Kreiranje protivnika i scene za borbu

- Napraviti scenu **Battle** (File->New Scene), dodati pozadinu **background02.png** na nju. Dodati još neke sličice na pozadinu, po izboru, voditi računa o layerima.

- Podeliti sličicu goblina 05.png na sprajtove pomoću **Sprite Editor-a** (označiti sliku pa u Inspectoru prvo označiti SpriteMode: Multiple pa kliknuti na Sprite Editor)**,** prevući dobijeni sprajt **05_03** na scenu i zatim preimenovati novi objekat u **Goblin** (goblin će izgledati kao na sledećoj slici):

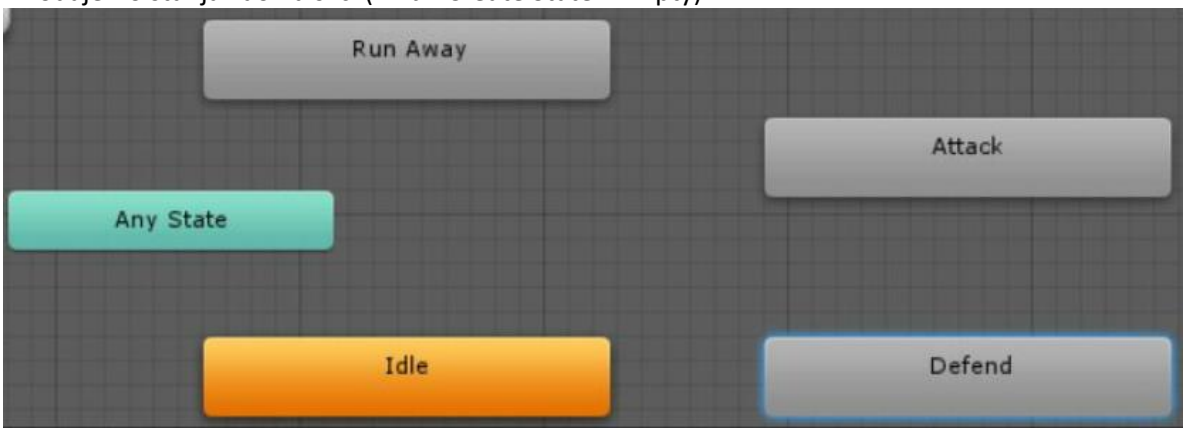
- Potrebno je dodati neku logiku ovom goblinu, to se neće koristiti u ovom poglavlju (tek u poglavlju 10) ali je dobro uraditi ovo na samom početku: unutar foldera Assets\Animation\Controllers napraviti novi **AnimatorController** (rmb->create->Animator Controller) - ovo zapravo predstavlja mašinu stanja za goblina. Dati mu naziv **GoblinAI**.

- Otvoriti GoblinAI. Dodati parametre kako bismo kontrolisali mašinu stanja (kliknuti na + unutar Parameters dela):

Tipovi ovih parametara:

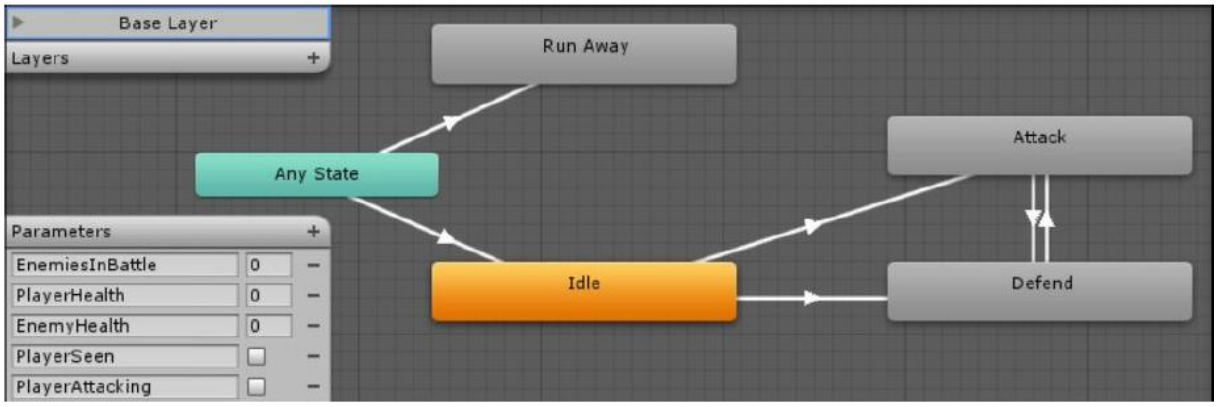
- EnemiesInBattle: Int
- PlayerHealth: Int
- EnemyHealth: Int
- PlayerSeen: Bool
- PlayerAttacking: Bool

- Dodajemo stanja kao na slici (rmb->Create State->Empty):



Stanje koje je obojeno u narandžasto je default tj. startno stanje. Moguće je promeniti default stanje klikom na njega pa Set As Default.

