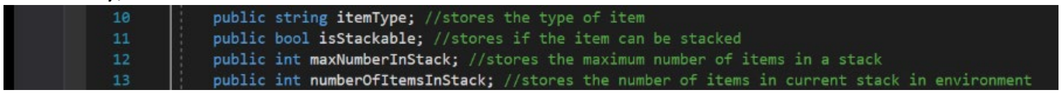
CHARACTER_INVENTORY_SYSTEM_DOCUMENTATION

Author: Aleksandra Zatorska
Version: 1.01f
Last Updated: 05/11/2020

Item_Class

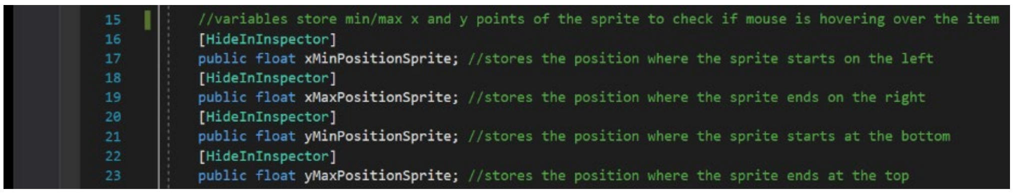
The 'Item' class defines the properties of each item: what type of object it is, whether it is stackable, the maximum number that a stack can contain and the number of items in the current stack in the environment. It also contains the min/max x and ypositions of the sprite (as well as the sprite itself to find the positions) for that item object as float values, which are used by the main function to detect whether the mouse is hovering and being clicked on the item. If the item is stackable the class will also store the text component in the child object to display the number of items in the stack. Each item object has its own 'Item' script attached to it, to store the values for that specific item.

These variables store values that define the type of item, whether it is stackable, and the number of items in the stack as well as the maximum stack number for that item type. These are used in the 'MainScript' class to check the properties of an item when it is moved to/from the inventory/environment.



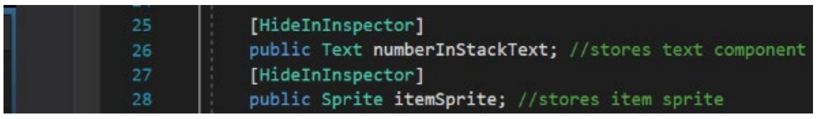
```
public string itemType; //stores the type of item
public bool isStackable; //stores if the item can be stacked
public int maxNumberInStack; //stores the maximum number of items in a stack
public int numberOfItemsInStack; //stores the number of items in current stack in environment
```

These variables store the float values of the min/max x and y positions of the sprite for that item. They are used by the 'MainScript' class to check whether the mouse is hovering over an item when the LMB/RMB are pressed.



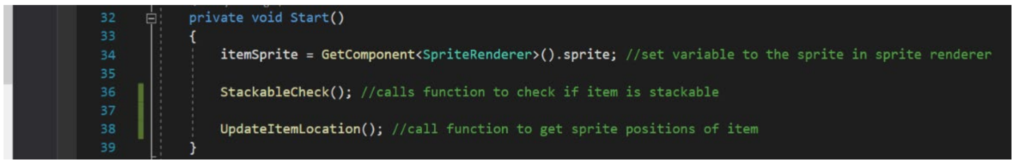
```
//variables store min/max x and y points of the sprite to check if mouse is hovering over the item
[HideInInspector]
public float xMinPositionSprite; //stores the position where the sprite starts on the left
[HideInInspector]
public float xMaxPositionSprite; //stores the position where the sprite ends on the right
[HideInInspector]
public float yMinPositionSprite; //stores the position where the sprite starts at the bottom
[HideInInspector]
public float yMaxPositionSprite; //stores the position where the sprite ends at the top
```

These variables store the components of the item objects/child objects. The 'numberInStackText' stores the text component of the item's text child object, which is used to display the number of items in the current stack if the object is stackable. The 'itemSprite' variable stores the sprite component of the item object, which is used to calculate the four boundary corner points for detecting mouse clicks hovering over the item, as well as storing the sprite for that item to add the sprite to an inventory slot when the item is added to inventory.



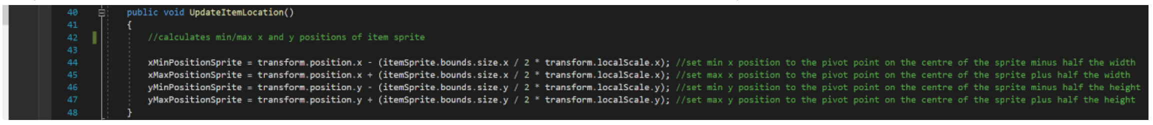
```
[HideInInspector]
public Text numberInStackText; //stores text component
[HideInInspector]
public Sprite itemSprite; //stores item sprite
```

The 'Start()' function is called when the game is loaded. In this function the 'StackableCheck()' function is called, and the 'itemSprite' is set to the sprite stored in the sprite renderer of the item object.



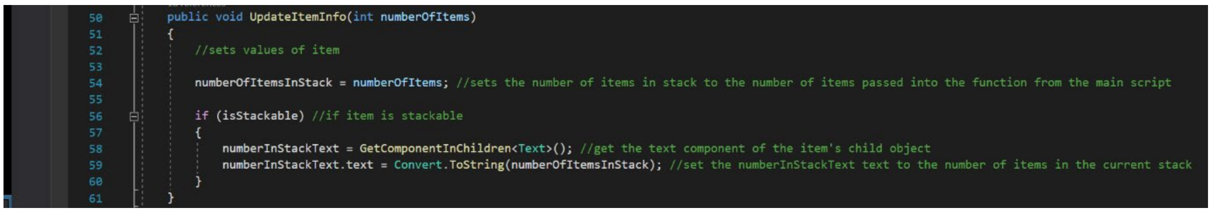
```
private void Start()
{
    itemSprite = GetComponent<SpriteRenderer>().sprite; //set variable to the sprite in sprite renderer

    StackableCheck(); //calls function to check if item is stackable

    UpdateItemLocation(); //call function to get sprite positions of item
}
```

This public function calculates the min/max x and y values of the sprite by getting the size of the sprite on the x and y axis (divided by 2 since the transform.position returns the pivot point at the centre of the sprite, multiplied by the scale on the x/y axis of the object since not all objects have a scale of 1 so the size of the sprite will change with the scale) and add/minus this to the current transform.position of the item object. It is called in the ‘Start()’ function of the ‘Item’ class to get the positions for each item when the game is loaded, and it is called from the ‘MainScript’ class when an object is instantiated into the environment from the inventory.



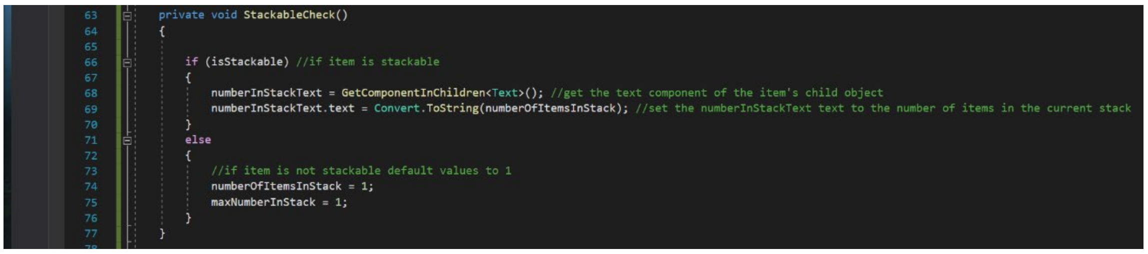
```
public void UpdateItemLocation()
{
    //calculates min/max x and y positions of item sprite

    xMinPositionSprite = transform.position.x - (itemSprite.bounds.size.x / 2 * transform.localScale.x); //set min x position to the pivot point on the centre of the sprite minus half the width
    xMaxPositionSprite = transform.position.x + (itemSprite.bounds.size.x / 2 * transform.localScale.x); //set max x position to the pivot point on the centre of the sprite plus half the width
    yMinPositionSprite = transform.position.y - (itemSprite.bounds.size.y / 2 * transform.localScale.y); //set min y position to the pivot point on the centre of the sprite minus half the height
    yMaxPositionSprite = transform.position.y + (itemSprite.bounds.size.y / 2 * transform.localScale.y); //set max y position to the pivot point on the centre of the sprite plus half the height
}
```

This public function updates the number of items in the current stack in the environment and the text object to that number if the item is stackable. It is called by the ‘MainScript’ class when an item is instantiated into the environment from the inventory, where an integer argument is passed to set the number of items in the stack.



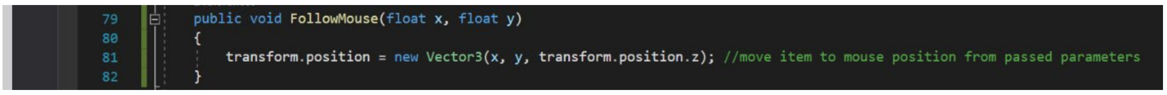
```
public void UpdateItemInfo(int numberOfItems)
{
    //sets values of item

    numberOfItemsInStack = numberOfItems; //sets the number of items in stack to the number of items passed into the function from the main script

    if (isStackable) //if item is stackable
    {
        numberInStackText = GetComponentInChildren<Text>(); //get the text component of the item's child object
        numberInStackText.text = Convert.ToString(numberOfItemsInStack); //set the numberInStackText text to the number of items in the current stack
    }
}
```

This private function is called by the ‘Start()’ function in the ‘Item’ class when the game is loaded to check if an item is stackable, if yes it gets the text component of the item’s child object and sets it to the number of items in the stack, if not it defaults the values to 1.



```
private void StackableCheck()
{

    if (isStackable) //if item is stackable
    {
        numberInStackText = GetComponentInChildren<Text>(); //get the text component of the item's child object
        numberInStackText.text = Convert.ToString(numberOfItemsInStack); //set the numberInStackText text to the number of items in the current stack
    }
    else
    {
        //if item is not stackable default values to 1
        numberOfItemsInStack = 1;
        maxNumberInStack = 1;
    }
}
```

This public function is called from the ‘MainScript’ class, which passes two float arguments (the x and y positions of the mouse on screen), when the LMB on the mouse is pressed over the item. It sets the item’s transform.position to the arguments passed so the item follows the mouse position while the LMB is being pressed from the ‘MainScript’ class.

