**COMSATS University Islamabad, Abbottabad Campus**  
**Department of Computer Science**  
**MIDTERM EXAMINATION SPRING 2023**

**Class:** BSE-7A/7B  
**Subject:** Game Development  
**Time Allowed:** 90 Minutes  
**Instructor:** Mr. Ibtisam Gul  
**Max Marks:** 50

**Instructions:**

1. Clearly mention the question number and part number in your answer script.
2. Don't attempt extra questions, as only the first one will be marked.
3. Follow the NOTE given in each question.

**Q.1 Fundamental concepts of Game Development.**

*(NOTE: Do any 1 from long questions and any 2 from short questions)*  
**(Total Marks: 20)**

**Long Questions (Marks: 10)**

A. C# scripting is widely used in Unity for game development. Explain the use of variables in methods and the significance of the [SerializeField] attribute. Compare the usage of [SerializeField] with public fields, highlighting their advantages and disadvantages.

B. Effectors in Unity allow for advanced physics interactions. Discuss the concept of effectors and their usage in Unity game development. Provide examples of scenarios where effectors can be applied to enhance gameplay mechanics.

C. Scriptable Objects are a powerful asset management tool in Unity. Explain the concept of Scriptable Objects and their usage in game development. Discuss the advantages and use cases of Scriptable Objects, providing examples to support your explanation.

**Short Questions (Marks: 10)**

A. Describe the concept of canvas in Unity’s UI system.  
B. What is the purpose of Rigidbody2D in Unity?  
C. Compare and contrast the concepts of screen space and world space in Unity's UI system.  
D. How can sound effects be incorporated into Unity games?  
E. Explain the role of SceneManager in Unity.  
F. Discuss the significance of using the Print to Console function in Unity.

**Q.2 Using different assets (game objects, components, scripts, and animations).**

*(NOTE: Do any 2 from long questions and any 4 from short questions)*  
**(Total Marks: 40)**

**Long Questions (Marks: 20)**

A. Implementing character movement is a fundamental aspect of game development. Explain how to achieve 2D game object movement using the transform.Rotate() and transform.Translate() functions in Unity. Provide a code snippet illustrating the implementation of these functions in a game scenario.

B. Collision detection plays a crucial role in game development. Discuss the implementation of collision detection in Unity, focusing on 2D colliders and triggers. Provide code examples to demonstrate the usage of colliders and triggers in a game scenario.

C. Tile Palette and Tilemap are powerful tools in Unity for designing game levels. Explain the concept of Tile Palette and how it can be used in conjunction with Tilemap to design and compose game levels. Provide step-by-step instructions and code snippets to demonstrate the usage of Tile Palette and Tilemap.

D. Buttons are commonly used for user interaction in games. Explain the concept of a button in Unity's UI system. Discuss the events and functionalities associated with buttons and provide code snippets to demonstrate their implementation.

E. Sprite Shape is a versatile game object in Unity. Explain the concept of Sprite Shape and how it can be utilized to create complex and dynamic 2D environments. Provide step-by-step instructions to demonstrate the use of Sprite Shape.

F. Rule Tile is a versatile feature in Unity for creating tile-based levels. Discuss the concept of Rule Tile and its usage in designing levels. Provide examples to illustrate the implementation of Rule Tile in a game scenario.

**Short Questions (Marks: 20)**

A. How can lists be used in C# for game development?  
B. Describe the process of creating 2D animations using sprite sheets in Unity.  
C. How can Rigidbody2D be used to apply different physics properties, such as torque and force, to game objects in Unity?  
D. Discuss the concept of crash detection in Unity. How can crash detection be implemented to handle collisions between game objects?  
E. Describe the process of creating and using particle effects in Unity.  
F. How can we use Unity input system to get user input?  
G. How can colliders be applied to a Tilemap in Unity?  
H. How can framerate independence be achieved in Unity?

**Long Questions (Marks: 10)**

**A. Use of Variables in Methods and the Significance of [SerializeField] Attribute**

In Unity's C# scripting, variables in methods are essential for storing and manipulating data. They allow developers to define and use data types that represent in-game attributes, like player health or score, and perform various calculations or logic-based actions. Variables in methods can be local (accessible only within the method) or class-level (accessible by other methods in the class).