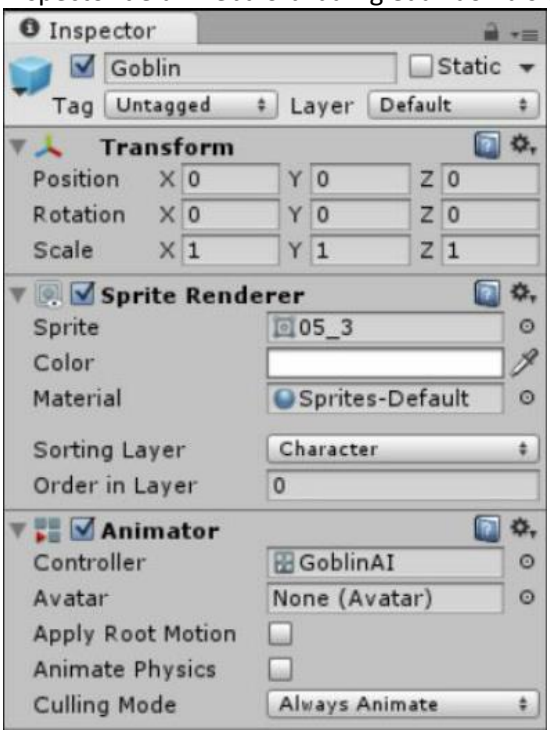
- Napraviti veze između stanja kao na slici:



Za svaku vezu je potrebno podesiti **Conditions** (u Inspectoru) na sledeći način:

- Idle | Attack, PlayerSeen - true (napadni igrača kada ga vidiš)
- Idle | Defend, PlayerSeen - true, PlayerAttacking - true (ako igrač prvi napadne, brani se)
- Attack | Defend, PlayerAttacking - true (ako igrač napadne, prebaci se iz stanja napada u stanje odbrane)
- Defend | Attack, PlayerAttacking - false (kada igrač prestane sa napadom, prebaci se iz stanja odbrane u stanje napada)
- Any State | Idle, PlayerSeen - false (ako izgubiš igrača iz vida, prebaci se u stanje mirovanja)
- Any State | Run Away, EnemyHealth > 0, EnemyHealth < 2, PlayerHealth > 2 (beži ako imaš zdravlje <2, a igrač ima zdravlje >2)

- Selektovati objekat Goblin unutar Hierarchy dela (leva strana); unutar Inspector dela dodati novu komponentu **Animator**; prevući fajl GoblinAI iz foldera Controllers na Animator komponentu u Inspector delu. Trebalo bi da izgleda kao na sledećoj slici:

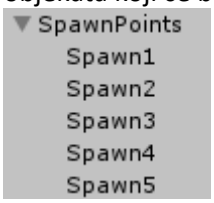


- Sada je potrebno od ovog goblina napraviti prefab - to se radi tako što prevučemo objekat Goblin iz Hierarchy dela u folder Assets\Prefabs\Characters. Obrisati objekat Goblin iz Hierarchy dela.

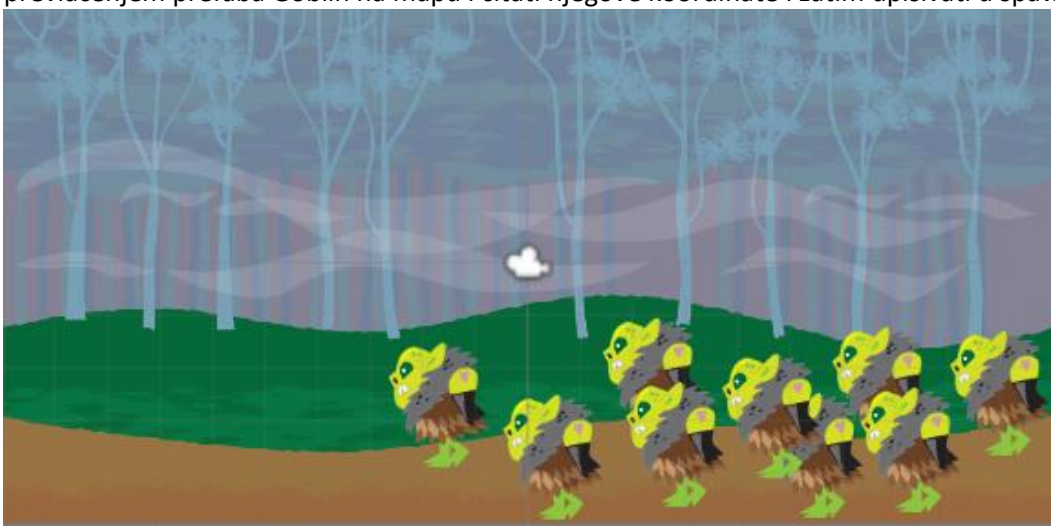
Ukoliko želimo da editujemo prefab, to radimo tako što selektujemo objekat u Characters folderu i zatim izvršimo izmene u Inspector delu - promene će se reflektovati na sve instance objekta na svim scenama gde se objekat nalazi. Ukoliko selektujemo objekat na samoj sceni pa vršimo izmene u Inspector-u, tada će se promene reflektovati samo na taj konkretan objekat.

Postavljanje spawn pozicija i upravljanje borbom

- Potrebno je da postavimo spawn (startne) pozicije na sceni na kojima će se pojaviti neprijatelji. Prvo pravimo novi objekat kog nazivamo SpawnPoints unutar Hierarchy dela (služiće kao kontejner koji će čuvati sve spawn-ove) i postavimo njegove Position koordinate na 0, 0, 0. Zatim pravimo 5 novih objekata koji će biti deca objektu SpawnPoints i nazivamo ih na sledeći način:



- Svakom spawnu dodeliti x i y koordinatu mesta gde će se nalaziti Goblin (to je lako moguće uraditi prevlačenjem prefaba Goblin na mapu i čitati njegove koordinate i zatim upisivati u spawn):



- Pošto smo poređali gobline, potrebno je da im naredimo da se na tim mestima i pojave preko skripte **BattleManager.cs** (napraviti unutar Assets/Scripts foldera). Svrha ove skripte je upravljanje svim aspektima borbe - od postavljanja scene i protivnika pa do same borbe. Ona radi samo kada smo u borbi. Trenutno, naša skripta će popuniti scenu sa protivnicima, a naš heroj će moći samo da pobegne.