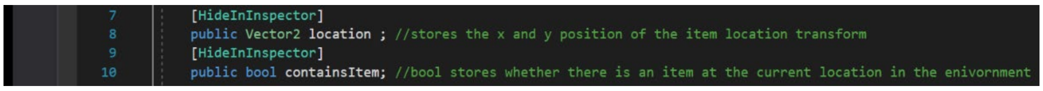


```
public void FollowMouse(float x, float y)
{
    transform.position = new Vector3(x, y, transform.position.z); //move item to mouse position from passed parameters
}
```

ItemLocation_Class

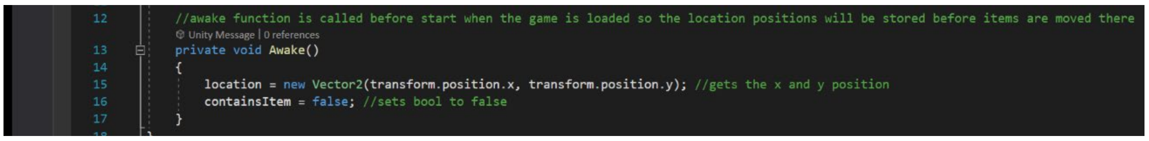
The ‘ItemLocation’ class stores the locations in the environment where the items should be placed and appear on screen to keep them organised and not overlapping each other. Each empty item location object contains the ‘ItemLocation’ script to store its position and whether an item is in that location. Each item location is an empty game object that is used to set the positions for each item object in the environment.

The Vector2 variable stores the x and y position of the empty game object so it can be used to set the position for items in the environment. The z position is not stored and used since everything is in 2D so the z values do not need to be changed. The bool variable is used to check whether an item is placed in that location.



```
[HideInInspector]
public Vector2 location ; //stores the x and y position of the item location transform
[HideInInspector]
public bool containsItem; //bool stores whether there is an item at the current location in the enivornment
```

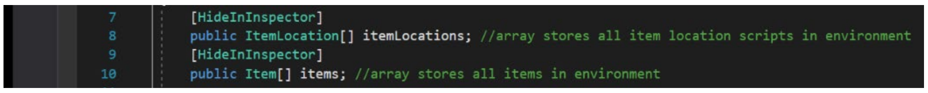
The function sets the ‘location’ variable to the x and y positions of the empty game object and the ‘containsItem’ bool is set to false as default for all item locations. This is done in the ‘Awake()’ function so that the location values can be stored before any item positions are set in the ‘Start()’ functions.



```
//awake function is called before start when the game is loaded so the location positions will be stored before items are moved there
Unity Message | 0 references
private void Awake()
{
    location = new Vector2(transform.position.x, transform.position.y); //gets the x and y position
    containsItem = false; //sets bool to false
}
```

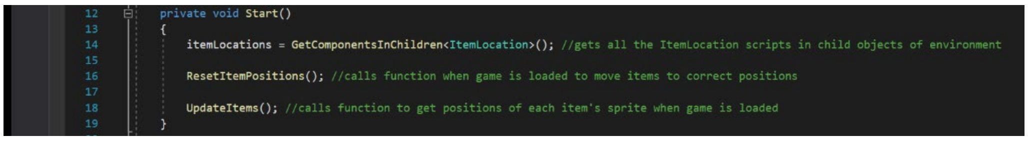
Environment_Class
The ‘Environment’ class stores arrays of ‘Item’ class objects and ‘ItemLocation’ class objects to set the positions of items to specific locations in the environment to keep everything organised.

The ‘itemLocations’ array stores all the ‘ItemLocation’ script components of the item location child objects of the environment. The ‘items’ array stores all the ‘Item’ script components of the item child objects of the environment.



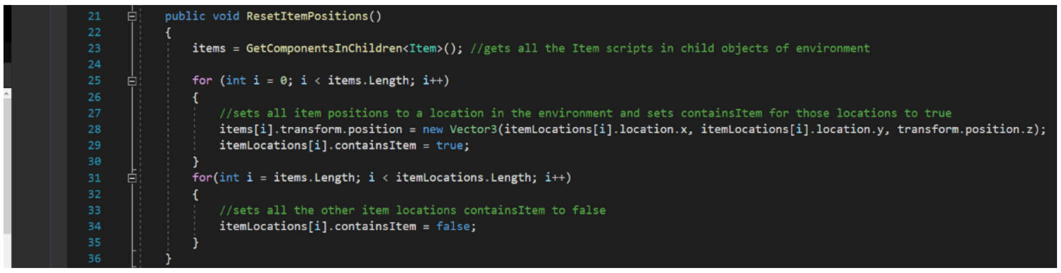
```csharp
    [HideInInspector]
    public ItemLocation[] itemLocations; //array stores all item location scripts in environment
    [HideInInspector]
    public Item[] items; //array stores all items in environment
```

The ‘Start()’ function sets the ‘itemLocations’ array to the script components of the item location child objects of the environment. It also calls the ‘ResetItemPositions()’ and ‘UpdateItems()’ functions.



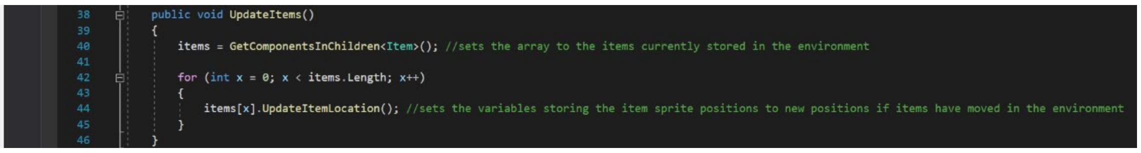
```csharp
    private void Start()
    {
        itemLocations = GetComponentsInChildren<ItemLocation>(); //gets all the ItemLocation scripts in child objects of environment

        ResetItemPositions(); //calls function when game is loaded to move items to correct positions

        UpdateItems(); //calls function to get positions of each item's sprite when game is loaded
    }
```

This public function sets the ‘items’ array to the script components of the item child objects of the environment, and sets the position of each item to an item location in the environment. If an item position is set to an item location the ‘containsItem’ bool for that ‘itemLocation’ object is set to true, and the rest of the ‘itemLocation’ objects have the bool set to false. This is to check whether an item’s position can be set to each location in the environment.



```csharp
    public void ResetItemPositions()
    {
        items = GetComponentsInChildren<Item>(); //gets all the Item scripts in child objects of environment

        for (int i = 0; i < items.Length; i++)
        {
            //sets all item positions to a location in the environment and sets containsItem for those locations to true
            items[i].transform.position = new Vector3(itemLocations[i].location.x, itemLocations[i].location.y, transform.position.z);
            itemLocations[i].containsItem = true;
        }
        for(int i = items.Length; i < itemLocations.Length; i++)
        {
            //sets all the other item locations containsItem to false
            itemLocations[i].containsItem = false;
        }
    }
```

This public function gets the ‘Item’ scripts from the child item objects in the environment when it is called to check if any items have been moved to/from the inventory. It then calls the ‘UpdateItemLocation()’ function from the ‘Item’ script for each item in the array to set the sprite positions if the items have moved in the environment.

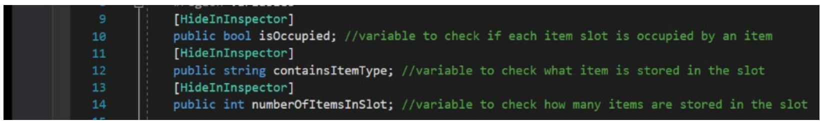


```csharp
    public void UpdateItems()
    {
        items = GetComponentsInChildren<Item>(); //sets the array to the items currently stored in the environment

        for (int x = 0; x < items.Length; x++)
        {
            items[x].UpdateItemLocation(); //sets the variables storing the item sprite positions to new positions if items have moved in the environment
        }
    }
```

InventorySlots_Class

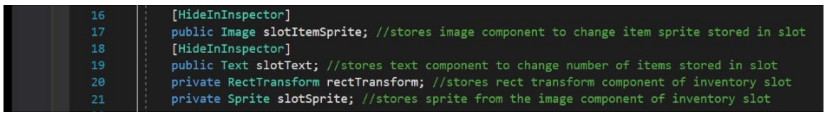
The ‘InventorySlots’ class defines the properties of an inventory slot: whether it contains an item, what item type it contains and how many items it contains. It also contains the min/max x and y positions of the sprite of the slot, as well as the image component of the child object to display the sprite of the item being stored there. Each slot object also stores the text component of its child object to show how many items are stored in that slot.

These variables store whether the slot contains an item, what type of item it contains and the number of items it contains.



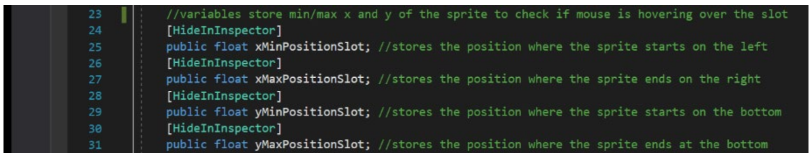
```
[HideInInspector]
public bool isOccupied; //variable to check if each item slot is occupied by an item
[HideInInspector]
public string containsItemType; //variable to check what item is stored in the slot
[HideInInspector]
public int numberOfItemsInSlot; //variable to check how many items are stored in the slot
```

These variables store the components of the slot and its child objects. It stores the rectTransform and sprite of the slot, which are used to calculate the min/max x and y positions of the slot sprite to detect whether the mouse is hovering over the slot. The image and text components of the child objects are also stored to display the type of item and number of items being stored in that slot.



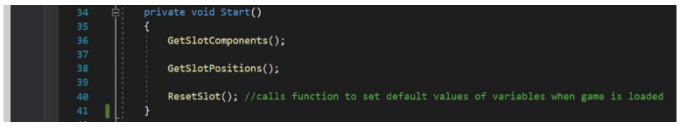
```
[HideInInspector]
public Image slotItemSprite; //stores image component to change item sprite stored in slot
[HideInInspector]
public Text slotText; //stores text component to change number of items stored in slot
private RectTransform rectTransform; //stores rect transform component of inventory slot
private Sprite slotSprite; //stores sprite from the image component of inventory slot
```

These variables store the float values of the positions of the four corners of the slot sprite.



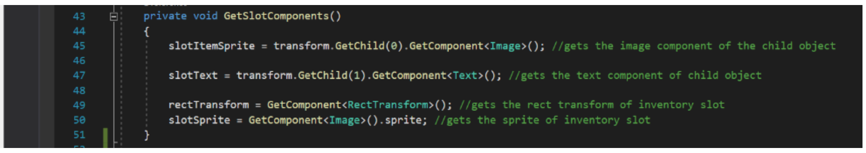
```
//variables store min/max x and y of the sprite to check if mouse is hovering over the slot
[HideInInspector]
public float xMinPositionSlot; //stores the position where the sprite starts on the left
[HideInInspector]
public float xMaxPositionSlot; //stores the position where the sprite ends on the right
[HideInInspector]
public float yMinPositionSlot; //stores the position where the sprite starts on the bottom
[HideInInspector]
public float yMaxPositionSlot; //stores the position where the sprite ends at the bottom
```

The ‘Start()’ function calls three functions: ‘GetSlotComponents()’, ‘GetSlotPositions()’ and ‘ResetSlot()’.



