiles and in a future iteration, planetoids, if they are further explored. Because this is a 2D game prototype, a sphere collider would not be required. A circle collider would be a better decision because it is designed for 2D. A mesh renderer is not required for 2D games, which typically use sprite graphics. A Rigid Body component is designed for 3D objects. A RigidBody2D set to kinematic would be the right selection, not a RigidBody component. A Particle System is not required for a game prototype. These can be process intensive and need to be planned out by a game designer. Particle Systems should be selectively added into a game after the prototyping phase. References Sphere Collider Canvas Mesh Renderer RigidBody SpriteRenderer Particle System

3.) **A Programmer is building a prototype for a 3D adventure game and needs to implement pickups for the player. The player game object has a RigidBody and a Capsule Collider attached to it, as well as a PlayerController script that contains the code below:**

```
"PlayerController.cs"

void OnCollisionEnter(Collision collision) {
  if(collision.gamebject.tag == "pickup") {
    collision.gameObject.GetComponent<pickup>().PlayerPickedUp();
  }
}

"Pickup.cs"

public enum PickUpType { AMMO, HEALTH, WEAPON };
...

[SerializeField]
private PickupType m_pickupType;

...

public void PlayerPickedUp() {
 Destroy(this.gameObject, 1);
}
```

The Pickup game object is tagged pickup and has a collider and a pickup script attached to it with a public method named PlayerPickedUp() in it as well as a public Enumeration named PickUpType declared just outside the Pickup Class. A private variable marked as set is set in the inspector for the pickup. It can be used to tell the player what type of pickup was obtained.

Currently there are only three pickup types (ammunition, health packs, and a generic weapon).

When the player collides with the pickup, it is successfully destroyed.

What does the Programmer need to do to ensure that the player can successfully pick up a pickup in the game and know which of the pickup types was collected?

Answers:

1. Change the PlayerPickedUp method in the Pickup script to return the type of the pickup before destroying the game object and also update the OnCollisionEnter method in the PlayerController to catch and then do something with the returned result.

2. Implement another method to return the type of pickup before calling the PlayerPickedUp() method on collision.

3. Change the PlayerPickedUp method in the Pickup script to return the type of the pickup before destroying the game object.

4. Query the Pickup class for the m_pickupType directly before calling the PlayerPickedUp method in the Pickup class and having the game object destroy itself.

5. Change the PlayerPickedUp method to the following: public PickUpType PlayerPickedUp() { return this; Destroy(this.gameObject); }

Correct Answers:

1. Change the PlayerPickedUp method in the Pickup script to return the type of the pickup before destroying the game object and also update the OnCollisionEnter method in the PlayerController to catch and then do something with the returned result.

Explanation:

Explanation The Programmer should change the PlayerPickedUp method in the Pickup script to return the type of the pickup before destroying the game object and also update the OnCollisionEnter method in the PlayerController to catch and then do something with the returned result. By changing the PlayerPickedUp method to return a PickUpType and having that method return the m_pickupType variable, as well as having the call to PlayerPickedUp catch the return, the Programmer will be able to determine the type for pickup that was set in the inspector for each specific pickup. The Programmer cannot query the m_pickupType variable directly because it is set to private. The Programmer should not change the PlayerPickedUp() method to return the type before Destroy is called. This will cause the pickups to no longer disappear or be destroyed. The Programmer should not query the Pickup class for the m_pickupType directly before calling the PlayerPickedUp method in the Pickup class. Adding an additional method to query the type of pickup is not efficient and adds unnecessary complexity to the coupling of these two objects. The Programmer should not change the PlayerPickedUp method in the Pickup script to return the type of the pickup before destroying the game object. If the Programmer only changes the PlayerPickedUp method in the Pickup class, they will still not know what type of pickup was returned. The Programmer needs to catch the returned object to see what m_pickupType was set to. References Object.Destroy MonoBehaviour.OnCollisionEnter(Collision) SerialzeField

4.) Which method is the correct way to log messages in the console during runtime with C#?