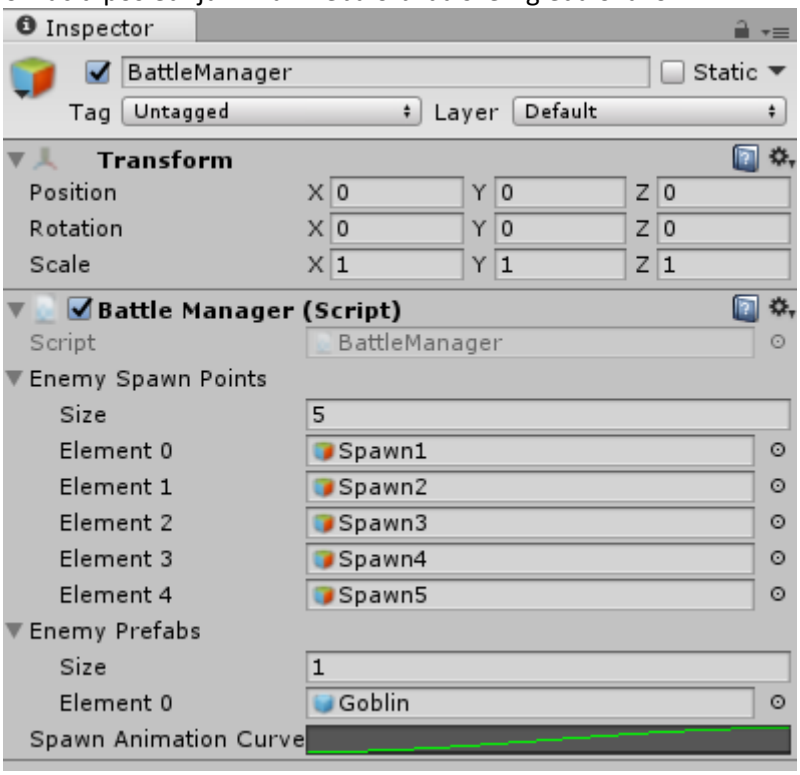
BattleManager.cs:

```
using System.Collections;
using UnityEngine;

public class BattleManager : MonoBehaviour {
```

```
    public GameObject[] EnemySpawnPoints; //spawn pozicije
    public GameObject[] EnemyPrefabs; //protivnici
    public AnimationCurve SpawnAnimationCurve; //koristimo AnimationCurve kako bismo
omogucili goblinima da (tecno) dosetaju od pozicije van ekrana do spawn pozicije

    private int enemyCount; //broj aktivnih protivnika na sceni
    enum BattlePhase
    {
        PlayerAttack,
        EnemyAttack
    }
    private BattlePhase phase; //fleg koji nam govori ko je u fazi napada

    // inicijalizacija scene za borbu - generise se slucajan broj goblina na mapi
    // zatim se oni postavljaju na spawn pozicije preko Coroutine poziva i zapocinje
se bitka tako sto igrac napadne prvi
    void Start ()
    {
        // broj generisanih protivnika po borbi
        enemyCount = Random.Range(1, EnemySpawnPoints.Length);

        // postavlja protivnike na spawn pozicije
        StartCoroutine("SpawnEnemies");

        // postavlja fleg za pocetno stanje bitke
        phase = BattlePhase.PlayerAttack;
    }

    IEnumerator SpawnEnemies()
    {
        // pravi gobline koji se krecu ka svojim spawn pozicijama, dolazeci van ekrana
        for (int i = 0; i < enemyCount; i++)
        {
            var newEnemy = (GameObject)Instantiate(EnemyPrefabs[0]); // novi protivnik
            newEnemy.transform.position = new Vector3(10, -3, 0); // setuje se pocetna
pozicija van ekrana
            yield return StartCoroutine(MoveCharacterToPoint(EnemySpawnPoints[i],
newEnemy)); // goblin dolazi na spawn poziciju
            newEnemy.transform.parent = EnemySpawnPoints[i].transform;
        }
    }

    IEnumerator MoveCharacterToPoint(GameObject destination, GameObject character)
    {
        float timer = 0f;
        var StartPosition = character.transform.position; //startna pozicija
protivnika
        if (SpawnAnimationCurve.length > 0)
        {
            //protivnik se krece sve dok ne dodje do krajnje pozicije
            while (timer < SpawnAnimationCurve.keys[SpawnAnimationCurve.length -
1].time)
            {
//Lerp omogucava postepeno pomeranje objekta od startne pozicije do destinacije
                character.transform.position = Vector3.Lerp(StartPosition,
destination.transform.position, SpawnAnimationCurve.Evaluate(timer));
                timer += Time.deltaTime;
                yield return new WaitForEndOfFrame(); //ceka se da sledeci frejm bude
spreman (inace bi se animacija izvrsila odjednom)
            }
        }
        else
```

```
        {
            character.transform.position = destination.transform.position;
            yield return null;
        }
    }

    // dugme se pojavljuje ukoliko se igrac nalazi u fazi napada
    void OnGUI()
    {
        if (phase == BattlePhase.PlayerAttack)
        {
            if (GUI.Button(new Rect(10, 10, 100, 50), "Run Away"))
            {
                NavigationMenager.NavigateTo("World");
            }
        }
    }
}
```

- Napraviti prazan objekat u Hierarchy delu i nazvati ga **BattleManager**. Prevući skriptu BattleManager.cs na njega. Unutar Inspector dela, Enemy Spawn Points->Size staviti na 5, pojaviće se 5 novih elemenata ispod, pa prevući sve Spawn Pointe iz Hierarchy dela na svaki element. U Enemy Prefabs->Size ukucati 1 pa ispod toga prevući prefab Goblin. Selektovati Spawn Animation Curve i tu označiti poslednju krivu. Trebalo bi da sve izgleda ovako:



- Dodati prefab **GM** u hijerarhiju koji će omogućiti prelaz izmeću scena.
- Dodati scenu **Battle** u Build Settings.