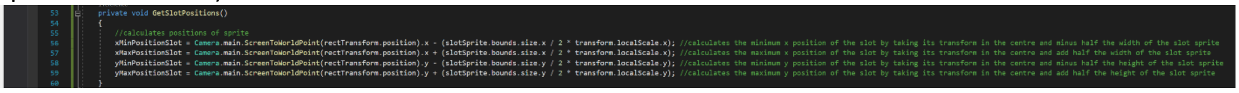
```
private void Start()
{
    GetSlotComponents();

    GetSlotPositions();

    ResetSlot(); //calls function to set default values of variables when game is loaded
}
```

This function is called by the ‘Start()’ function when the game loads to store the sprite, image and text components of the slot and its child objects.



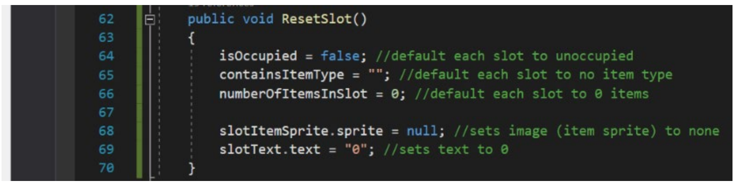
```
private void GetSlotComponents()
{
    slotItemSprite = transform.GetChild(0).GetComponent<Image>(); //gets the image component of the child object

    slotText = transform.GetChild(1).GetComponent<Text>(); //gets the text component of child object

    rectTransform = GetComponent<RectTransform>(); //gets the rect transform of inventory slot
    slotSprite = GetComponent<Image>().sprite; //gets the sprite of inventory slot
}
```

This function is called by the ‘Start()’ function to calculate the min/max x and y positions of the slot sprite. The ‘slotSprite’ variable is used to get the width/height of the slot sprite (divided by 2 since the ‘rectTransform’ returns the pivot point at the centre of the sprite, multiplied by the scale on the x/y axis of the object) plus the position of the ‘rectTransform’ in world space coordinates (the ‘ScreenToWorldPosition()’ function converts the ‘rectTransform’ position on screen/canvas to world space coordinates).



```
private void GetSlotPositions()
{
    //calculates positions of sprite
    xMinPositionSlot = Camera.main.ScreenToWorldPoint(rectTransform.position).x - (slotSprite.bounds.size.x / 2 * transform.localScale.x); //calculates the minimum x position of the slot by taking its transform in the centre and minus half the width of the slot sprite
    xMaxPositionSlot = Camera.main.ScreenToWorldPoint(rectTransform.position).x + (slotSprite.bounds.size.x / 2 * transform.localScale.x); //calculates the maximum x position of the slot by taking its transform in the centre and add half the width of the slot sprite
    yMinPositionSlot = Camera.main.ScreenToWorldPoint(rectTransform.position).y - (slotSprite.bounds.size.y / 2 * transform.localScale.y); //calculates the minimum y position of the slot by taking its transform in the centre and minus half the height of the slot sprite
    yMaxPositionSlot = Camera.main.ScreenToWorldPoint(rectTransform.position).y + (slotSprite.bounds.size.y / 2 * transform.localScale.y); //calculates the maximum y position of the slot by taking its transform in the centre and add half the height of the slot sprite
}
```

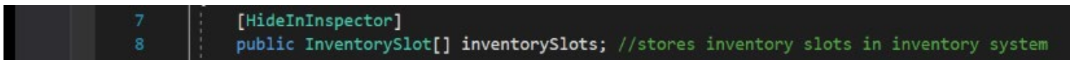
This public function defaults all the slot variables that store values about the item in the inventory slot. It is called by the ‘Start()’ function to default all slots to empty at the start of the game, and from the ‘MainScript’ class when an item is removed from that inventory slot.



```
public void ResetSlot()
{
    isOccupied = false; //default each slot to unoccupied
    containsItemType = ""; //default each slot to no item type
    numberOfItemsInSlot = 0; //default each slot to 0 items

    slotItemSprite.sprite = null; //sets image (item sprite) to none
    slotText.text = "0"; //sets text to 0
}
```

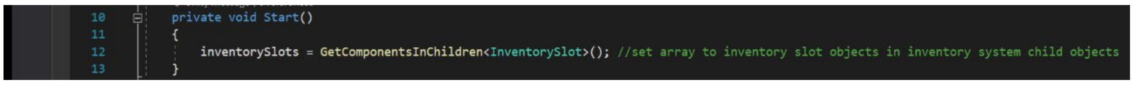
InventorySystem_Class
The ‘InventorySystem’ class is used to store all the ‘inventorySlot’ components of the slot child objects of the inventory system.

The array stores all the ‘InventorySlot’ components of the child objects.



```
[HideInInspector]
public InventorySlot[] inventorySlots; //stores inventory slots in inventory system
```

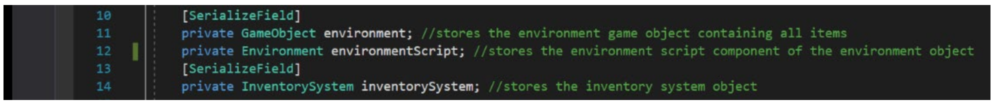
The ‘Start()’ function gets all the ‘InventorySlot’ components of the child objects of the inventory system.



```
private void Start()
{
    inventorySlots = GetComponentsInChildren<InventorySlot>(); //set array to inventory slot objects in inventory system child objects
}
```

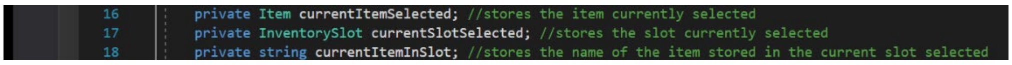
MainScript_Class
The ‘MainScript’ class contains the functions for detecting mouse clicks to add/remove items to/from the inventory.

The ‘environment’ variable stores the environment game object in the scene (this is set in the inspector). The ‘environmentScript’ variable stores the ‘Environment’ component of the environment object. The ‘inventorySystem’ variable stores the ‘InventorySystem’ component of the inventory system object (set in the inspector).



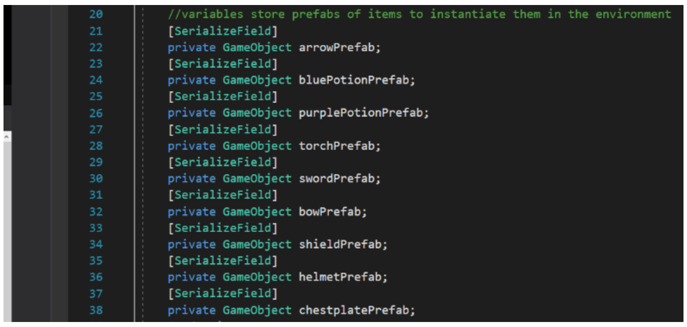
```
[SerializeField]
private GameObject environment; //stores the environment game object containing all items
private Environment environmentScript; //stores the environment script component of the environment object
[SerializeField]
private InventorySystem inventorySystem; //stores the inventory system object
```

The ‘currentItemSelected’ stores the item that was right clicked by the mouse or that is being dragged by left clicking the mouse. The ‘currentSlotSelected’ stores the slot that was right clicked by the mouse or left clicked and dragged by the mouse. The ‘currentItemInSlot’ variable stores the item in the slot currently selected by right clicking the mouse or left clicking and dragging.



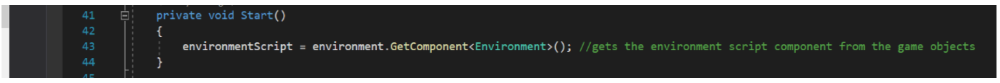
```
private Item currentItemSelected; //stores the item currently selected
private InventorySlot currentSlotSelected; //stores the slot currently selected
private string currentItemInSlot; //stores the name of the item stored in the current slot selected
```

These variable store prefabs of items in the environment so they can be instantiated into the environment when an item is removed from the inventory.



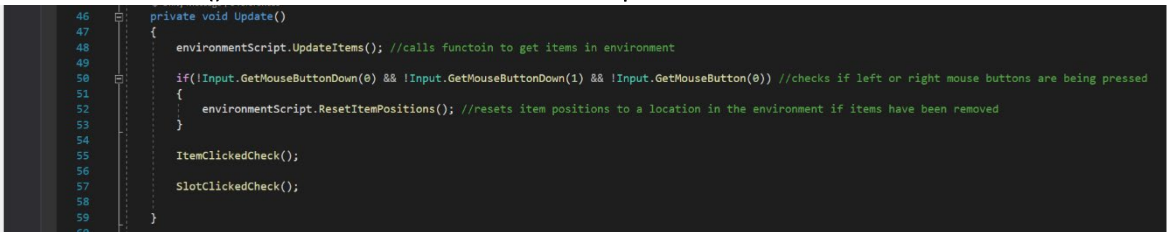
```
//variables store prefabs of items to instantiate them in the environment
[SerializeField]
private GameObject arrowPrefab;
[SerializeField]
private GameObject bluePotionPrefab;
[SerializeField]
private GameObject purplePotionPrefab;
[SerializeField]
private GameObject torchPrefab;
[SerializeField]
private GameObject swordPrefab;
[SerializeField]
private GameObject bowPrefab;
[SerializeField]
private GameObject shieldPrefab;
[SerializeField]
private GameObject helmetPrefab;
[SerializeField]
private GameObject chestplatePrefab;
```

The ‘Start()’ function gets the ‘Environment’ script component from the environment object.



