Brett Stelly

July 10, 2018, 2018

Cross-platform Development

1. **Requirements Documentation**
   1. **Description of problem**

**Name:** Backpack Game

**Problem Statement:** Build and deploy an application to multiple platforms

**Problem Specifications**: The project must support multiple platforms and their specific input devices, demonstrate inheritance, implement a GUI, use scriptable objects, use json format to persist data, and use an editor tool that builds objects separate from behaviours.

* 1. **Input information**

The user uses the keyboard to move the character and interact with the inventory if on PC. If using an android device, the user uses the touchscreen to move the character and interact with the inventory.

* 1. **Output Information**

The application displays text to the screen to display the users score and images in a panel at the top of the screen to show inventory slots.

* 1. **User Interface**

The user interface contains a textbox and a panel with 3 images.

1. **System Architecture**

**CameraBehaviour.cs**

Prototype: Start

Arguments: None

Description: Gets the player game object from the scene and sets it to a variable

Precondition: None

Postcondition: The player game object is set to a variable

Protection Level: private

Prototype: Update

Arguments: None

Description: Every frame, the camera will follow the player

Precondition: There must be a gameobject in the scene called "Player"

Postcondition: Camera positon changed

Protection Level: private

**ItemBehaviour.cs**

Prototype: Start

Arguments: None

Description: Gets the sprite renderer component and the sprite

Precondition: None

Postcondition: The sprite renderer component and the sprite contained in the item

are stored in variables

Protection Level: private

Prototype: OnTriggerEnter2D

Arguments: Collider2D

Description: Adds Item to the players inventory list if player and Item collide

Precondition: Player and Item must collide

Postcondition: The item is added to the player's list of inventory

Protection Level: private

**PanelBehaviour.cs**

Prototype: Start

Arguments: None

Description: Creates a new list of images and gets the player object’s

PlayerBehaviour component

Precondition: None

Postcondition: The panel has a new list of images and has access to the player

object's PlayerBehaviour component

Protection Level: private

Prototype: Update

Arguments: None

Description: Refreshes the image boxes in the panel and displays all currently picked

up items

Precondition: None

Postcondition: The image boxes in the panel are refreshed

Protection Level: private

**ScoreManagerBehaviour.cs**

Prototype: Start

Arguments: None

Description: Retrieves the player object's PlayerBehaviour component

Precondition: None

Postcondition: Score manager has access to a player object PlayerBehaviour component

Protection Level: private

Prototype: Update

Arguments: None

Description: Updates the text in the text box with the player's score every frame

Precondition: None

Postcondition: The textbox is updated

Protection Level: private

**PlayerBehaviour.cs**

Prototype: Start

Arguments: None

Description: retrieves the sprite renderer component

Precondition: None

Postcondition: PlayerBehaviour can now access the player's sprite renderer component

Protection Level: private

Prototype: Update

Arguments: None

Description: Gets input and checks player's position. If player falls below -100,

set player's position back to starting point

Precondition: None

Postcondition: Player's positon is updated

Protection Level: private

Prototype: GetInput

Arguments: None

Description: Checks for input from keyboard or touch controls

Precondition: None

Postcondition: Moves player or removes items from inventory based on input

Protection Level: private

Prototype: OnCollisionEnter2D

Arguments: Collision2D

Description: sets a player's CanJump value to true if player collides with a world

object, or adds to player's score if player collides with a coin

Precondition: Player must collide with either an object in the list of "World" or

list of "Coins"

Postcondition: Player's CanJump value is set to true or player's score is updated

Protection Level: private

**Inventory.cs**

Prototype: Add

Arguments: Item

Description: Adds Item object to list of items

Precondition: An inventory scriptable object must exist

Postcondition: An Item object is added to items list

Protection Level: public

Prototype: Remove

Arguments: Item

Description: Removes an Item object from list of items

Precondition: An inventory scriptable object must exist

Postcondition: An Item is removed from items list

Protection Level: public

Prototype: Serialize

Arguments: None

Description: Serializes an Inventory scriptable object to a file

Precondition: An inventory scriptable object must exist

Postcondition: An inventory scriptable object is serialized

Protection Level: public

Prototype: Deserialize

Arguments: string

Description: Deserializes an inventory scriptable object

Precondition: An inventory scriptable object must exist

Postcondition: An inventory object is overwritten by the deserialized inventory

Protection Level: public

1. **Source** **Code**

**CameraBehaviour.cs**

public class CameraBehaviour : MonoBehaviour

{

private GameObject player;

private float z = -90;

void Start ()

{

player = GameObject.Find("Player");

}

void Update ()

{

transform.position = new Vector3(player.transform.position.x, player.transform.position.y + 10, z);

}

}

**ItemBehaviour.cs**

public class ItemBehaviour : MonoBehaviour

{

public Item item;

public SpriteRenderer sr;

private bool isColliding = false;

void Start()

{

sr = GetComponent<SpriteRenderer>();

sr.sprite = item.image;