Čuvanje poslednje pozicije igrača na mapi

- Kada kliknemo na Run Away, vidimo da se igrač uvek nalazi na istom mestu na mapi. Da bismo to promenili, potrebno je da čuvamo poslednju poznatu poziciju igrača na mapi. Pravimo skriptu unutar Assets\Scripts koja se naziva **GameState.cs**.

GameState.cs:

```
using System.Collections.Generic;
using UnityEngine;

public static class GameState
{
    public static Player CurrentPlayer = ScriptableObject.CreateInstance<Player>();

    //recnik koji cuva scene i poslednju poziciju u sceni gde je bio igrac
    public static Dictionary<string, Vector3> LastScenePositions = new
Dictionary<string, Vector3>();

    public static Vector3 GetLastScenePosition(string sceneName)
    {
        //kada trazimo vrednost iz recnika, prvo se proverava da li vrednost
postoji...
        if (GameState.LastScenePositions.ContainsKey(sceneName))
        {
            var lastPos = GameState.LastScenePositions[sceneName];
            return lastPos;
        }
        //...inace se vraca default vrednost
        else
        {
            return Vector3.zero;
        }
    }

    public static void SetLastScenePosition(string sceneName, Vector3 position)
    {
        //kada dodajemo novu vrednost u recnik, prvo se proverava da li vrednost vec
postoji
        //u recniku. Ako postoji, onda se vrsi update...
        if (GameState.LastScenePositions.ContainsKey(sceneName))
        {
            GameState.LastScenePositions[sceneName] = position;
        }
        //...ako ne postoji, onda se jednostavno pravi novi unos u recnik
        else
        {
            GameState.LastScenePositions.Add(sceneName, position);
        }
    }
}
```

- Nakon ovoga, potrebno je ažurirati skriptu **MapMovement.cs** kako bi mapa učitavala poslednju lokaciju igrača ukoliko takva postoji i da čuva poslednju poziciju igrača kada izlazi sa scene.

MapMovement.cs:

```
    //trazi poslednju poziciju igraca za trenutnu scenu. Ukoliko takva postoji -
pomera igraca na nju
void Awake()
```

```
{
    var lastPosition = GameState.GetLastScenePosition(Application.loadedLevelName);
    if (lastPosition != Vector3.zero)
    {
        transform.position = lastPosition;
    }
}

// kada se napusta scena, cuva se poslednja poznata pozicija igraca
void OnDestroy()
{
    GameState.SetLastScenePosition(Application.loadedLevelName, transform.position);
}
```

Nasumično generisanje borbi

- Potrebno je odraditi da igrač, dok je na glavnoj mapi, može naleteti na nasumično generisanu borbu dok putuje po mapi. Ažuriraćemo **MapMovement.cs**:

MapMovement.cs:

```
// promenljive koje se ticu verovatnoce desavanja borbe na mapi
int EncounterChance = 30;
float EncounterDistance = 0;
```

ažuriramo metodu Update:

```
if (Input.GetMouseButtonUp(0))
{
... stari kod ...
            //odredjuje da li ce biti borbe. Ako ce biti, onda se odredjuje
            //distanca putanje koju ce preci igrac do trenutka borbe
            var EncounterProbability = Random.Range(1, 100);
            if (EncounterProbability < EncounterChance)
            {
                EncounterDistance = (Vector3.Distance(StartLocation, TargetLocation) /
100) * Random.Range(10, 100);
            }
            else
            {
                EncounterDistance = 0;
            }
}
else if (Input.touchCount > 0)
{

 ... stari kod ...

//copy-paste odozgo
   var EncounterProbability = Random.Range(1, 100);
   if (EncounterProbability < EncounterChance)
   {
       EncounterDistance = (Vector3.Distance(StartLocation, TargetLocation) / 100) *
Random.Range(10, 100);
   }
   else
   {
       EncounterDistance = 0;
   }
}
```

```
if (TargetLocation != Vector3.zero && TargetLocation != transform.position &&
TargetLocation != StartLocation)
        {
            // pokretanje igraca na mapi
            transform.position = Vector3.Lerp(StartLocation, TargetLocation,
MovementCurve.Evaluate(timer));

            timer += Time.deltaTime;
        }

        //ako je daljina postavljena, borba se mora odigrati.
        //Prema tome, kada je igrac putovao dovoljno daleko, ulazi se na ekran za
        //borbu
if (EncounterDistance > 0)
{
   if (Vector3.Distance(StartLocation, transform.position) > EncounterDistance)
   {
       TargetLocation = Vector3.zero;
       NavigationMenager.NavigateTo("Battle");
   }
}
```

- Dodati scenu **Battle** u Build Settings.