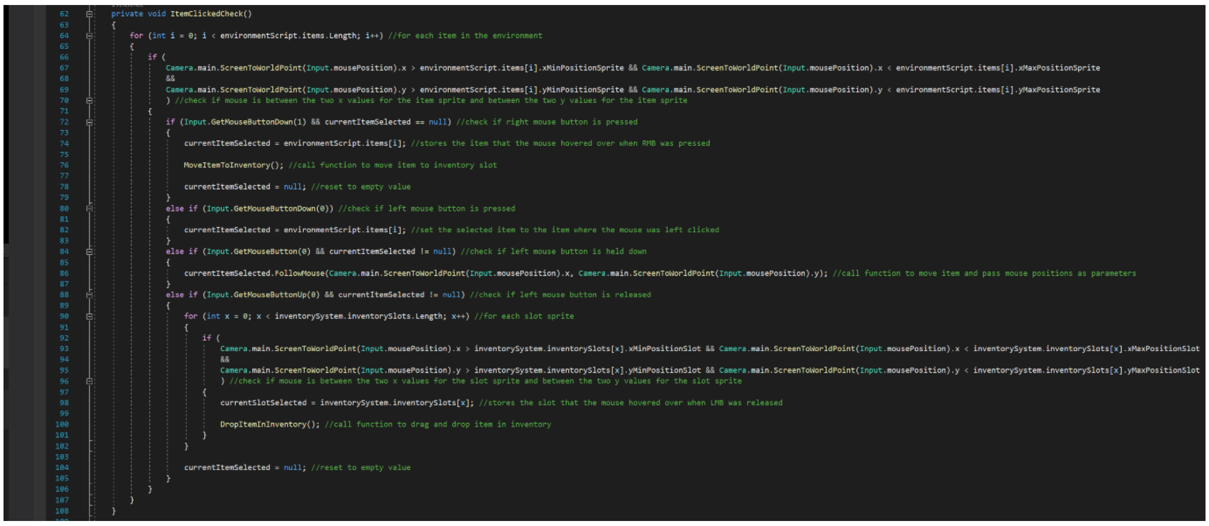
```
private void Start()
{
    environmentScript = environment.GetComponent<Environment>(); //gets the environment script component from the game objects
}
```

The ‘Update()’ function calls the ‘UpdateItems()’ function from the ‘Environment’ class to check if items have been added/removed to/from the environment, so their sprite positions can be updated. If no mouse buttons are being clicked or held down it calls the ‘ResetItemPositions()’ function from the ‘Environment’ class to move item to the first empty locations in the environment. ‘Update()’ calls two functions: ‘ItemClickedCheck()’ to check if the RMB or LMB was pressed on an item, and ‘SlotClickedCheck()’ to check if the RMB or LMB was pressed on a slot.



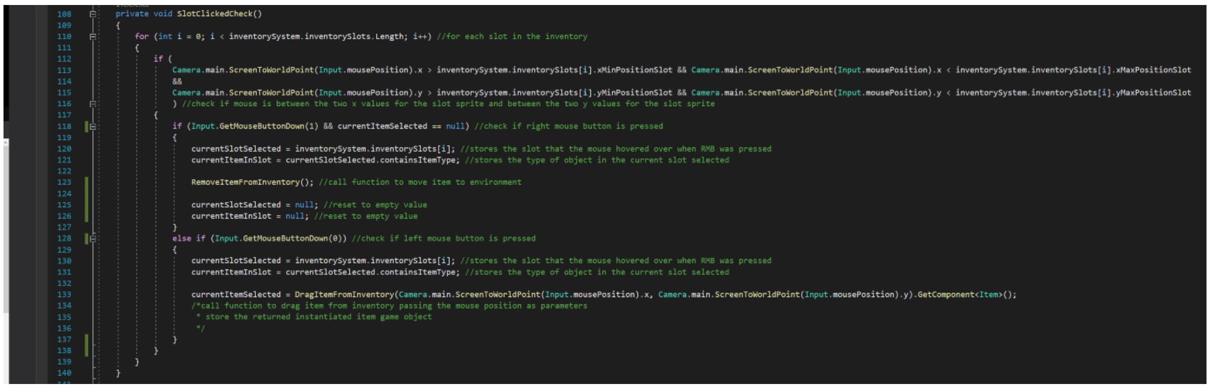
```
private void Update()
{
    environmentScript.UpdateItems(); //calls functoin to get items in environment

    if(!Input.GetMouseButtonDown(0) && !Input.GetMouseButtonDown(1) && !Input.GetMouseButton(0)) //checks if left or right mouse buttons are being pressed
    {
        environmentScript.ResetItemPositions(); //resets item positions to a location in the environment if items have been removed
    }

    ItemClickedCheck();

    SlotClickedCheck();

}
```

This function checks if the mouse is hovering over an item sprite and if the RMB is pressed or the LMB is held down.
It checks if the world coordinate mouse position is between the min/max positions on the x and y axis of the sprite for each item. If yes, it checks if the RMB is pressed and there is no other item selected (dragged), and if yes call ‘MoveItemToInventory()’ function to add item to the inventory. Otherwise check if the LMB is pressed, if yes set the ‘currentItemSelected’ to that item. In the next frame, while the LMB is held down on that item call the ‘FollowMouse()’ function from the ‘Item’ class passing the x and y positions of the mouse as arguments to make the item follow the mouse. In the frame where the LMB is released check if the mouse is hovering over a slot (if it is between the min/max x and y sprite positions), if yes call the ‘DropItemInInventory()’ function to add item to that slot.



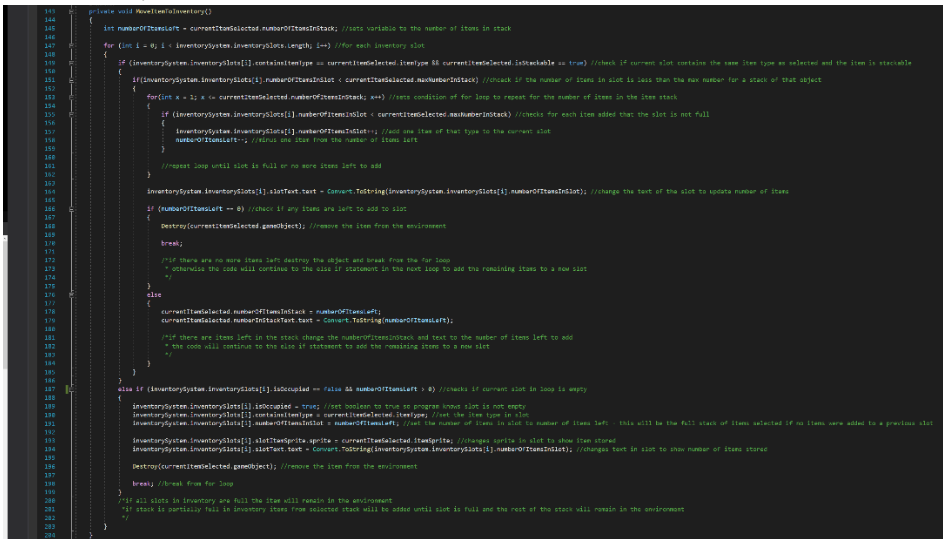
```
private void ItemClickedCheck()
{
    for (int i = 0; i < environmentScript.items.Length; i++) //for each item in the environment
    {
        if (
            Camera.main.ScreenToWorldPoint(Input.mousePosition).x > environmentScript.items[i].xMinPositionSprite && Camera.main.ScreenToWorldPoint(Input.mousePosition).x < environmentScript.items[i].xMaxPositionSprite
            &&
            Camera.main.ScreenToWorldPoint(Input.mousePosition).y > environmentScript.items[i].yMinPositionSprite && Camera.main.ScreenToWorldPoint(Input.mousePosition).y < environmentScript.items[i].yMaxPositionSprite
            ) //check if mouse is between the two x values for the item sprite and between the two y values for the item sprite
        {
            if (Input.GetMouseButtonDown(1) && currentItemSelected == null) //check if right mouse button is pressed
            {
                currentItemSelected = environmentScript.items[i]; //stores the item that the mouse hovered over when RMB was pressed

                MoveItemToInventory(); //call function to move item to inventory slot

                currentItemSelected = null; //reset to empty value
            }
            else if (Input.GetMouseButtonDown(0)) //check if left mouse button is pressed
            {
                currentItemSelected = environmentScript.items[i]; //set the selected item to the item where the mouse was left clicked
            }
            else if (Input.GetMouseButton(0) && currentItemSelected != null) //check if left mouse button is held down
            {
                currentItemSelected.FollowMouse(Camera.main.ScreenToWorldPoint(Input.mousePosition).x, Camera.main.ScreenToWorldPoint(Input.mousePosition).y); //call function to move item and pass mouse positions as parameters
            }
            else if (Input.GetMouseButtonUp(0) && currentItemSelected != null) //check if left mouse button is released
            {
                for (int x = 0; x < inventorySystem.inventorySlots.Length; x++) //for each slot sprite
                {
                    if (
                        Camera.main.ScreenToWorldPoint(Input.mousePosition).x > inventorySystem.inventorySlots[x].xMinPositionSlot && Camera.main.ScreenToWorldPoint(Input.mousePosition).x < inventorySystem.inventorySlots[x].xMaxPositionSlot
                        &&
                        Camera.main.ScreenToWorldPoint(Input.mousePosition).y > inventorySystem.inventorySlots[x].yMinPositionSlot && Camera.main.ScreenToWorldPoint(Input.mousePosition).y < inventorySystem.inventorySlots[x].yMaxPositionSlot
                        ) //check if mouse is between the two x values for the slot sprite and between the two y values for the slot sprite
                    {
                        currentSlotSelected = inventorySystem.inventorySlots[x]; //stores the slot that the mouse hovered over when LMB was released

                        DropItemInInventory(); //call function to drag and drop item in inventory
                    }
                }

                currentItemSelected = null; //reset to empty value
            }
        }
    }
}
```

This function checks if the mouse is hovering over a slot and if the RMB is pressed or the LMB is held down.
It checks if the world coordinate mouse position is between the min/max positions on the x and y axis of the sprite for each slot. If yes, it checks if the RMB is pressed and there is no item selected (being dragged), and if yes call ‘RemoveItemFromInventory()’ function to remove the item from inventory and instantiate it into the environment. Otherwise, check if the LMB is pressed, if yes call the ‘DragItemFromInventory()’ passing the x and y positions of the mouse as arguments to instantiate the items into the environment at the mouse position, and store the returned instantiated object in ‘currentItemSelected’. The ‘currentItemSelected is then used when the ‘Update()’ function calls the ‘ItemClickedCheck’ in the next frame to make the item follow the mouse while the LMB is held down by calling the ‘FollowMouse()’ function.