Answers:

1. Debug.Log()

2. Console.Log()

3. System.Writeln()

4. System.Out()

Correct Answers:

1. Debug.Log()

Explanation:

Explanation Debug.Log() is the best way to send messages to the console during runtime. You should not use Console.Log() in Unity with C#. Console.Log() is a JavaScript method. You should use Debug.Log(). You should not use System.Writeln() in Unity with C#. System.WriteLn() is a C++ method. You should use Debug.Log(). You should not use System.Out() in Unity with C#. System.Out() is a C++ method. You should use Debug.Log(). References Debug.Log

5.) **A Game Programmer is prototyping a patrolling NPC enemy that is supposed to walk around on a platform. The NPC keeps falling off the platform. Currently the NPC uses raycasts to detect the ground ahead of it. Which method is best to help debug the NPCs behavior and see why it keeps falling off the platform?**

Answers:

1. Gizmos.DrawRay to visually inspect the raycasts

2. Debug.Log to write out the raycast position and direction

3. Debug.Log to write out the raycast start and end points

4. Debug.LogWarning to write out the results of the raycast when the edge of a platform is found

5. Gizmos.DrawWireSphere to visually inspect the raycast range

Correct Answers:

1. Gizmos.DrawRay to visually inspect the raycasts

Explanation:

Explanation Gizmos.DrawRay should be used to visualize how the ray is behaving in the scene. It will help to determine if the raycasts are positioned and pointing in the right direction. Gizmos.DrawWireSphere only draws a wireframe sphere given a position and a radius. It would not help to determine the direction a raycast is pointing. Debug.Log would print out vector3 values in plain text. These would be very difficult to interpret in plain text and would not help debugging this type of issue. Debug.LogWarning is designed to log warnings about your code to the console or a log file. It is not intended to debug this type of issue. References Gizmos.DrawRay Gizmos.DrawWireSphere Debug.Log Debug.LogWarning

6.) A Programmer is developing a prototype for a 3D adventure game,

When the player collides with the pickups, the pickups are moving just before they are destroyed and collected.

What should the Programmer check and correct first? (Choose two.)

Answers:

1. The Programmer should check that the pickup item has its RigidBody position frozen on the x, y, and z axis.

2. The Programmer should check that the RigidBody is set to kinematic.

3. The Programmer should check that the pickup item has its RigidBody rotation frozen on the x, y, and z axis.

4. The Programmer should check that the pickup item has a RigidBody attached to it.

5. The Programmer should check that the Collider on the Pickup is set to be a trigger and the method OnTriggerEnter(Collider) is being used and not OnCollisionEnter(Collision).

Correct Answers:

2. The Programmer should check that the RigidBody is set to kinematic.

5. The Programmer should check that the Collider on the Pickup is set to be a trigger and the method OnTriggerEnter(Collider) is being used and not OnCollisionEnter(Collision).

Explanation:

Explanation The Programmer should check that the Collider on the Pickup is set to be a trigger and the method OnTriggerEnter(Collider) is being used and not OnCollisionEnter(Collision). When working with game objects that you want to collide with but not have them react to physics, you need to use the OnTrigger methods and have the collider set to is Trigger. The Programmer should also check that That the RigidBody is set to kinematic. Kinematic RigidBodies do not react to collisions or suffer gravity effects, but enable the trigger to detect any collider. The position freeze of the RigidBody in not the issue. The physics system for the pickup object itself should be disabled so that a trigger fires but the collision is still registered. The rotation freeze of the RigidBody in not the issue. The physics system for the pickup object itself should be disabled so that a trigger fires but the collision is still registered. The Pickup item already has a RigidBody attached to it or it would not be moving on collisions. It would not detect or respond to other game objects. References Collider.OnTriggerEnter(Collider) Collider.OnCollisionEnter(Collision) Ridigbody.isKinematic Rigidbody.freezeRotation Position Constraints

7.) A Programmer is working on a prototype for a physics game that adds thrust to a game object's Rigidbody component. The variable rb is a reference to the Game Object's Rigidbody. The thrust variable has been set in the inspector and is of float type.
In which method should the Programmer place the following code to ensure that it works correctly?
rb.AddForce(transform.forward * thrust);

Answers:

1. Update()

2. LateUpdate()

3. EarlyUpdate()

4. FixedUpdate()

Correct Answers:

4. FixedUpdate()

Explanation:

Explanation The Programmer should place the code in the FixedUpdate method. The FixedUpdate method is Unity's frame-rate independent method for physics calculations. The Programmer should not use the Update method. It is called every frame and should not be used for physics calculations like applying force to Rigidbodies. The Programmer should not use the LateUpdate method. It fires after all Update methods have fired and is best suited for things like follow cameras. The Programmer should not use the EarlyUpdate method. It fires before all Update methods are called and is currently classed as experimental. References MonoBehaviour.FixedUpdate() MonoBehaviour.Update() MonoBehaviour.LateUpdate() EarlyUpdate