PRODAVNICA I INVENTAR

Kreiranje scene

- Pravimo novu scenu koju čuvamo pod nazivom **Shop** (unutar Scenes foldera)

- Prevući sliku **ShopScreen** iz foldera Sprites na glavni ekran scene i proveriti da se novi game objekat naziva ShopScreen (unutar Hierarchy prostora)

- Podeliti sliku **Weapon Icons 1** preko Sprite Editora na delove

- U Hierarchy delu, napraviti nove objekte tako da izgledaju kao na slici:

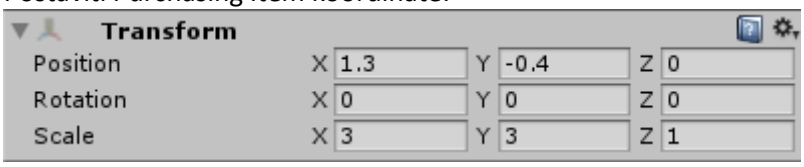
- Dodati komponentu **Sprite Renderer** na objekte BackButton i BuyButton. Prevući sličice BackButton i BuyButton u odgovarajući Sprite Renderer svakog objekta. Scena bi trebalo da izgleda ovako:

- Dodati komponentu **Box Colider 2D** sledećim objektima Slot1-Slot6, BuyButton. Za objekat BackButton dodati komponentu **Circle Colider 2D**. Za Slot1-Slot6 postaviti Scale za X i Y koordinatu na 0,6 i postaviti Position koordinate na sledeći način:

Slot1 X: -1.24, Y: 0.5
Slot2 X: -0.24, Y: 0.5
Slot3 X: -1.24, Y: -0.24
Slot4 X: -0.24, Y: -0.24
Slot5 X: -1.24, Y: -0.94
Slot6 X: -0.24, Y: -0.94

(za BuyButton i BackButton nije potrebno podešavati koordinate Box/Circle Colider-a jer će se one postaviti automatski kako bi popunile sličicu)

Postaviti Purchasing Item koordinate:



Inventar bi trebalo da izgleda ovako:

Svakom objektu dodati komponentu Sprite Renderer i postaviti Sorting Layer svakog objekta na **GUI**, osim za objekat ShopScreen, koga treba postaviti na **Background**.

Kreiranje predmeta za skladište (inventory)

- Pravimo novu C# skriptu (Assets/Scripts) pod nazivom **InventoryItem,** obrisati sav kod iz nje i napisati sledeće:

InventoryItem.cs:

```
using UnityEngine;
public class InventoryItem : ScriptableObject
{
	public Sprite Sprite;
	public Vector3 Scale;
	public string ItemName;
	public int Cost;
	public int Strength;
	public int Defense;
}
```

Ovim smo definisali pojedina svojstva naših predmeta, kao što su cena, snaga, odbrana...

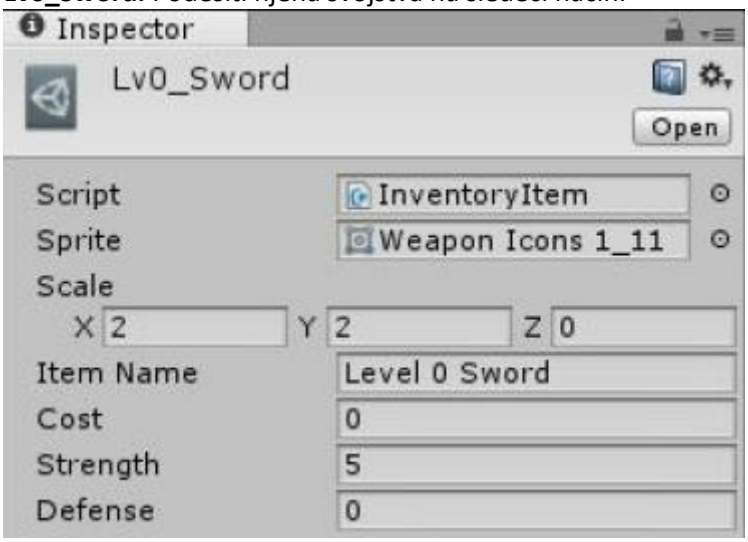
- Napraviti novu C# skriptu u (Assets/Scripts/Editor) pod nazivom **InventoryItemAssetCreator.**

InventoryItemAssetCreator.cs

```
using UnityEngine;
using UnityEditor;
public class InventoryItemAssetCreator : MonoBehaviour
{
   [MenuItem("Assets/Create/Inventory Item")]
    public static void CreateAsset()
    {
        //Kreira novu instancu ScribtibleObject
        InventoryItem InventoryItemAssetCreator =
                                    ScriptableObject.CreateInstance<InventoryItem>();

        //Kreira .asset fajl za naš novi objekat i čuva ga
        AssetDatabase.CreateAsset(InventoryItemAssetCreator, "Assets/newItem.asset");
        AssetDatabase.SaveAssets();

        //Prebacuje inspektor na naš novi objekat
        EditorUtility.FocusProjectWindow();
        Selection.activeObject = InventoryItemAssetCreator;
    }
}
```

Ovim smo dodali stavku u meni pod nazivom **Inventory Item**, sada možemo praviti objekte tog tipa.

- Sada je potrebno da napravimo neki Inventory Item. Napraviti folder **Inventory Items** unutar **Assets/Resources**. Napraviti novu InventoryItem klasu (RMB->Create->Inventory Item). Nazvati je **Lv0_Sword.** Podesiti njena svojstva na sledeći način:



Ponoviti postupak za još jedan item koji će se zvati Lv0_Axe (Weapon Icons 1_0).