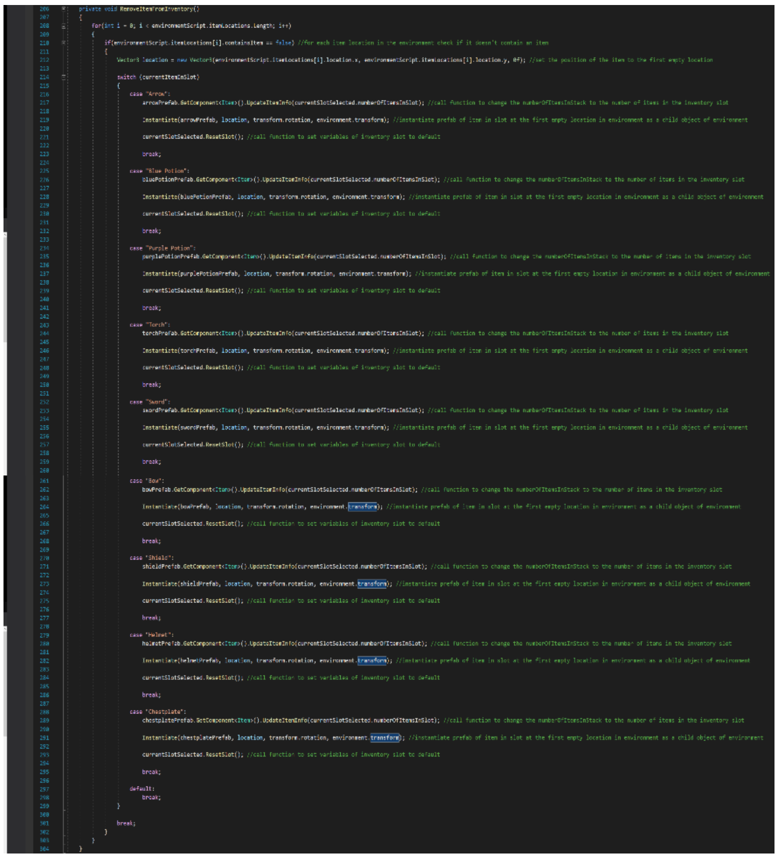
```
private void SlotClickedCheck()
{
    for (int i = 0; i < inventorySystem.inventorySlots.Length; i++) //for each slot in the inventory
    {
        if (
            Camera.main.ScreenToWorldPoint(Input.mousePosition).x > inventorySystem.inventorySlots[i].xMinPositionSlot && Camera.main.ScreenToWorldPoint(Input.mousePosition).x < inventorySystem.inventorySlots[i].xMaxPositionSlot
            &&
            Camera.main.ScreenToWorldPoint(Input.mousePosition).y > inventorySystem.inventorySlots[i].yMinPositionSlot && Camera.main.ScreenToWorldPoint(Input.mousePosition).y < inventorySystem.inventorySlots[i].yMaxPositionSlot
            ) //check if mouse is between the two x values for the slot sprite and between the two y values for the slot sprite
        {
            if (Input.GetMouseButtonDown(1) && currentItemSelected == null) //check if right mouse button is pressed
            {
                currentSlotSelected = inventorySystem.inventorySlots[i]; //stores the slot that the mouse hovered over when RMB was pressed
                currentItemInSlot = currentSlotSelected.containsItemType; //stores the type of object in the current slot selected

                RemoveItemFromInventory(); //call function to move item to environment

                currentSlotSelected = null; //reset to empty value
                currentItemInSlot = null; //reset to empty value
            }
            else if (Input.GetMouseButtonDown(0)) //check if left mouse button is pressed
            {
                currentSlotSelected = inventorySystem.inventorySlots[i]; //stores the slot that the mouse hovered over when RMB was pressed
                currentItemInSlot = currentSlotSelected.containsItemType; //stores the type of object in the current slot selected

                currentItemSelected = DragItemFromInventory(Camera.main.ScreenToWorldPoint(Input.mousePosition).x, Camera.main.ScreenToWorldPoint(Input.mousePosition).y).GetComponent<Item>();
                /*call function to drag item from inventory passing the mouse position as parameters
                 * store the returned instantiated item game object
                 */
            }
        }
    }
}
```

This function is called by the 'ItemClickedCheck()' function if the RMB is pressed on an item. It checks what the next available slot is to add the item that was right clicked to the inventory. It checks if a slot already contains that item type and if the maximum stack number hasn't been reached, if yes add items from the environment stack to the slot until the slot is full and set the 'numberOfItemsLeft' to the number of items left in the environment stack. Otherwise, check if a slot is empty and there are items still left to add, if yes set the slot values to the item values and destroy the item object (remove it from the environment/scene).



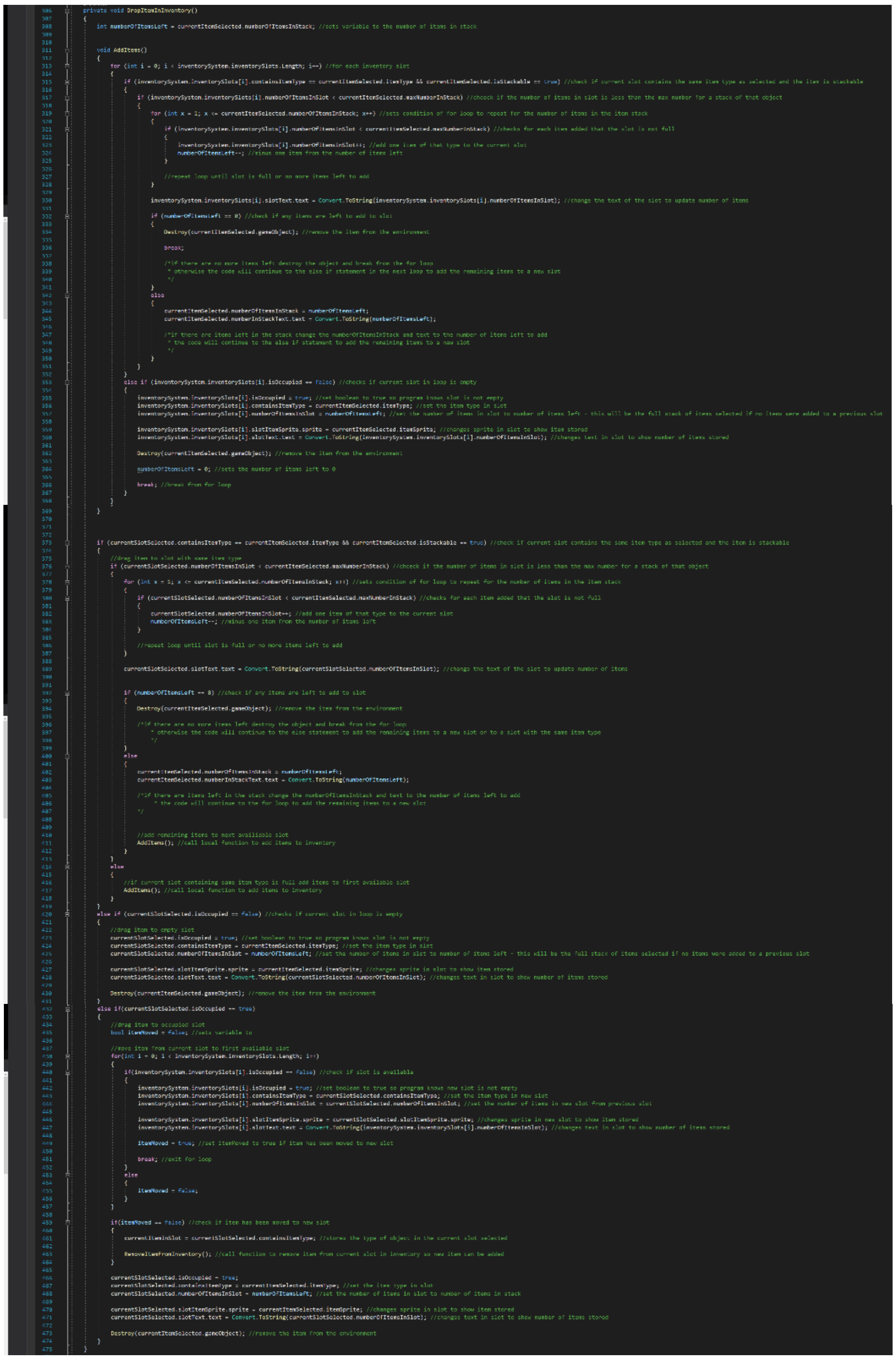
```csharp
private void MoveItemToInventory()
{
    int numberOfItemsLeft = currentItemSelected.numberOfItemsInStack; //sets variable to the number of items in stack

    for (int i = 0; i < inventorySystem.inventorySlots.Length; i++) //for each inventory slot
    {
        if (inventorySystem.inventorySlots[i].containsItemType == currentItemSelected.itemType && currentItemSelected.isStackable == true) //check if current slot contains the same item type as selected and the item is stackable
        {
            if(inventorySystem.inventorySlots[i].numberOfItemsInSlot < currentItemSelected.maxNumberInStack) //chcack if the number of items in slot is less than the max number for a stack of that object
            {
                for(int x = 1; x <= currentItemSelected.numberOfItemsInStack; x++) //sets condition of for loop to repeat for the number of items in the item stack
                {
                    if (inventorySystem.inventorySlots[i].numberOfItemsInSlot < currentItemSelected.maxNumberInStack) //checks for each item added that the slot is not full
                    {
                        inventorySystem.inventorySlots[i].numberOfItemsInSlot++; //add one item of that type to the current slot
                        numberOfItemsLeft--; //minus one item from the number of items left
                    }

                    //repeat loop until slot is full or no more items left to add
                }

                inventorySystem.inventorySlots[i].slotText.text = Convert.ToString(inventorySystem.inventorySlots[i].numberOfItemsInSlot); //change the text of the slot to update number of items

                if (numberOfItemsLeft == 0) //check if any items are left to add to slot
                {
                    Destroy(currentItemSelected.gameObject); //remove the item from the environment

                    break;

                    /*if there are no more items left destroy the object and break from the for loop
                     * otherwise the code will continue to the else if statement in the next loop to add the remaining items to a new slot
                     */
                }
                else
                {
                    currentItemSelected.numberOfItemsInStack = numberOfItemsLeft;
                    currentItemSelected.numberInStackText.text = Convert.ToString(numberOfItemsLeft);

                    /*if there are items left in the stack change the numberOfItemsInStack and text to the number of items left to add
                     * the code will continue to the else if statement to add the remaining items to a new slot
                     */
                }
            }
        }
        else if (inventorySystem.inventorySlots[i].isOccupied == false && numberOfItemsLeft > 0) //checks if current slot in loop is empty
        {
            inventorySystem.inventorySlots[i].isOccupied = true; //set boolean to true so program knows slot is not empty
            inventorySystem.inventorySlots[i].containsItemType = currentItemSelected.itemType; //set the item type in slot
            inventorySystem.inventorySlots[i].numberOfItemsInSlot = numberOfItemsLeft; //set the number of items in slot to number of items left - this will be the full stack of items selected if no items were added to a previous slot

            inventorySystem.inventorySlots[i].slotItemSprite.sprite = currentItemSelected.itemSprite; //changes sprite in slot to show item stored
            inventorySystem.inventorySlots[i].slotText.text = Convert.ToString(inventorySystem.inventorySlots[i].numberOfItemsInSlot); //changes text in slot to show number of items stored

            Destroy(currentItemSelected.gameObject); //remove the item from the environment

            break; //break from for loop
        }
        /*if all slots in inventory are full the item will remain in the environment
         *if stack is partially full in inventory items from selected stack will be added until slot is full and the rest of the stack will remain in the environment
         */
    }
}
```

This function is called by the 'SlotClickedCheck()' function if the RMB is pressed on a slot. It checks what type of item is stored in that slot and instantiates that item into the environment. It also calls the 'ResetSlot()' function from the 'InventorySlot' class to set the slot values to default when an item is removed.