8.) A Programmer needs to clean up the runtime spawning of fireball prefabs into a 3rd-person action/adventure game. The fireballs are currently being instanced by a line of code that reads as follows:

GameObject fireball = Instantiate(m_fireballPrefab, m_spawnLocation, Quaternion.identity);

The m_fireballPrefab variable has been configured in the inspector to point to the Fireball prefab. The m_spawnLocation variable has been configured in the inspector to the correct position.

The m_fireballContainer variable is set in the inspector to the "Fireballs" Game Object in the hierarchy.

Now the Programmer needs to ensure that the newly spawned prefab is attached to a parent "Fireballs" Game Object in the project hierarchy to help keep the hierarchy neat and orderly.

Which code should the Programmer use?

Answers:

1. fireball.transform.parent = m_fireballContainer.transform;

2. fireball = m_fireballContainer;

3. fireball.transform.parent = m_fireballContainer;

4. fireball.parent = m_fireballContainer;

Correct Answers:

1. fireball.transform.parent = m_fireballContainer.transform;

Explanation:

Explanation The Programmer should use the following code: fireball.transform.parent = m_fireballContainer.transform; This correctly sets the transform of the m_fireballContainer in the hierarchy to be the parent transform of the newly spawned fireball Game Object. The Programmer should not use the following code: fireball = m_fireballContainer; fireball is a Game Object and not a transform, and m_fireballContainer is a Game Object and not the transform of the Game Object. The Programmer should not use the following code: fireball.parent = m_fireballContainer; fireball.parent is unknown to Unity, and this will throw an exception. The Programmer should not use the following code: fireball.transform.parent = m_fireballContainer; The m_fireballContainer is a Game Object and not the transform of the Game Object. References Transform.parent

9.) A Programmer is building a 3D action game that requires a lot of explosions in quick repetition during some points of game play. It has been decided that the Programmer should implement a thread safe object pool to instantiate the exploding prefabs at runtime.

The Programmer creates the generic object pool class shown below so they can re-use it for other objects if needed. They know that this way, all they need to do is create an empty class that inherits from ObjectPool, attach that class to a game object in the inspector, and set the prefab and number of objects to pool. The Programmer has a prefab of the explosion and is trying to get the pool to work but it throws errors.

The Programmer realizes that they need to warm the pool with an Awake() method.

Which Awake() method should the Programmer implement?

```
using UnityEngine;
using System.Collections.Generic;
public class ObjectPool<T> : MonoBehaviour where T : MonoBehaviour {
  public T m_prefab;
  public int m_size;
  private List<T> m_freeList;
  private List<T> m_usedList;
  public T Get() {
    var numFree = m_freeList.Count;
    if (numFree == 0)
      return null;
    var pooledObject = m_freeList[numFree - 1];
    m_freeList.RemoveAt(numFree - 1);
    m_usedList.Add(pooledObject);
    return pooledObject;
  }
  public void ReturnObject(T pooledObject) {
    Debug.Assert(m_usedList.Contains(pooledObject));
    m_usedList.Remove(pooledObject);
    m_freeList.Add(pooledObject);
    var pooledObjectTransform = pooledObject.transform;
    pooledObjectTransform.parent = transform;
    pooledObjectTransform.localPosition = Vector3.zero;
    pooledObject.gameObject.SetActive(false);
  }
}
```

Answers:

1. public void Awake() { m_freeList = new List(m_size); m_usedList = new List(m_size); }

2. public void Awake() { m_freeList = new List(m_size); m_usedList = new List(m_size); for (var i = 0; i < m_size; i++) { var pooledObject = Instantiate(m_prefab, transform); pooledObject.gameObject.SetActive(false); m_freeList.Add(pooledObject); } }

3. public void Awake() { m_freeList = new List(m_size); m_usedList = new List(m_size); for (var i = 0; i < m_size; i++) { var pooledObject = Instantiate(m_prefab, transform); m_freeList.Add(pooledObject); } }

4. public void Awake() { for (var i = 0; i < m_size; i++) { var pooledObject = Instantiate(m_prefab, transform); pooledObject.gameObject.SetActive(false); m_freeList.Add(pooledObject); } }