The [SerializeField] attribute in Unity enables private fields to be serialized, making them visible and editable in the Unity Inspector without exposing them as public. This is particularly useful for keeping data encapsulated while still allowing designers to tweak variables in the Inspector.

* **Advantages of [SerializeField]:**
  + Maintains encapsulation while allowing Inspector customization.
  + Reduces potential misuse by keeping the field private.
* **Advantages of Public Fields:**
  + Accessible from other scripts, which is useful for variables meant to interact with multiple components.
* **Disadvantages of [SerializeField]:**
  + Variables are limited in accessibility to other scripts.
* **Disadvantages of Public Fields:**
  + Exposes the field to other scripts, which could lead to unintended modifications and decreased encapsulation.

**B. Effectors in Unity and Their Usage in Game Development**

Effectors in Unity are components that allow for advanced physics interactions. They enable objects to behave according to specific physical rules, enhancing gameplay realism and immersion. Common Unity effectors include **Area Effector 2D** (applies directional forces), **Point Effector 2D** (attracts or repels objects), and **Surface Effector 2D** (mimics friction or conveyor belts).

**Example Use Cases for Effectors**:

* **Area Effector** for a wind zone that pushes objects within a defined area.
* **Point Effector** for a magnet effect, attracting or repelling nearby items.
* **Platforming Games** where Surface Effectors make conveyors to move objects in specific directions.

**C. Scriptable Objects in Unity**

Scriptable Objects are data containers in Unity that allow developers to store large amounts of information independently of scene instances. They are created using the CreateAssetMenu attribute and can hold data such as character stats, item attributes, or settings.

**Advantages**:

* Reduces memory usage by creating centralized assets used across multiple instances.
* Simplifies data management and promotes reusability.
* Separates data from code, enabling designers to modify assets without affecting scripts directly.

**Example Use Cases**:

* **Inventory System**: Store item properties (name, description, stats) in Scriptable Objects.
* **Character Attributes**: Use Scriptable Objects to manage stats like health, mana, or strength, shared across characters.

**Short Questions (Marks: 10)**

**A. Canvas in Unity’s UI System**

The Canvas is a UI component in Unity that acts as a root container for all UI elements. It organizes and manages UI objects and defines their rendering order and scaling based on screen or world space.

**B. Purpose of Rigidbody2D in Unity**

The Rigidbody2D component enables 2D game objects to interact with Unity's physics system, allowing them to respond to forces like gravity and collisions, making them more dynamic and realistic.

**C. Screen Space vs. World Space in Unity's UI System**

* **Screen Space**: Renders UI elements relative to the camera view, unaffected by the game world's position.
* **World Space**: Renders UI elements within the 3D scene, allowing them to interact with game objects and physics.

**D. Incorporating Sound Effects into Unity Games**

Sound effects in Unity are added using the Audio Source component, which plays audio clips triggered by game events. Sound can enhance gameplay by signaling actions, transitions, or creating ambiance.

**E. Role of SceneManager in Unity**

The SceneManager controls scene management in Unity, allowing developers to load, unload, or switch scenes in the game, essential for transitioning between levels or menus.

**F. Significance of Print to Console Function in Unity**

The Debug.Log() function is used to print messages to the Unity Console, helping developers track variable values, debug logic, and verify code behavior during development.

**Long Answer Questions (20 Marks Each)**

**A. Implementing 2D Game Object Movement in Unity**

In Unity, the transform.Translate() function is used to move game objects in a specific direction, and transform.Rotate() is used to rotate game objects. Here's how to implement movement using these functions:

1. **transform.Translate()** moves the game object along specified axes, making it ideal for controlling movement in 2D games.
2. **transform.Rotate()** rotates the game object around an axis, commonly used for changing direction.

**Example Code:**

using UnityEngine;

public class PlayerMovement : MonoBehaviour

{

public float moveSpeed = 5f;

public float rotateSpeed = 100f;

void Update()

{

// Move forward based on vertical input (W/S or Up/Down keys)

float move = Input.GetAxis("Vertical") \* moveSpeed \* Time.deltaTime;

transform.Translate(0, move, 0);

// Rotate based on horizontal input (A/D or Left/Right keys)

float rotation = Input.GetAxis("Horizontal") \* rotateSpeed \* Time.deltaTime;

transform.Rotate(0, 0, -rotation);

}

}

This script uses Unity's input system to move a game object forward and backward and rotate it based on player input.