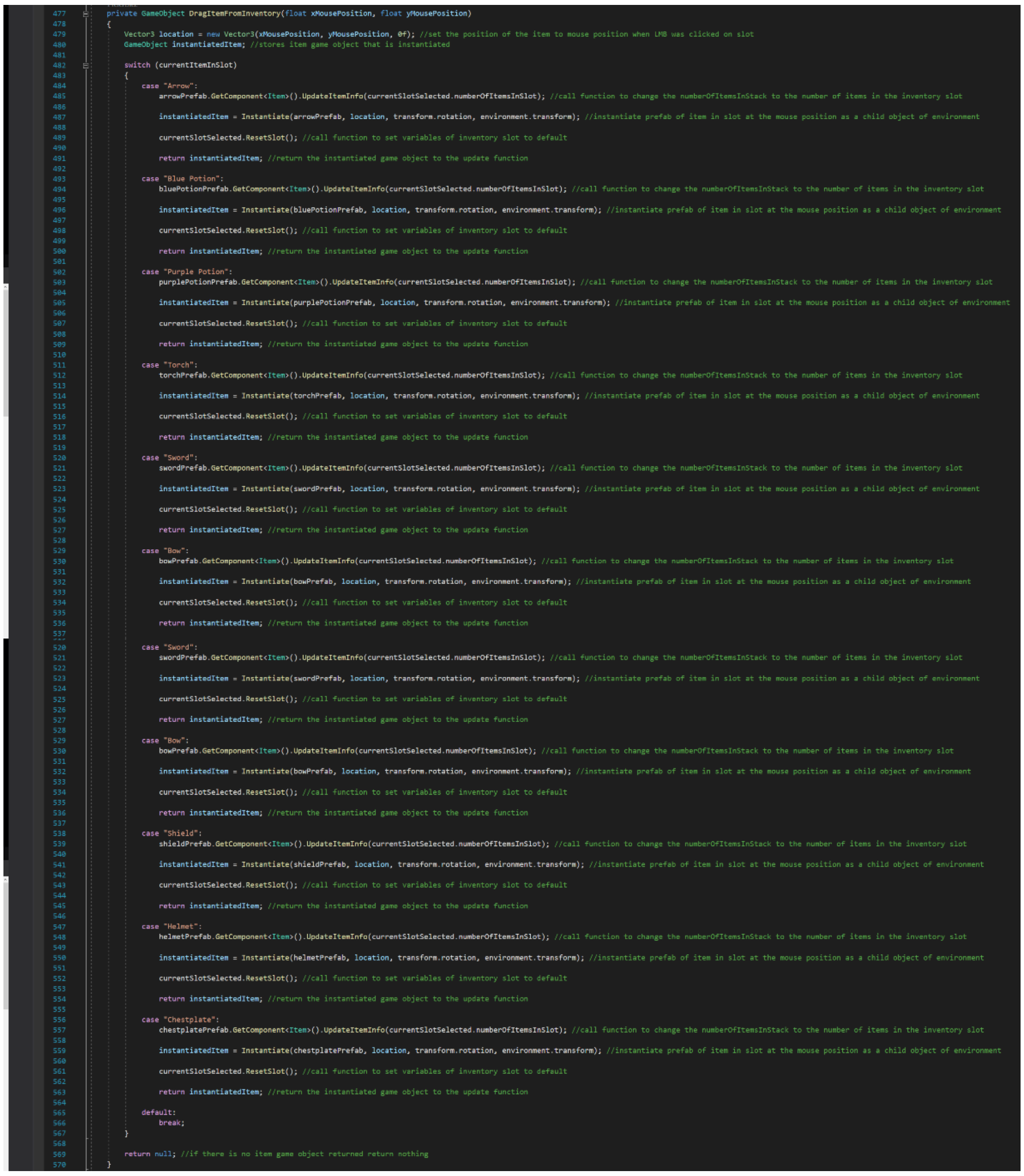
```csharp
private void RemoveItemFromInventory()
{
    for(int i = 0; i < environmentScript.itemLocations.Length; i++)
    {
        if(environmentScript.itemLocations[i].containsItem == false) //for each item location in the environment check if it doesn't contain an item
        {
            Vector3 location = new Vector3(environmentScript.itemLocations[i].location.x, environmentScript.itemLocations[i].location.y, 0f); //set the position of the item to the first empty location

            switch (currentItemInSlot)
            {
                case "Arrow":
                    arrowPrefab.GetComponent<Item>().UpdateItemInfo(currentSlotSelected.numberOfItemsInSlot); //call function to change the numberOfItemsInStack to the number of items in the inventory slot

                    Instantiate(arrowPrefab, location, transform.rotation, environment.transform); //instantiate prefab of item in slot at the first empty location in environment as a child object of environment

                    currentSlotSelected.ResetSlot(); //call function to set variables of inventory slot to default

                    break;

                case "Blue Potion":
                    bluePotionPrefab.GetComponent<Item>().UpdateItemInfo(currentSlotSelected.numberOfItemsInSlot); //call function to change the numberOfItemsInStack to the number of items in the inventory slot

                    Instantiate(bluePotionPrefab, location, transform.rotation, environment.transform); //instantiate prefab of item in slot at the first empty location in environment as a child object of environment

                    currentSlotSelected.ResetSlot(); //call function to set variables of inventory slot to default

                    break;

                case "Purple Potion":
                    purplePotionPrefab.GetComponent<Item>().UpdateItemInfo(currentSlotSelected.numberOfItemsInSlot); //call function to change the numberOfItemsInStack to the number of items in the inventory slot

                    Instantiate(purplePotionPrefab, location, transform.rotation, environment.transform); //instantiate prefab of item in slot at the first empty location in environment as a child object of environment

                    currentSlotSelected.ResetSlot(); //call function to set variables of inventory slot to default

                    break;

                case "Torch":
                    torchPrefab.GetComponent<Item>().UpdateItemInfo(currentSlotSelected.numberOfItemsInSlot); //call function to change the numberOfItemsInStack to the number of items in the inventory slot

                    Instantiate(torchPrefab, location, transform.rotation, environment.transform); //instantiate prefab of item in slot at the first empty location in environment as a child object of environment

                    currentSlotSelected.ResetSlot(); //call function to set variables of inventory slot to default

                    break;

                case "Sword":
                    swordPrefab.GetComponent<Item>().UpdateItemInfo(currentSlotSelected.numberOfItemsInSlot); //call function to change the numberOfItemsInStack to the number of items in the inventory slot

                    Instantiate(swordPrefab, location, transform.rotation, environment.transform); //instantiate prefab of item in slot at the first empty location in environment as a child object of environment

                    currentSlotSelected.ResetSlot(); //call function to set variables of inventory slot to default

                    break;

                case "Bow":
                    bowPrefab.GetComponent<Item>().UpdateItemInfo(currentSlotSelected.numberOfItemsInSlot); //call function to change the numberOfItemsInStack to the number of items in the inventory slot

                    Instantiate(bowPrefab, location, transform.rotation, environment.transform); //instantiate prefab of item in slot at the first empty location in environment as a child object of environment

                    currentSlotSelected.ResetSlot(); //call function to set variables of inventory slot to default

                    break;

                case "Shield":
                    shieldPrefab.GetComponent<Item>().UpdateItemInfo(currentSlotSelected.numberOfItemsInSlot); //call function to change the numberOfItemsInStack to the number of items in the inventory slot

                    Instantiate(shieldPrefab, location, transform.rotation, environment.transform); //instantiate prefab of item in slot at the first empty location in environment as a child object of environment

                    currentSlotSelected.ResetSlot(); //call function to set variables of inventory slot to default

                    break;

                case "Helmet":
                    helmetPrefab.GetComponent<Item>().UpdateItemInfo(currentSlotSelected.numberOfItemsInSlot); //call function to change the numberOfItemsInStack to the number of items in the inventory slot

                    Instantiate(helmetPrefab, location, transform.rotation, environment.transform); //instantiate prefab of item in slot at the first empty location in environment as a child object of environment

                    currentSlotSelected.ResetSlot(); //call function to set variables of inventory slot to default

                    break;

                case "Chestplate":
                    chestplatePrefab.GetComponent<Item>().UpdateItemInfo(currentSlotSelected.numberOfItemsInSlot); //call function to change the numberOfItemsInStack to the number of items in the inventory slot

                    Instantiate(chestplatePrefab, location, transform.rotation, environment.transform); //instantiate prefab of item in slot at the first empty location in environment as a child object of environment

                    currentSlotSelected.ResetSlot(); //call function to set variables of inventory slot to default

                    break;

                default:
                    break;
            }

            break;
        }
    }
}
```

This function is called by the ‘ItemClickedCheck()’ function when the LMB is pressed. It works similarly to the ‘MoveItemToInventory()’ but instead of finding the first empty slot in inventory it adds the dragged item to the slot where the LMB was released. First it checks if that slot contains the same item type already, if yes add items to that slot until it is full and then call local ‘AddItems()’ function to add the rest of the items to the first empty slot. This code is in a separate function within the function because it is called more than once in different if/else statements so it reduces the amount of repeating code, and it needs to have access to the local variable ‘numberOfItemsLeft’. Otherwise, if the slot is empty add the item to that slot. If the slot is occupied, move the current item in the slot to the first empty slot (or remove from inventory to environment by calling the ‘RemoveFromInventory()’ function if all slots are full) and add the dragged item to that slot.
The local ‘AddItems()’ function checks each inventory slot if it contains that item type to add the environment stack to it, or find an empty slot and add the item to that slot.