Correct Answers:

2. public void Awake() { m_freeList = new List(m_size); m_usedList = new List(m_size); for (var i = 0; i < m_size; i++) { var pooledObject = Instantiate(m_prefab, transform); pooledObject.gameObject.SetActive(false); m_freeList.Add(pooledObject); } }

Explanation:

Explanation The Programmer should implement the following Awake() method: public void Awake() { m_freeList = new List<T>(m_size); m_usedList = new List<T>(m_size); for (var i = 0; i < m_size; i++) { var pooledObject = Instantiate(m_prefab, transform); pooledObject.gameObject.SetActive(false); m_freeList.Add(pooledObject); } } This will initialize the free and used lists properly and instantiate the correct number of prefabs into the pool for use. The Programmer should not implement the following Awake() method: public void Awake() { for (var i = 0; i < m_size; i++) { var pooledObject = Instantiate(m_prefab, transform); pooledObject.gameObject.SetActive(false); m_freeList.Add(pooledObject); } } This does not properly initialize the free and used lists before populating the pool. The Programmer should initialize the lists before populating them. The Programmer should not implement the following Awake() method: public void Awake() { m_freeList = new List<T>(m_size); m_usedList = new List<T>(m_size); for (var i = 0; i < m_size; i++) { var pooledObject = Instantiate(m_prefab, transform); m_freeList.Add(pooledObject); } } This does not disable the prefabs in the free list. In the case of explosions, that would cause the explosions to explode in the pool, which is not the desired outcome. The Programmer should not implement the following Awake() method: public void Awake() { m_freeList = new List<T>(m_size); m_usedList = new List<T>(m_size); } This initializes the lists for the object pool but does not populate them with the prefab. References Type-safe object pool for Unity TypeSafeObjectPool

10.) A Programmer is working on a prototype for a mobile platformer game and needs to build a script that will generate all of the platforms in the game. How should the Programmer implement the logic for this script?

Answers:

1. The Programmer should create a single level generator script and attach it to an empty Game Object in the hierarchy of the scene.

2. The Programmer should attach a script to a prefab that is loaded in the game from the game manager.

3. The Programmer should build base classes that do not use MonoBehaviour.

4. The Programmer should separate all of the logic for creating the platforms into small discrete pieces that are connected through a controlling script that is not attached to anything.

Correct Answers:

1. The Programmer should create a single level generator script and attach it to an empty Game Object in the hierarchy of the scene.

Explanation:

Explanation The Programmer should create a single level generator script and attach it to an empty Game Object in the hierarchy of the scene to keep all of the level generation logic in a single place that is easy to access. The Programmer should not embed generators or managers in prefabs unless there is a strong case to do this. Embedding scripts in prefabs will require additional processing by Unity, so the performance will suffer. The Programmer should not separate all of the logic for creating the platforms into small discrete pieces that are connected through a controlling script that is not attached to anything. The logic for platform generation is best suited in a single location. The Programmer should not build base classes that do not use MonoBehaviour if they are attached directly to Game Objects in the hierarchy. MonoBehaviour allows the Programmer to leverage the built-in functions that Unity provides such as Start, and Update. Generator and Manager scripts benefit from the use of the MonoBehaviour base class and should be used. References Game Managers

11.) A Programmer is given the following code as a starting point for a GameManager script:

```
public class GameManager : MonoBehaviour
{
   public BoardManager m_boardScript;


   private int m_level = 1;


   void Awake()
   {
      m_boardScript = GetComponent<BoardManager>();
      InitGame();
   }


   void InitGame()
   {
      m_boardScript.SetupScene(m_level);
   }
}
```

What can the Programmer determine about the game from this code? (Choose two.)

Answers:

1. GameManager has a method named SetupScene that takes a single integer value to represent the level that should be set up.

2. BoardManager is a singleton.

3. The private variable m_level will allow the Programmer to test setup of any level that exists if it has been properly numbered.

4. There is a method in the BoardManager named SetupScene that takes a single integer value to represent the level the Board Manager should set up.

5. The use of a Board Manager is redundant and the Game Manger should do all the work to keep the code in one place.

Correct Answers:

3. The private variable m_level will allow the Programmer to test setup of any level that exists if it has been properly numbered.

4. There is a method in the BoardManager named SetupScene that takes a single integer value to represent the level the Board Manager should set up.