}

void OnTriggerEnter2D(Collider2D other)

{

if (isColliding)

{

return;

}

isColliding = true;

Debug.Log("collided");

if (other.CompareTag("Player"))

{

var player = other.GetComponent<PlayerBehaviour>();

player.inventory.Add(item);

Destroy(gameObject);

}

}

}

**PanelBehaviour.cs**

public class PanelBehaviour : MonoBehaviour {

public PlayerBehaviour player;

public Image imageBox1;

public Image imageBox2;

public Image imageBox3;

private List<Image> images;

void Start()

{

images = new List<Image>{imageBox1, imageBox2, imageBox3};

player = GameObject.Find("Player").GetComponent<PlayerBehaviour>();

}

void Update () {

var count = player.inventory.items.Count;

for (int i = 0; i < 3; i++)

{

images[i].sprite = null;

}

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

{

images[i].sprite = player.inventory.items[i].image;

}

}

}

**ScoreManagerBehaviour.cs**

public class ScoreManagerBehaviour : MonoBehaviour

{

private int score;

public Text scoreText;

private PlayerBehaviour player;

void Start () {

player = GameObject.Find("Player").GetComponent<PlayerBehaviour>();

}

void Update () {

scoreText.text = "Score: " + player.score;

}

}

**Item.cs**

[CreateAssetMenu(menuName = "Item")]

public class Item : ScriptableObject

{

public Sprite image;

}

**PlayerBehaviour.cs**

public Inventory inventory;

public static bool CanJump { get; set; }

public static bool DiscardKeyDown { get; set; }

public float jumpModifier = 1;

public int score;

private SpriteRenderer renderer;

void Start ()

{

renderer = GetComponent<SpriteRenderer>();

}

void Update ()

{

GetInput();

if (transform.position.y <= -100)

{

transform.position = new Vector3(-10, -19.77f, 0);

score -= 100;

Debug.Log("Game Over");

}

}

void GetInput()

{

if (Input.GetKey(KeyCode.A))

{

transform.position += Vector3.left;

renderer.flipX = true;

}

if (Input.GetKey(KeyCode.D))

{

transform.position += Vector3.right;

renderer.flipX = false;

}

if (Input.GetButton("Jump"))

{

if (CanJump)

{

transform.position += Vector3.up \* jumpModifier;

}

}

if (Input.GetButtonUp("Jump"))

{

CanJump = false;

}

if (Input.GetKeyDown(KeyCode.Alpha1) && !DiscardKeyDown)

{

DiscardKeyDown = true;

if(inventory.items.Count >= 1)

inventory.Remove(inventory.items[0]);

}

if (Input.GetKey(KeyCode.Alpha2) && !DiscardKeyDown)

{

DiscardKeyDown = true;

if (inventory.items.Count >= 2)

inventory.Remove(inventory.items[1]);

}

if (Input.GetKey(KeyCode.Alpha3) && !DiscardKeyDown)

{

DiscardKeyDown = true;

if (inventory.items.Count >= 3)

inventory.Remove(inventory.items[2]);

}

if (Input.GetKeyUp(KeyCode.Alpha1) || Input.GetKeyUp(KeyCode.Alpha2) || Input.GetKeyUp(KeyCode.Alpha3))

{

DiscardKeyDown = false;

}

if (Input.GetKeyDown(KeyCode.Return) || Input.GetKeyDown(KeyCode.KeypadEnter))

{

inventory.Serialize();

}

if (Input.GetKeyDown(KeyCode.RightShift))

{

string directory = Application.persistentDataPath + @"\save.json";

inventory.Deserialize(directory);

}

}

void OnCollisionEnter2D(Collision2D col)

{

GameObject world = GameObject.Find("World");

GameObject coins = GameObject.Find("Coins");

for (int i = 0; i < world.transform.childCount; i++)

{

if (col.gameObject.name == world.transform.GetChild(i).name)

{

CanJump = true;

}

}

for (int i = 0; i < coins.transform.childCount; i++)

{

if (col.gameObject.name == coins.transform.GetChild(i).name)

{

Destroy(coins.transform.GetChild(i).gameObject);

score += 100;

}

}

**Inventory.cs**

[CreateAssetMenu(menuName = "Inventory")]

[Serializable]

public class Inventory : ScriptableObject

{

[SerializeField] public List<Item> items;

[TextArea]

[SerializeField]

//string json;

InventoryTemp temp = new InventoryTemp();

public void Add(Item item)

{

items.Add(item);

}

public void Remove(Item item)

{

items.Remove(item);

}

public void Serialize()

{

string json = JsonUtility.ToJson(this, true);

string directory = Application.persistentDataPath;

File.WriteAllText(directory + @"\save.json", json);

Debug.Log(directory);

}

public void Deserialize(string json)

{

var newInventory = CreateInstance<Inventory>();

JsonUtility.FromJsonOverwrite(json, newInventory);

items = newInventory.items;

}

}