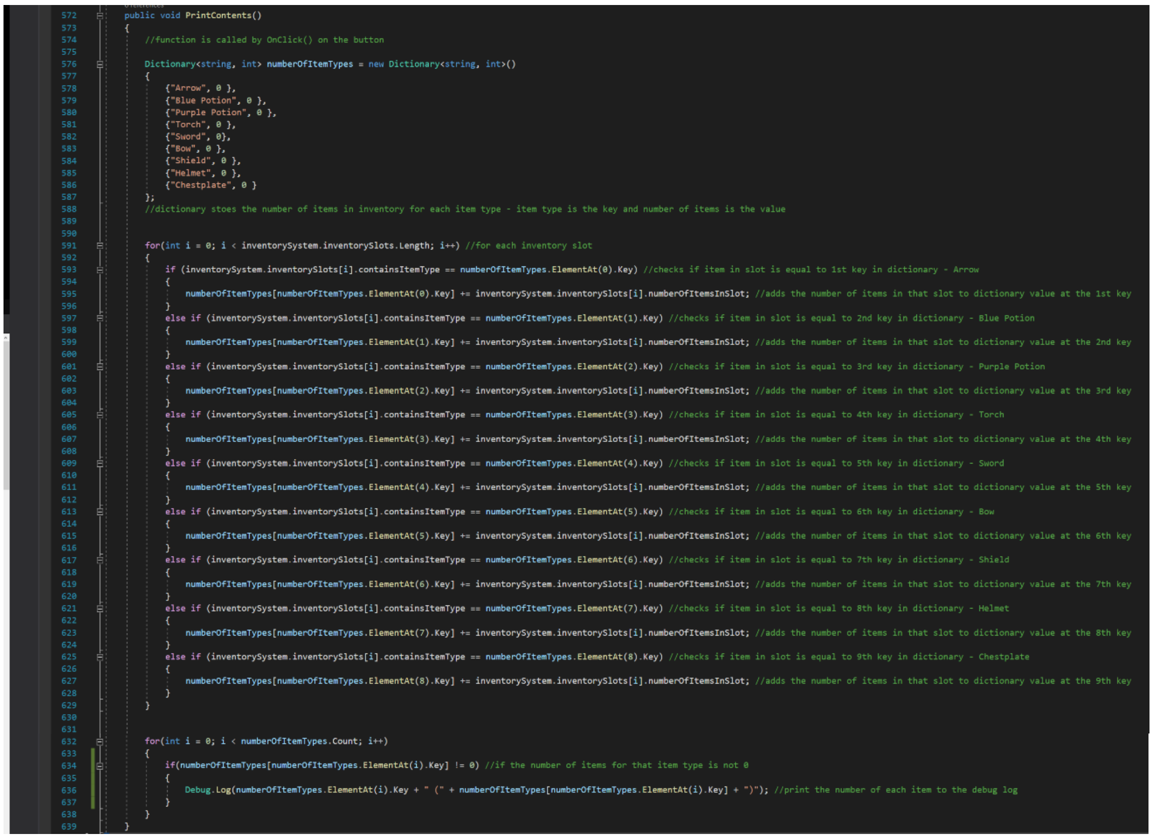
```
private void DropItemInInventory()
{
    int numberOfItemsLeft = currentItemSelected.numberOfItemsInStack; //sets variable to the number of items in stack


    void AddItems()
    {
        for (int i = 0; i < inventorySystem.inventorySlots.Length; i++) //for each inventory slot
        {
            if (inventorySystem.inventorySlots[i].containsItemType == currentItemSelected.itemType && currentItemSelected.isStackable == true) //check if current slot contains the same item type as selected and the item is stackable
            {
                if (inventorySystem.inventorySlots[i].numberOfItemsInSlot < currentItemSelected.maxNumberInStack) //chceck if the number of items in slot is less than the max number for a stack of that object
                {
                    for (int x = 1; x <= currentItemSelected.numberOfItemsInStack; x++) //sets condition of for loop to repeat for the number of items in the item stack
                    {
                        if (inventorySystem.inventorySlots[i].numberOfItemsInSlot < currentItemSelected.maxNumberInStack) //checks for each item added that the slot is not full
                        {
                            inventorySystem.inventorySlots[i].numberOfItemsInSlot++; //add one item of that type to the current slot
                            numberOfItemsLeft--; //minus one item from the number of items left
                        }

                        //repeat loop until slot is full or no more items left to add
                    }

                    inventorySystem.inventorySlots[i].slotText.text = Convert.ToString(inventorySystem.inventorySlots[i].numberOfItemsInSlot); //change the text of the slot to update number of items

                    if (numberOfItemsLeft == 0) //check if any items are left to add to slot
                    {
                        Destroy(currentItemSelected.gameObject); //remove the item from the environment

                        break;

                        /*if there are no more items left destroy the object and break from the for loop
                         * otherwise the code will continue to the else if statement in the next loop to add the remaining items to a new slot
                         */
                    }
                    else
                    {
                        currentItemSelected.numberOfItemsInStack = numberOfItemsLeft;
                        currentItemSelected.numberInStackText.text = Convert.ToString(numberOfItemsLeft);

                        /*if there are items left in the stack change the numberOfItemsInStack and text to the number of items left to add
                         * the code will continue to the else if statement to add the remaining items to a new slot
                         */
                    }
                }
            }
            else if (inventorySystem.inventorySlots[i].isOccupied == false) //checks if current slot in loop is empty
            {
                inventorySystem.inventorySlots[i].isOccupied = true; //set boolean to true so program knows slot is not empty
                inventorySystem.inventorySlots[i].containsItemType = currentItemSelected.itemType; //set the item type in slot
                inventorySystem.inventorySlots[i].numberOfItemsInSlot = numberOfItemsLeft; //set the number of items in slot to number of items left - this will be the full stack of items selected if no items were added to a previous slot

                inventorySystem.inventorySlots[i].slotItemSprite.sprite = currentItemSelected.itemSprite; //changes sprite in slot to show item stored
                inventorySystem.inventorySlots[i].slotText.text = Convert.ToString(inventorySystem.inventorySlots[i].numberOfItemsInSlot); //changes text in slot to show number of items stored

                Destroy(currentItemSelected.gameObject); //remove the item from the environment

                numberOfItemsLeft = 0; //sets the number of items left to 0

                break; //break from for loop
            }
        }
    }



    if (currentSlotSelected.containsItemType == currentItemSelected.itemType && currentItemSelected.isStackable == true) //check if current slot contains the same item type as selected and the item is stackable
    {
        //drag item to slot with same item type
        if (currentSlotSelected.numberOfItemsInSlot < currentItemSelected.maxNumberInStack) //chceck if the number of items in slot is less than the max number for a stack of that object
        {
            for (int x = 1; x <= currentItemSelected.numberOfItemsInStack; x++) //sets condition of for loop to repeat for the number of items in the item stack
            {
                if (currentSlotSelected.numberOfItemsInSlot < currentItemSelected.maxNumberInStack) //checks for each item added that the slot is not full
                {
                    currentSlotSelected.numberOfItemsInSlot++; //add one item of that type to the current slot
                    numberOfItemsLeft--; //minus one item from the number of items left
                }

                //repeat loop until slot is full or no more items left to add
            }

            currentSlotSelected.slotText.text = Convert.ToString(currentSlotSelected.numberOfItemsInSlot); //change the text of the slot to update number of items


            if (numberOfItemsLeft == 0) //check if any items are left to add to slot
            {
                Destroy(currentItemSelected.gameObject); //remove the item from the environment

                /*if there are no more items left destroy the object and break from the for loop
                    * otherwise the code will continue to the else statement to add the remaining items to a new slot or to a slot with the same item type
                    */
            }
            else
            {
                currentItemSelected.numberOfItemsInStack = numberOfItemsLeft;
                currentItemSelected.numberInStackText.text = Convert.ToString(numberOfItemsLeft);

                /*if there are items left in the stack change the numberOfItemsInStack and text to the number of items left to add
                    * the code will continue to the for loop to add the remaining items to a new slot
                */


                //add remaining items to next availiable slot
                AddItems(); //call local function to add items to inventory
            }
        }
        else
        {
            //if current slot containing same item type is full add items to first available slot
            AddItems(); //call local function to add items to inventory
        }
    }
    else if (currentSlotSelected.isOccupied == false) //checks if current slot in loop is empty
    {
        //drag item to empty slot
        currentSlotSelected.isOccupied = true; //set boolean to true so program knows slot is not empty
        currentSlotSelected.containsItemType = currentItemSelected.itemType; //set the item type in slot
        currentSlotSelected.numberOfItemsInSlot = numberOfItemsLeft; //set the number of items in slot to number of items left - this will be the full stack of items selected if no items were added to a previous slot

        currentSlotSelected.slotItemSprite.sprite = currentItemSelected.itemSprite; //changes sprite in slot to show item stored
        currentSlotSelected.slotText.text = Convert.ToString(currentSlotSelected.numberOfItemsInSlot); //changes text in slot to show number of items stored

        Destroy(currentItemSelected.gameObject); //remove the item from the environment
    }
    else if(currentSlotSelected.isOccupied == true)
    {
        //drag item to occupied slot
        bool itemMoved = false; //sets variable to

        //move item from current slot to first available slot
        for(int i = 0; i < inventorySystem.inventorySlots.Length; i++)
        {
            if(inventorySystem.inventorySlots[i].isOccupied == false) //check if slot is available
            {
                inventorySystem.inventorySlots[i].isOccupied = true; //set boolean to true so program knows new slot is not empty
                inventorySystem.inventorySlots[i].containsItemType = currentSlotSelected.containsItemType; //set the item type in new slot
                inventorySystem.inventorySlots[i].numberOfItemsInSlot = currentSlotSelected.numberOfItemsInSlot; //set the number of items in new slot from previous slot

                inventorySystem.inventorySlots[i].slotItemSprite.sprite = currentSlotSelected.slotItemSprite.sprite; //changes sprite in new slot to show item stored
                inventorySystem.inventorySlots[i].slotText.text = Convert.ToString(inventorySystem.inventorySlots[i].numberOfItemsInSlot); //changes text in slot to show number of items stored

                itemMoved = true; //set itemMoved to true if item has been moved to new slot

                break; //exit for loop
            }
            else
            {
                itemMoved = false;
            }
        }

        if(itemMoved == false) //check if item has been moved to new slot
        {
            currentItemInSlot = currentSlotSelected.containsItemType; //stores the type of object in the current slot selected

            RemoveItemFromInventory(); //call function to remove item from current slot in inventory so new item can be added
        }

        currentSlotSelected.isOccupied = true;
        currentSlotSelected.containsItemType = currentItemSelected.itemType; //set the item type in slot
        currentSlotSelected.numberOfItemsInSlot = numberOfItemsLeft; //set the number of items in slot to number of items in stack

        currentSlotSelected.slotItemSprite.sprite = currentItemSelected.itemSprite; //changes sprite in slot to show item stored
        currentSlotSelected.slotText.text = Convert.ToString(currentSlotSelected.numberOfItemsInSlot); //changes text in slot to show number of items stored

        Destroy(currentItemSelected.gameObject); //remove the item from the environment
    }
}
```