Explanation:

Explanation The Programmer can determine that the BoardManager script has a method named SetupScene that takes a single integer variable that represents the level the BoardManager should set up. The Programmer can also determine that changing the value of the private integer variable m_level should cause the BoardManager to set up the level identified. The Programmer cannot accurately determine if the BoardManager is a singleton. If BoardManager was a correctly written singleton, GetComponent would not be called. Instead, m_boardScript would be set via assignment of an instance variable from within the BoardManager class. The Programmer cannot determine that the BoardManager is redundant. In fact, better maintainable and readable code comes from keeping the purpose of Scripts singular and modular in nature. The Programmer cannot determine that the GameManger has a method named SetupScene. The BoardManager has this method, not the GameManager. References Writing the Game Manager

12.) A Programmer is building a 3D FPS game that heavily uses delegates in most of the code base. There are memory leaks caused by improperly removing game objects from the levels at runtime, and systems are running out of memory.
The Programmer knows that the memory issues are caused because of all the delegates and needs to ensure that they are cleaned up properly before the objects are removed.
What should the Programmer do before removing the objects from the levels so that the memory leaks are fixed?

Answers:

1. Unsubscribe from any UnityEvent that an object may be listening to before removing it from the level.

2. Unsubscribe from any UnityAction that an object may be listening to before removing it from the level.

3. Set any UnityAction that an object may be listening to as null before removing it from the level.

4. Set any UnityEvent that an object may be listening to as null before removing it from the level.

Correct Answers:

1. Unsubscribe from any UnityEvent that an object may be listening to before removing it from the level.

Explanation:

Explanation The Programmer should unsubscribe from any UnityEvent that an object may be listening to before removing it from the level. Unsubscribing from any UnityEvents an object is listening to is required before removing a game object from a scene. Failing to do so will cause memory leaks. The Programmer should not set any UnityEvents to null or any other value. You do not Set UnityEvents, you Subscribe or Unsubscribe from them. The Programmer should not set any UnityActions to null before removing the object from the level. UnityActions are used to dynamically call multiple functions within a class. You do not need to set these to null before removing an object from the scene. The Programmer should not unsubscribe from any UnityActions. UnityActions are used to dynamically call multiple functions within a class. You do not need to unsubscribe from them before removing an object from a scene. References UnityEvent UnityAction

13.) A Programmer is working on a 3rd-person action/adventure game for PC and must implement a GPU instanced particle system. The Programmer has been supplied with the mesh that the particle system must render.
The Programmer needs to implement a Particle system for GPU Instancing.
What should the programmer do? (Choose two).

Answers:

1. Use a shader for the renderer material that supports GPU instancing.

2. Set the Particle system's renderer mode to Billboard.

3. Set the Particle system's renderer mode to Mesh.

4. De-select the Enable GPU Instancing checkbox.

5. The Programmer does not need to perform any additional actions. GPU Instancing is the default behavior for Particle systems in Unity.

Correct Answers:

1. Use a shader for the renderer material that supports GPU instancing.

3. Set the Particle system's renderer mode to Mesh.

Explanation:

Explanation The Programmer needs to set the Particle system's renderer mode to Mesh and ensure that they are using a shader for the render material that supports GPU instancing. The default behavior of the Unity Particle System is Billboard particles. The Programmer should not set the Particle System's renderer mode to Billboard. Mesh mode is required to use the GPU for Instancing in the particle system. The Programmer should not de-select the Enable GPU Instancing checkbox. This selection is needed to ensure that Unity uses the GPU for instancing mesh particle systems. References Particle System GPU Instancing

14.) A Programmer is building a 3rd-person racing game and needs to use a Particle System to emit dust clouds from the cars that trail the vehicle for a short time (.25 to .5 seconds).
Which two Particle System modules should the Programmer use to achieve the desired effect? (Choose two.)