Upravljanje prodavnicom

Pošto imamo izgled inventara i 2 predmeta, vreme je da ih spojimo. (S obzirom da će folderi ShopSlot i ShopManager zavisiti jedan od drugog, u toku kucanja koda pojavljivaće se greške, što je normalno).

- U folderu Assets/Scripts napraviti folder **Shop**, zatim u istom folderu napraviti C# skript **ShopManager**.

ShopManager.cs:

```
using UnityEngine;
public class ShopManager : MonoBehaviour
{
    public Sprite ShopOwnerSprite;
    public Vector3 ShopOwnerScale;
    public GameObject ShopOwnerLocation;
    public GameObject PurchasingSection;
    public SpriteRenderer PurchaseItemDisplay;
    public ShopSlot[] ItemSlots;
    public InventoryItem[] ShopItems;
    private static ShopSlot SelectedShopSlot;
    private int nextSlotIndex = 0;

    //kada igrac udje na ekran za kupovinu, prikazuje se vlasnik prodavnice i njegova
roba
    void Start()
    {
        var OwnerSpriteRenderer = ShopOwnerLocation.GetComponent<SpriteRenderer>();
        OwnerSpriteRenderer.sprite = ShopOwnerSprite;
        OwnerSpriteRenderer.transform.localScale = ShopOwnerScale;

        //pakujemo iteme u inventar
        if (ItemSlots.Length > 0 && ShopItems.Length > 0)
        {
            for (int i = 0; i < ShopItems.Length; i++)
            {
                if (nextSlotIndex > ItemSlots.Length) break;
                ItemSlots[nextSlotIndex].AddShopItem(ShopItems[i]);
                ItemSlots[nextSlotIndex].Manager = this;
                nextSlotIndex++;
            }
        }
    }

    //mogucnost selektovanja itema u prodavnici
    public void SetShopSelectedItem(ShopSlot slot)
    {
        SelectedShopSlot = slot;
        PurchaseItemDisplay.sprite = slot.Item.Sprite;
        PurchasingSection.SetActive(true);
    }

    //mogucnost brisanja itema iz prodavnice
    public void ClearSelectedItem()
    {
        Debug.Log("Clearing Shop Purchase area");
        SelectedShopSlot = null;
        PurchaseItemDisplay.sprite = null;
        PurchasingSection.SetActive(false);
    }
```

```
    //kupovina trenutno selektovanog itema
    public static void PurchaseSelectedItem()
    {
        SelectedShopSlot.PurchaseItem();
    }
}
```

- Pravimo novi C# skript **ShopSlot** u folderu Shop. Taj skript će definisati slotove u prodavnici koji pamte šta se drži na policama.

ShopSlot.cs:

```
using UnityEngine;

public class ShopSlot : MonoBehaviour
{
    public InventoryItem Item;
    public ShopManager Manager;

    //omogucava dodavanje itema na trenutni slot i prikazuje ga
    public void AddShopItem(InventoryItem item)
    {
        var spriteRenderer = GetComponent<SpriteRenderer>();
        spriteRenderer.sprite = item.Sprite;
        spriteRenderer.transform.localScale = item.Scale;
        Item = item;
    }

    //kontrolise kako je item kupljen
    public void PurchaseItem()
    {
        GameState.CurrentPlayer.Inventory.Add(Item);
        Item = null;
        var spriteRenderer = GetComponent<SpriteRenderer>();
        spriteRenderer.sprite = null;
        Manager.ClearSelectedItem();
    }

    //omogucava kliktanje po itemima na slotovima
    void OnMouseDown()
    {
        if (Item != null)
        {
            Manager.SetShopSelectedItem(this);
        }
    }
}
```

- Napraviti novi C# skript **BuyButton** i smestiti ga u folder Shop.

BuyButton.cs:

```
using UnityEngine;

public class BuyButton : MonoBehaviour
{
    void OnMouseDown()
    {
        ShopManager.PurchaseSelectedItem();
    }
}
```

- Napraviti novi C# skript **BackButton** i smestiti ga u folder Shop.

```
using UnityEngine;

public class BackButton : MonoBehaviour
{
    void OnMouseDown()
    {
        NavigationMenager.GoBack();
    }
}
```

- Moramo dodati mogućnost da se igrač, kada izađe iz prodavnice, nađe na mestu gde je u nju i ušao. Modifikovaćemo NavigationManager kako bismo zapamtili poslednje mesto na kojem je igrač bio. Otvoriti **NavigationManager** (Assets/Scripts):

- dodati linije

```
using UnityEngine;
using UnityEngine.SceneManagement;
```

- unutar klase dodati liniju `private static string PreviousLocation;`
- modifikovati NavigateTo metodu, dodati na vrh:

```
PreviousLocation = SceneManager.GetActiveScene().name; //cuva poslednju scenu na kojoj je bio igrac
```

- dodati novu funkciju:

