**Intro to Game Programming**
**Instructor: Caitlyn Lenhoff**
**Due: December 20th, 2020**

**Zoe Bailey**
**Jonathan Hansen**
**Zachary Law**

**Corrupted.exe: A Sci-Fi Shooter-Survival Game**

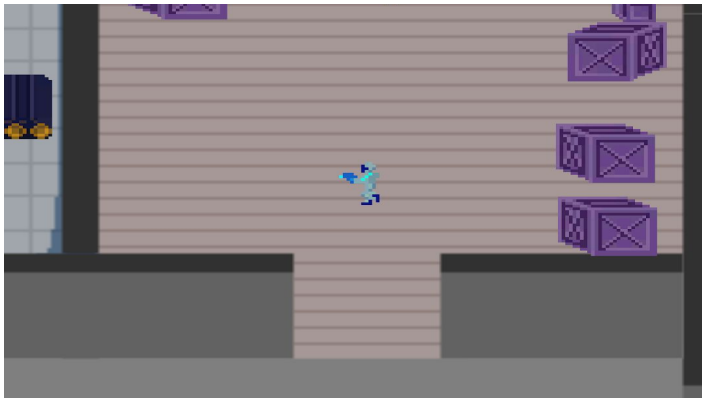
[Quigmodacon/Final_Project]

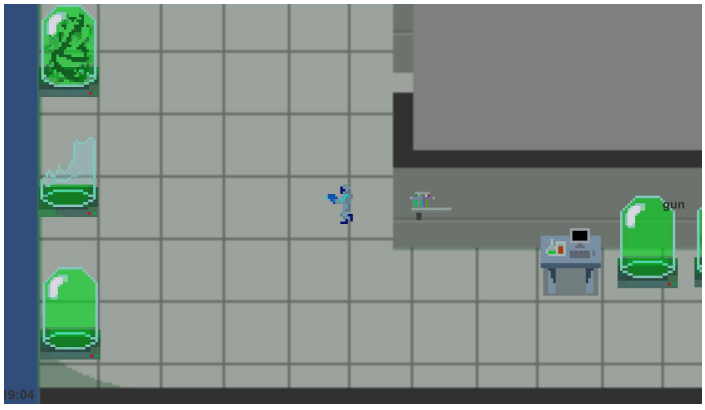
## Gameplay

Our project is a sci-fi shooter game. You play as a “janitor” type cyborg who goes around and purges dark magic that has infected the technology of the world. Using tab you can switch between tools (though right now only the gun has any functionality in the game) to do various tasks. The main task in our current version would be killing enemies though their graphics are currently temporary. Other tasks would be puzzles that are done to purge the environment of dark magic but those are not implemented at the moment.

Killing enemies or breaking boxes drops craftable pieces that can be crafted into better weapons. Currently the main goal is to just survive as long as possible.

**Key Bindings:**
**Left Click -** Interact/Shoot
**Up Key -** Up
**Down Key -** Down
**Right Key -** Right
**Left Key -** Left
**I -** Open/Close Inventory
**Tab -** Change Tool
**Esc -** Pause



```
public class InventoryPanel : MonoBehaviour
[
    [SerializeField] ItemContainer inventory;
    [SerializeField] List<InventoryButton> buttons;

    private void Start()
    {
        Show();
        SetIndex();
    }

    private void FixedUpdate()
    {
        Show();
    }
    private void SetIndex()
    {
        for (int i = 0; i < inventory.slots.Count; i++)
        {
            buttons[i].SetIndex(i);
        }
    }

    private void Show()
    {
        for (int i = 0; i < inventory.slots.Count; i++)
        {
            if (inventory.slots[i].item == null)
            {
                buttons[i].Clean();
            }
            else
            {
                buttons[i].Set(inventory.slots[i]);
            }
            if(inventory.slots[i].count == 0
            && inventory.slots[i].item.stackable) { buttons[i].Clean(); }
        }
    }
    public void SubtractItem(int index, int number)
    {
        inventory.slots[index].count -= number;
    }


    public Item GetItem(int index) { return inventory.slots[index].item; }
    public int GetCount(int index) { return inventory.slots[index].count; }
]
```

```
public class InventoryButton : MonoBehaviour
{
    [SerializeField] Item spell;
    [SerializeField] Image icon;
    [SerializeField] Text text;

    int myIndex;


    public void SetIndex(int index)
    {
        myIndex = index;
    }

    public void Set(ItemSlot slot)
    {
        icon.gameObject.SetActive(true);
        icon.sprite = slot.item.icon;

        if (slot.item.stackable == true)
        {
            text.gameObject.SetActive(true);
            text.text = slot.count.ToString();
        }
        else
        {
            text.gameObject.SetActive(false);
        }
    }

    public void Clean()
    {
        icon.sprite = null;
        icon.gameObject.SetActive(false);

        text.gameObject.SetActive(false);
    }

    public void Craft()
    {//statement to check if the item and if more
        if(GameManager.instance.inventoryContainer.slots[myIndex].item.name =="CorruptShard"
        && GameManager.instance.inventoryContainer.slots[myIndex].count >= 5 )
        {
            GameManager.instance.inventoryContainer.slots[myIndex].count -= 5;
            GameManager.instance.inventoryContainer .Add(spell, 1);
        }
    }
}
```

**The Inventory**

We used a few scriptable objects to make an inventory system capable of storing items dropped by enemies or found in crates/chests. The inventory is capable of keeping track of how many of any item you have in that inventory in real time during runtime. A crafting system on a conceptual level was also developed as to where the inventory checks whether or not it has a specific item and if it has enough of that specific item. Moving forward we would make a game object database that contained a dictionary of items(another scriptable object) that our game has. So calling items into code would not be as complicated and we simply need to reference the game object which could even further reduce redundancy by adding it into our GameManager system.

Clicking on “shards” within the inventory crafts them into a spell. Going forward, a tooltip that explains what each item does could be used to create a better user experience.

```
public class ItemSlot
{
    public Item item;
    public int count;
}

[CreateAssetMenu(menuName = "Data/Item Container")]

public class ItemContainer : ScriptableObject
{
    public List<ItemSlot> slots;

    public void Add(Item item, int count = 1)
    {
        if (item.stackable == true)
        {
            ItemSlot itemSlot = slots.Find(x => x.item == item);
            if (itemSlot != null)
            {
                itemSlot.count += count;
            }
            else
            {
                itemSlot = slots.Find(x => x.item == null);
                if (itemSlot != null)
                {
                    itemSlot.item = item;
                    itemSlot.count = count;
                }
            }
        }
        else
        {
            ItemSlot itemSlot = slots.Find(x => x.item == null);
            if (itemSlot != null)
            {
                itemSlot.item = item;
            }
        }

    }
}
```

**Top Left -** Code for the inventory panel
**Bottom Left -** Code controlling buttons within the inventory
**Center -** Code for items within the inv.
**Right -** Inventory