Answers:

1. Noise module

2. Inherit Velocity module

3. Trails module

4. Sub Emitters module

5. Triggers module

Correct Answers:

2. Inherit Velocity module

3. Trails module

Explanation:

Explanation The Programmer should use the Trails module in particle mode to have the dust particles trail the vehicles. The Programmer should use the Inherit Velocity module to emit the dust clouds from the car. This effect is very useful for emitting particles from a moving object when particles should be moving at a percentage of the speed of the object they appear to come from. The Programmer should not use the Sub Emitters module for a simple dust trail that does not require different effects or textures to emulate. The Programmer should not use the Triggers module. The Triggers module should be used when you need to implement a callback on collisions from the Particle System, which is not required in this case. The Programmer should not use the Noise module. While the Noise module is a simple and effective way to create interesting patterns and effects like smoke rising and swirling from a campfire, the Inherit Velocity module will do a better job of rendering the effect you want. References Trails module Inherit Velocity module Sub Emitters module Triggers module Noise module

15.) A Programmer is tasked with improving the clouds in a 3rd-person action/adventure game for PC. The play testers commented that the current particle system for the clouds looks weak and lacks volume.
The current Particle System is using a simple texture for the cloud Particle System.
What should the Programmer do to improve this Particle System?

Answers:

1. Change the Pivot point for rotating the texture to make the clouds appear more random from one another in final shape.

2. Using the Render module, switch the system from billboards to Mesh and use an actual Mesh object for the clouds.

3. Use the Flip property of the Render module to eliminate some of the repetitive patterns from the Particle System.

4. Increase the Max-Particle size number of the Render module until the cloud looks like it takes up more volume.

Correct Answers:

2. Using the Render module, switch the system from billboards to Mesh and use an actual Mesh object for the clouds.

Explanation:

Explanation The Programmer should use the Render module to switch the particles from billboard textures to Mesh objects. This is the best way to improve the sense of volume for the clouds. The Programmer should not increase the Max-Particle size of the Render module. This will result in bigger clouds that still lack volume. The Programmer should not flip some of the particles to remove some of the repetitive patterns from the Particle System. This will not make the clouds appear to take up more volume, but it will help to reduce any repetitive patterns. The Programmer should not change the Pivot point for rotating the texture. This will not make the clouds appear to take up more volume, but it will help to reduce any repetitive patterns. References Renderer module

16.) A Programmer wants to simulate a soft shadow under his characters in a 3D adventure game.
The Programmer needs these shadows to look good but also be as efficient as possible.
Which method is the most efficient for the Programmer to generate a soft shadow under a character?

Answers:

1. Use a spotlight with Soft Shadows enabled.

2. Use a light that does not affect geometry, but uses the Glow effect.

3. Use a light that does not affect geometry, but uses the Halo effect.

4. Use a Projector to project a shadow texture.

Correct Answers:

4. Use a Projector to project a shadow texture.

Explanation:

Explanation The Programmer should use a Projector to project a shadow texture. A Projector can paint a texture on the ground and is more efficient at suggesting a soft shadow than using a light. Shadows are the number one use of Projectors in Unity. The Programmer should not use the Glow effect. The Glow effect uses Bloom, which creates fringes of light extending from the borders of light areas. This is not suitable for shadows. The Programmer should not use the Halo effect. Halos are light areas around light sources. This is not suitable for shadows. The Programmer should not use a spotlight with Soft Shadows enabled. A spotlight has high rendering overhead and should be used sparingly. This is not an efficient way to cast shadows for characters. References Projector Image Effect reference Bloom Halo Spotlight

17.) What has the highest impact on performance and should be eliminated wherever possible to improve performance?

Answers:

1. Baking in static lighting

2. Texture compression

3. Realtime shadows

4. Draw call batching

Correct Answers:

3. Realtime shadows

Explanation:

Explanation Realtime shadows are nice, but they can have a high impact on performance, both in terms of extra draw calls for the CPU and extra processing on the GPU. These should be eliminated whenever possible. Using compressed textures decreases the size of your textures. This is a performance improvement and should be done whenever possible. Baked in static lighting not only looks better, but it can perform two to three times better. This is a performance improvement and should be done whenever possible. Draw call batching is a great way to improve performance. This should be done whenever possible. References Optimizing graphics performance Light troubleshooting and performance Draw call batching

18.) A Programmer is working on a 3D action / adventure game that includes a small shipping and receiving area in a building where the player is required to load several objects onto the back of a truck within a time limit.
The player will drive a forklift and pick up the items to load onto the truck. The forklift has two forks, and the player can either attempt to pick up items using one or both forks. Additionally, the player can choose to wedge items between the forks.
The Programmer needs to set up the RigidBody and Colliders for the forklift.
What should the Programmer do to setup and handle collision for the forks to accomplish this functionality without causing any performance issues?