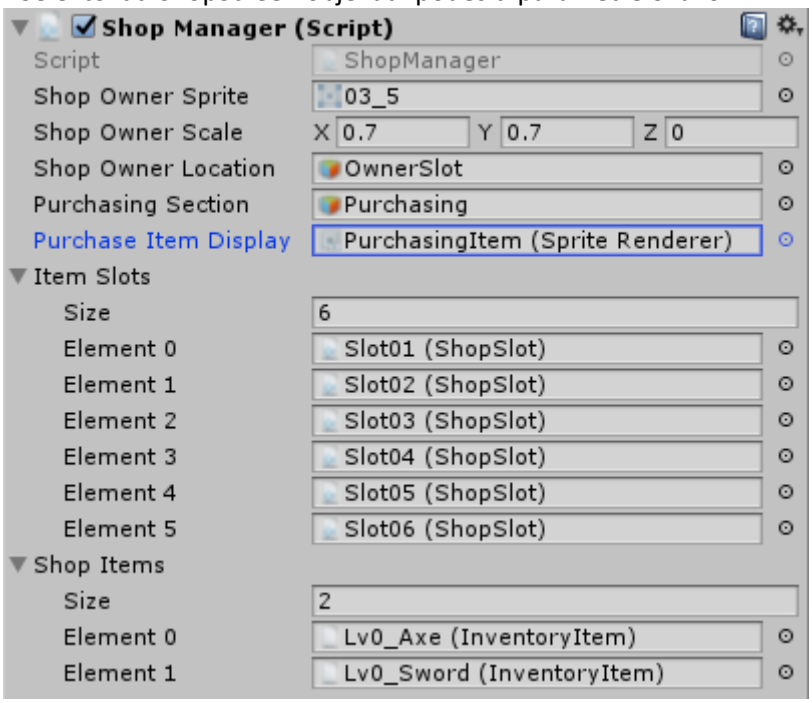
```
    //vracanje na prethodnu scenu
    public static void GoBack()
    {
        var backlocation = PreviousLocation; //cuva prethodnu lokaciju
        PreviousLocation  =  SceneManager.GetActiveScene().name;  //postavlja  trenutnu scenu na mesto prethodne
        Debug.Log(backlocation);
        FadeInOutManager.Instance.Fading(backlocation);
    }
```

- Otvoriti klasu **Player** unutar foldera (Assets/Scripts) i ažurirati je da izgleda ovako:

```
using System.Collections.Generic;

public class Player : Entity
{
    //definisemo one atribute koji su konkretni za igraca i
    //sto ga razlikuje od ostalih entiteta u igrici
    public List<InventoryItem> Inventory = new
                                System.Collections.Generic.List<InventoryItem>();
    public string[] Skills;
    public int Money;
}
```

- Prevući **ShopManager** skriptu (Assets/Shop) na **ShopScreen** objekat u Hierarchy delu.
- Prevući **BuyButton** skriptu (Assets/Shop) na **BuyButton** objekat u Hierarchy delu.
- Prevući **BackButton** skriptu (Assets/Shop) na **BackButton** objekat u Hierarchy delu.
- Prevući **ShopSlot** skriptu (Assets/Shop) na **Slot1-Slot6** objekte u Hierarchy delu.

- Selektovati ShopScreen objekat i podesiti parametre ovako:



(kliknuti na tačkicu pored svakog polja i izabrati odgovarajuće nazive)

- Igrač će ući u prodavnicu tako što će stati ispred nje i pritisnuti taster UP kako bi ušao. Pravimo novu C# skriptu unutar foldera Assets/Scripts pod nazivom **ShopEntry:**

ShopEntry.cs

```
using UnityEngine;
public class ShopEntry : MonoBehaviour
{
    bool canEnterShop; //kontrolise da li mozemo uci u prodavnicu ili ne

    //promena flega se vrsi preko ove funkcije
    void DialogVisible(bool visibility)
    {
        canEnterShop = visibility;
    }

    //detektuje da li je igrac ispred prodavnice
    void OnTriggerEnter2D(Collider2D col)
    {
        DialogVisible(true);
    }
    void OnTriggerExit2D(Collider2D col)
    {
        DialogVisible(false);
    }

    //detektuje da li je igrac pritisnuo UP za ulazak
    void Update()
    {
        if (canEnterShop && Input.GetKeyDown(KeyCode.UpArrow))
        {
            if (NavigationMenager.CanNavigate(this.tag))
            {
                NavigationMenager.NavigateTo(this.tag);
```

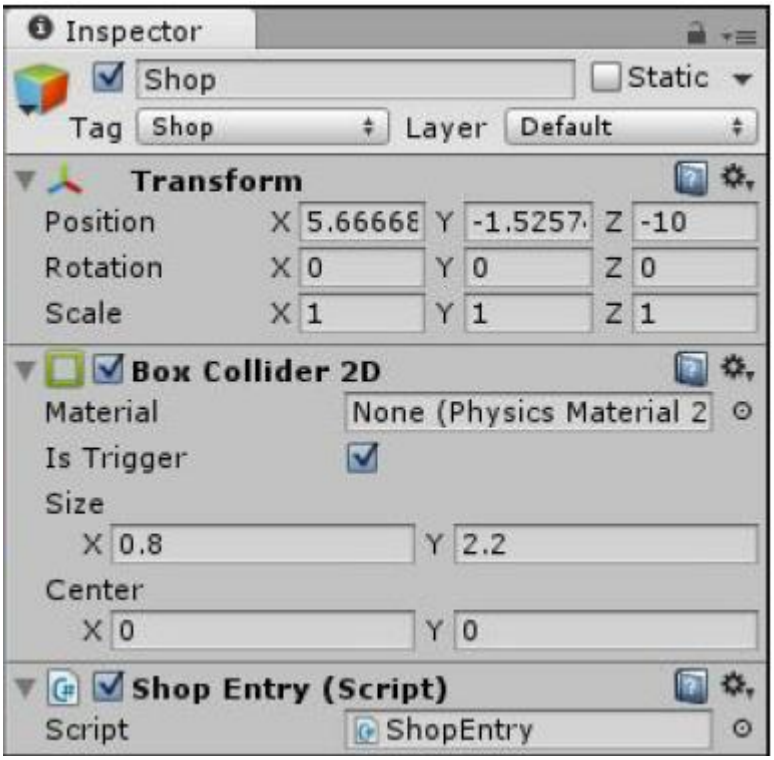
```
            }
        }
    }
    //igrac dobija obavestenje da moze uci u prodavnicu kada se nadje ispred nje
    void OnGUI()
    {
        if (canEnterShop)
        {
            //layout start
            GUI.BeginGroup(
            new Rect(
            Screen.width / 2 - 150,
            50,
            300,
            50));
            //the menu background box
            GUI.Box(new Rect(0, 0, 300, 250), "");
            //Dialog detail—updated to get better detail
            GUI.Label(
            new Rect(15, 10, 300, 68),
            "Do you want to Enter the Shop? (Press up)");
            //layout end
            GUI.EndGroup();
        }
    }
}
```