**B. Collision Detection in Unity Using 2D Colliders and Triggers**

Unity uses 2D colliders (like BoxCollider2D, CircleCollider2D, and PolygonCollider2D) for detecting collisions. The **OnCollisionEnter2D** and **OnTriggerEnter2D** methods handle interactions between colliders.

**Example Code:**

csharp

Copy code

using UnityEngine;

public class CollisionDetection : MonoBehaviour

{

void OnCollisionEnter2D(Collision2D collision)

{

Debug.Log("Collision with " + collision.gameObject.name);

}

void OnTriggerEnter2D(Collider2D other)

{

Debug.Log("Trigger with " + other.gameObject.name);

}

}

In this example, **OnCollisionEnter2D** detects when a rigidbody collides with another collider, and **OnTriggerEnter2D** is triggered when an object with a Trigger collider passes through another object.

**C. Tile Palette and Tilemap in Unity**

The **Tile Palette** is a Unity tool that allows developers to paint tiles on a grid, while the **Tilemap** component serves as the canvas.

**Step-by-Step Instructions:**

1. **Create a Tile Palette**: Go to *Window > 2D > Tile Palette* and create a new palette.
2. **Add Tiles**: Drag images or sprites into the palette to add them as tiles.
3. **Create a Tilemap**: Create an empty GameObject, then add a *Grid* component and a *Tilemap* component.
4. **Paint the Tilemap**: Select the Tile Palette window, choose a tile, and paint on the Tilemap.

**Example Code:**

csharp

Copy code

// Typically, no additional code is required for basic Tilemap usage,

// but you can modify tiles in code by accessing the Tilemap component.

using UnityEngine;

using UnityEngine.Tilemaps;

public class TilemapExample : MonoBehaviour

{

public Tilemap tilemap;

public TileBase newTile;

void Start()

{

// Set a tile at a specific position

tilemap.SetTile(new Vector3Int(0, 0, 0), newTile);

}

}

**D. Buttons in Unity’s UI System**

Buttons in Unity’s UI trigger events and are typically connected to functions within scripts.

**Example Code:**

public class ButtonExample : MonoBehaviour

{

public Button myButton;

void Start()

{

myButton.onClick.AddListener(OnButtonClick);

}

void OnButtonClick()

{

Debug.Log("Button clicked!");

}

}

This example shows how to create a button click event by attaching a function.

**E. Using Sprite Shape in Unity**

Sprite Shape enables creating custom 2D paths. Here’s how to use it:

1. **Add a Sprite Shape Renderer**: *GameObject > 2D Object > Sprite Shape*.
2. **Edit the Sprite Shape**: Use the Sprite Shape Profile to define paths and assign different sprites.
3. **Adjust Points on the Path**: Click and drag points to form shapes.

**No code is required** for basic Sprite Shape usage, but you can programmatically change the shape path if needed.

**F. Rule Tile in Unity**

Rule Tile allows for pattern-based tile placement.

**Example Code:**

1. Add a Rule Tile and configure its rules (e.g., placing specific tiles based on adjacent tiles).
2. Use the Tile Palette to paint tiles that auto-adjust based on set rules.

**No specific code** is needed unless manipulated Rule Tiles in code.

**Short Answer Questions (5 Marks Each)**

**A. Lists in Game Development:** Lists in C# are dynamic collections that can store various game objects, like inventory items or enemies.

**B. 2D Animation with Sprite Sheets:** In Unity, import a sprite sheet, slice it in the Sprite Editor, and use it in an animation clip for frame-by-frame animation.

**C. Rigidbody2D for Physics:** Rigidbody2D provides physical properties. Use AddForce() to apply linear force and AddTorque() for rotational force.

**D. Crash Detection in Unity:** Use colliders and collision events (OnCollisionEnter2D) to detect crashes and respond with in-game logic.

**E. Particle Effects:** Create a Particle System, customize settings, and attach it to game objects to enhance visuals.

**F. Unity Input System:** The input system captures user actions via Input.GetKey(), Input.GetAxis(), etc., to control character movement or interact with UI.

**G. Tilemap Colliders:** Attach a TilemapCollider2D to a Tilemap for collision functionality.

**H. Framerate Independence:** Use Time.deltaTime to adjust movement for consistent gameplay across devices with varying frame rates.