Answers:

1. Create two Box colliders, one for each fork, and use a single RigidBody component on the same object.

2. Create a single Mesh collider that mimics the shape of both forks and use a single RigidBody component on the same object.

3. Create two Mesh colliders, one for each fork, and parent them under a single RigidBody.

4. Create three Box colliders, one for each fork and one set as a trigger, to act as the empty space between the forks for wedging objects, and parent them to a single RigidBody.

5. Create a single Box collider that covers the shape of both forks and use a single RigidBody component on the same object.

Correct Answers:

1. Create two Box colliders, one for each fork, and use a single RigidBody component on the same object.

Explanation:

Explanation The Programmer should create two Box colliders, one for each fork, and use a single RigidBody component on the same object. The concave geometry of the forklift's forks and the requirement to wedge items between them means the simple Box Colliders under one Rigidbody will perform the best for this gameplay. The Programmer should not use one or two Mesh colliders. Mesh colliders are performance heavy, and the physics system needs to do extensive checking on Mesh colliders. While the functionality could be implemented, it would come at a performance cost. The Programmer should not use a single Box collider. A single Box collider would not allow the player to wedge objects between the two forks. This would not produce the requested functionality. The Programmer should not use three Box colliders. Adding a third Box collider as a trigger would be an unnecessary resource, as would the dedicated child objects for the colliders. References Rigidbody BoxCollider Mesh Collider

19.) A Programmer is working on a simple 2D game for mobile and wants to ensure that the game runs fast and is optimized for mobile devices.
What should the Programmer do to optimize for the mobile platforms? (Choose two.)

Answers:

1. Use Object Pooling.

2. Use RigidBodies limited to 2 dimensions.

3. Make sure that all objects are destroyed when out of sight and only instanced as they need to be.

4. Make use of OnGui() for the HUD.

5. Write a custom physics system for the game.

Correct Answers:

1. Use Object Pooling.

5. Write a custom physics system for the game.

Explanation:

Explanation The Programmer should write a custom physics system for the game. While it is more complicated and limited, it will be much faster than the Unity Physics system providing it is a simple game. The Programmer should also use Object Pooling. Instantiate and Destroy are process-intensive methods, and Object Pooling will dramatically improve performance. The Programmer should not Use RigidBodies limited to 2 dimensions. This is a waste of resources and the reason for the RigidBody2D component. The Programmer should not make sure that all objects are destroyed when out of sight. Instantiate and Destroy are process-intensive, so Object Pooling is recommended. The Programmer should not use OnGui() for the HUD or HUD elements. Canvas elements are much better for HUDs and performance. References Scripting and Gameplay Methods

20.) A Programmer is building a game that will be available for multiple platforms, including mobile platforms.
The Programmer needs to be able to provide assets that provide the same gameplay for all variants of the game regardless of the platform.
What should the Programmer implement to accomplish this?

Answers:

1. Take the high-end hardware into account and target for performance on that platform.

2. Use AssetBundle variants to allow for models, graphics, and scripts that are better suited for both low-end and high-end platforms.

3. The Programmer should do a SOAK test to determine which specs to target and eliminate the systems that run out of resources early from the list of supported platforms.

4. Use platform dependent compilation in the game.

5. Take the low-end hardware into account and target for performance for those platforms.

Correct Answers:

2. Use AssetBundle variants to allow for models, graphics, and scripts that are better suited for both low-end and high-end platforms.

Explanation:

Explanation The Programmer should implement AssetBundles for variants. This is the best way to target a wide range of hardware with varying degrees of performance in a single game. By offering the same assets with different size and complexity of files, the Programmer can offer a game that performs consistently across a variety of hardware. The Programmer should not take the low-end hardware into account and reduce the performance of the game to match the lowest specs. This will have an adverse effect on players using newer hardware. The Programmer should not take only the high-end of performance into account. That will make the game unplayable on low-end systems and ruin the player experience. The Programmer should not use platform dependent compilation. This is only useful to execute different code for different platforms and would not help at all with graphic resolutions or poly count in mesh files. This would be hard to maintain. The Programmer should not use A SOAK test to identify the platforms that run out of memory for the purpose of eliminating them from the list of platforms the game is to run on. A SOAK test is designed to run your game for a long time to determine the patterns of performance. It will not help you to make decisions for platforms. A Game Designer will make the decisions which platforms the game should run on. References Assets, Resources and AssetBundles Platform-specific Best practice guides Platform dependent compilation