- Ići u **File->Build Settings** i tamo kliknuti na dugme **Add Open Scenes**, kako bi se pojavila nova stavka Scenes/Shop.

- Ažurirati skriptu **NavigationManager**. Dodati u rečnik (Dictionary) sledeći red:

```
{ "Shop", new Route {CanTravel = true}},
```

- Otvaramo sačuvanu scenu unutar Assets/Scenes pod nazivom Main. Pravimo novi objekat kao dete objektu WorldBounds, pod nazivom **Shop**. Prevući ShopEntry (Assets/Scripts) na novokreirani objekat Shop. Dodati **Box Collider 2D** komponentu objektu Shop. Štiklirati **Is Trigger**. Podesiti ostala svojstva da izgledaju ovako:



- Dodati novi tag Shop (klikne se na padajući meni pa stavka Add Tag unutar Inspector dela) i dodeliti tag Shop objektu Shop.

- Nakon ovoga, moguće je pokrenuti projekat i dovesti igrača ispred prodavnice, gde će se ispisati poruka.

Ubaciti GM u hijerarhiju za Shop.

Prikaz inventara igrača

- Napraviti novu skriptu unutar Scripts foldera pod nazivom **PlayerInventoryDisplay**.

PlayerInventoryDisplay.cs

```
using UnityEngine;

public class PlayerInventoryDisplay : MonoBehaviour
{
    bool displayInventory = false; //odredjuje da li je prozor inventara prikazan ili
ne
    Rect inventoryWindowRect;
    private Vector2 inventoryWindowSize = new Vector2(150, 150);
    Vector2 inventoryItemIconSize = new Vector2(130, 32);
    float offsetX = 6;
    float offsetY = 6;

    void Awake()
    {
```

```
        inventoryWindowRect = new Rect(
        Screen.width - inventoryWindowSize.x,
        Screen.height - 40 - inventoryWindowSize.y,
        inventoryWindowSize.x,
        inventoryWindowSize.y);
    }

    //prikaz dugmeta za pristup igracevom inventaru
    void OnGUI()
    {
        if (GUI.Button(new Rect(Screen.width - 40, Screen.height - 40, 40, 40), "INV"))
        {
            displayInventory = !displayInventory;
        }
        if (displayInventory)
        {
            inventoryWindowRect = GUI.Window(0, inventoryWindowRect,
DisplayInventoryWindow, "Inventory");
            inventoryWindowSize = new Vector2(inventoryWindowRect.width,
inventoryWindowRect.height);
        }
    }

    //prolazi kroz sve iteme u igracevom inventaru (ako ih ima) i prikazuje ih
    void DisplayInventoryWindow(int windowID)
    {
        var currentX = 0 + offsetX;
        var currentY = 18 + offsetY;
        foreach (var item in GameState.CurrentPlayer.Inventory)
        {
            Rect texcoords = item.Sprite.textureRect;
            texcoords.x /= item.Sprite.texture.width;
            texcoords.y /= item.Sprite.texture.height;
            texcoords.width /= item.Sprite.texture.width;
            texcoords.height /= item.Sprite.texture.height;
            GUI.DrawTextureWithTexCoords(new Rect(
            currentX,
            currentY,
            item.Sprite.textureRect.width,
            item.Sprite.textureRect.height),
            item.Sprite.texture,
            texcoords);
            currentX += inventoryItemIconSize.x;
            if (currentX + inventoryItemIconSize.x + offsetX > inventoryWindowSize.x)
            {
                currentX = offsetX;
                currentY += inventoryItemIconSize.y;
                if (currentY + inventoryItemIconSize.y + offsetY >
inventoryWindowSize.y)
                {
                    return;
                }
            }
        }
    }
}
```

- Prevući skriptu **PlayerInventoryDisplay.cs** na objekat Player u Hierarchy delu (objekat iz scene Main)

- Kada igrač kupi oružje potrebno je da omogućimo da to oružje i koristi kada je potrebno - otvoriti **Player** klasu unutar (Assets/Scripts). Dodati novu funkciju:

```
//promena statistike igraca u zavisnosti od novog itema
    public void AddinventoryItem(InventoryItem item)
    {
        this.Strength += item.Strength;
        this.Defense += item.Defense;
        Inventory.Add(item);
    }
```

- Ažurirati ShopSlot skriptu (Assets/Scripts/Shop) promeniti prvi red unutar metode PurchaseItem() sa ovom linijom:

```
GameState.CurrentPlayer.AddinventoryItem(Item);
```