This function is called by the ‘SlotClickedCheck’ when the LMB is pressed. It works similarly to the ‘RemoveItemFromInventory()’ function but instead of instantiating the item at a location in the environment, it instantiates the item at the position of the mouse so it can be dragged. It takes the x an y positions of the mouse as parameters and instantiates the item at those positions. It also returns the instantiated object which is stored in ‘currentItemSelected’ so it can be dragged while the LMB is held down. The function checks the type of item stored in that inventory slot and instantiates the prefab for that item type.



```
private GameObject DragItemFromInventory(float xMousePosition, float yMousePosition)
{
    Vector3 location = new Vector3(xMousePosition, yMousePosition, 0f); //set the position of the item to mouse position when LMB was clicked on slot
    GameObject instantiatedItem; //stores item game object that is instantiated

    switch (currentItemInSlot)
    {
        case "Arrow":
            arrowPrefab.GetComponent<Item>().UpdateItemInfo(currentSlotSelected.numberOfItemsInSlot); //call function to change the numberOfItemsInStack to the number of items in the inventory slot

            instantiatedItem = Instantiate(arrowPrefab, location, transform.rotation, environment.transform); //instantiate prefab of item in slot at the mouse position as a child object of environment

            currentSlotSelected.ResetSlot(); //call function to set variables of inventory slot to default

            return instantiatedItem; //return the instantiated game object to the update function

        case "Blue Potion":
            bluePotionPrefab.GetComponent<Item>().UpdateItemInfo(currentSlotSelected.numberOfItemsInSlot); //call function to change the numberOfItemsInStack to the number of items in the inventory slot

            instantiatedItem = Instantiate(bluePotionPrefab, location, transform.rotation, environment.transform); //instantiate prefab of item in slot at the mouse position as a child object of environment

            currentSlotSelected.ResetSlot(); //call function to set variables of inventory slot to default

            return instantiatedItem; //return the instantiated game object to the update function

        case "Purple Potion":
            purplePotionPrefab.GetComponent<Item>().UpdateItemInfo(currentSlotSelected.numberOfItemsInSlot); //call function to change the numberOfItemsInStack to the number of items in the inventory slot

            instantiatedItem = Instantiate(purplePotionPrefab, location, transform.rotation, environment.transform); //instantiate prefab of item in slot at the mouse position as a child object of environment

            currentSlotSelected.ResetSlot(); //call function to set variables of inventory slot to default

            return instantiatedItem; //return the instantiated game object to the update function

        case "Torch":
            torchPrefab.GetComponent<Item>().UpdateItemInfo(currentSlotSelected.numberOfItemsInSlot); //call function to change the numberOfItemsInStack to the number of items in the inventory slot

            instantiatedItem = Instantiate(torchPrefab, location, transform.rotation, environment.transform); //instantiate prefab of item in slot at the mouse position as a child object of environment

            currentSlotSelected.ResetSlot(); //call function to set variables of inventory slot to default

            return instantiatedItem; //return the instantiated game object to the update function

        case "Sword":
            swordPrefab.GetComponent<Item>().UpdateItemInfo(currentSlotSelected.numberOfItemsInSlot); //call function to change the numberOfItemsInStack to the number of items in the inventory slot

            instantiatedItem = Instantiate(swordPrefab, location, transform.rotation, environment.transform); //instantiate prefab of item in slot at the mouse position as a child object of environment

            currentSlotSelected.ResetSlot(); //call function to set variables of inventory slot to default

            return instantiatedItem; //return the instantiated game object to the update function

        case "Bow":
            bowPrefab.GetComponent<Item>().UpdateItemInfo(currentSlotSelected.numberOfItemsInSlot); //call function to change the numberOfItemsInStack to the number of items in the inventory slot

            instantiatedItem = Instantiate(bowPrefab, location, transform.rotation, environment.transform); //instantiate prefab of item in slot at the mouse position as a child object of environment

            currentSlotSelected.ResetSlot(); //call function to set variables of inventory slot to default

            return instantiatedItem; //return the instantiated game object to the update function

        case "Sword":
            swordPrefab.GetComponent<Item>().UpdateItemInfo(currentSlotSelected.numberOfItemsInSlot); //call function to change the numberOfItemsInStack to the number of items in the inventory slot

            instantiatedItem = Instantiate(swordPrefab, location, transform.rotation, environment.transform); //instantiate prefab of item in slot at the mouse position as a child object of environment

            currentSlotSelected.ResetSlot(); //call function to set variables of inventory slot to default

            return instantiatedItem; //return the instantiated game object to the update function

        case "Bow":
            bowPrefab.GetComponent<Item>().UpdateItemInfo(currentSlotSelected.numberOfItemsInSlot); //call function to change the numberOfItemsInStack to the number of items in the inventory slot

            instantiatedItem = Instantiate(bowPrefab, location, transform.rotation, environment.transform); //instantiate prefab of item in slot at the mouse position as a child object of environment

            currentSlotSelected.ResetSlot(); //call function to set variables of inventory slot to default

            return instantiatedItem; //return the instantiated game object to the update function

        case "Shield":
            shieldPrefab.GetComponent<Item>().UpdateItemInfo(currentSlotSelected.numberOfItemsInSlot); //call function to change the numberOfItemsInStack to the number of items in the inventory slot

            instantiatedItem = Instantiate(shieldPrefab, location, transform.rotation, environment.transform); //instantiate prefab of item in slot at the mouse position as a child object of environment

            currentSlotSelected.ResetSlot(); //call function to set variables of inventory slot to default

            return instantiatedItem; //return the instantiated game object to the update function

        case "Helmet":
            helmetPrefab.GetComponent<Item>().UpdateItemInfo(currentSlotSelected.numberOfItemsInSlot); //call function to change the numberOfItemsInStack to the number of items in the inventory slot

            instantiatedItem = Instantiate(helmetPrefab, location, transform.rotation, environment.transform); //instantiate prefab of item in slot at the mouse position as a child object of environment

            currentSlotSelected.ResetSlot(); //call function to set variables of inventory slot to default

            return instantiatedItem; //return the instantiated game object to the update function

        case "Chestplate":
            chestplatePrefab.GetComponent<Item>().UpdateItemInfo(currentSlotSelected.numberOfItemsInSlot); //call function to change the numberOfItemsInStack to the number of items in the inventory slot

            instantiatedItem = Instantiate(chestplatePrefab, location, transform.rotation, environment.transform); //instantiate prefab of item in slot at the mouse position as a child object of environment

            currentSlotSelected.ResetSlot(); //call function to set variables of inventory slot to default

            return instantiatedItem; //return the instantiated game object to the update function

        default:
            break;
    }

    return null; //if there is no item game object returned return nothing
}
```

This function is called by the built-in ‘OnClick()’ function on the UI button. It is used to print the contents of the inventory to the console in Unity. A dictionary stores the item types as keys and the number of each item type in the inventory as the values for those keys. It checks what item type is stored in each slot and adds the number of that item to the value stored in the dictionary for that item type. It then prints each item type and number (if there is at least one in the inventory) to the console.



```
public void PrintContents()
{
    //function is called by OnClick() on the button

    Dictionary<string, int> numberOfItemTypes = new Dictionary<string, int>()
    {
        {"Arrow", 0 },
        {"Blue Potion", 0 },
        {"Purple Potion", 0 },
        {"Torch", 0 },
        {"Sword", 0},
        {"Bow", 0 },
        {"Shield", 0 },
        {"Helmet", 0 },
        {"Chestplate", 0 }
    };
    //dictionary stoes the number of items in inventory for each item type - item type is the key and number of items is the value


    for(int i = 0; i < inventorySystem.inventorySlots.Length; i++) //for each inventory slot
    {
        if (inventorySystem.inventorySlots[i].containsItemType == numberOfItemTypes.ElementAt(0).Key) //checks if item in slot is equal to 1st key in dictionary - Arrow
        {
            numberOfItemTypes[numberOfItemTypes.ElementAt(0).Key] += inventorySystem.inventorySlots[i].numberOfItemsInSlot; //adds the number of items in that slot to dictionary value at the 1st key
        }
        else if (inventorySystem.inventorySlots[i].containsItemType == numberOfItemTypes.ElementAt(1).Key) //checks if item in slot is equal to 2nd key in dictionary - Blue Potion
        {
            numberOfItemTypes[numberOfItemTypes.ElementAt(1).Key] += inventorySystem.inventorySlots[i].numberOfItemsInSlot; //adds the number of items in that slot to dictionary value at the 2nd key
        }
        else if (inventorySystem.inventorySlots[i].containsItemType == numberOfItemTypes.ElementAt(2).Key) //checks if item in slot is equal to 3rd key in dictionary - Purple Potion
        {
            numberOfItemTypes[numberOfItemTypes.ElementAt(2).Key] += inventorySystem.inventorySlots[i].numberOfItemsInSlot; //adds the number of items in that slot to dictionary value at the 3rd key
        }
        else if (inventorySystem.inventorySlots[i].containsItemType == numberOfItemTypes.ElementAt(3).Key) //checks if item in slot is equal to 4th key in dictionary - Torch
        {
            numberOfItemTypes[numberOfItemTypes.ElementAt(3).Key] += inventorySystem.inventorySlots[i].numberOfItemsInSlot; //adds the number of items in that slot to dictionary value at the 4th key
        }
        else if (inventorySystem.inventorySlots[i].containsItemType == numberOfItemTypes.ElementAt(4).Key) //checks if item in slot is equal to 5th key in dictionary - Sword
        {
            numberOfItemTypes[numberOfItemTypes.ElementAt(4).Key] += inventorySystem.inventorySlots[i].numberOfItemsInSlot; //adds the number of items in that slot to dictionary value at the 5th key
        }
        else if (inventorySystem.inventorySlots[i].containsItemType == numberOfItemTypes.ElementAt(5).Key) //checks if item in slot is equal to 6th key in dictionary - Bow
        {
            numberOfItemTypes[numberOfItemTypes.ElementAt(5).Key] += inventorySystem.inventorySlots[i].numberOfItemsInSlot; //adds the number of items in that slot to dictionary value at the 6th key
        }
        else if (inventorySystem.inventorySlots[i].containsItemType == numberOfItemTypes.ElementAt(6).Key) //checks if item in slot is equal to 7th key in dictionary - Shield
        {
            numberOfItemTypes[numberOfItemTypes.ElementAt(6).Key] += inventorySystem.inventorySlots[i].numberOfItemsInSlot; //adds the number of items in that slot to dictionary value at the 7th key
        }
        else if (inventorySystem.inventorySlots[i].containsItemType == numberOfItemTypes.ElementAt(7).Key) //checks if item in slot is equal to 8th key in dictionary - Helmet
        {
            numberOfItemTypes[numberOfItemTypes.ElementAt(7).Key] += inventorySystem.inventorySlots[i].numberOfItemsInSlot; //adds the number of items in that slot to dictionary value at the 8th key
        }
        else if (inventorySystem.inventorySlots[i].containsItemType == numberOfItemTypes.ElementAt(8).Key) //checks if item in slot is equal to 9th key in dictionary - Chestplate
        {
            numberOfItemTypes[numberOfItemTypes.ElementAt(8).Key] += inventorySystem.inventorySlots[i].numberOfItemsInSlot; //adds the number of items in that slot to dictionary value at the 9th key
        }
    }


    for(int i = 0; i < numberOfItemTypes.Count; i++)
    {
        if(numberOfItemTypes[numberOfItemTypes.ElementAt(i).Key] != 0) //if the number of items for that item type is not 0
        {
            Debug.Log(numberOfItemTypes.ElementAt(i).Key + " (" + numberOfItemTypes[numberOfItemTypes.ElementAt(i).Key] + ")"); //print the number of each item to the debug log
        }
    }
}
```