Eyes of the Dragon Tutorials
Part 43
Equipment

I'm writing these tutorials for the MonoGame 3.8 framework using Visual Studio 2019. The tutorials will make more sense if they are read in order. You can find the list of tutorials on the Eyes of the Dragon page of my web blog. I will be making each version of the project available on GitHub here. It will be included on the page that links to the tutorials.

This tutorial will cover equipping items from inventory on the player character. It will be on the entity that is part of the player character. The reason being that both characters and mobs will require access to items, and they all contain entities. Let's get started.

First, I want to move a few things from the character class to the entity class. These items are related to the gear that the character has equipped. Update the character and entity classes to the following code.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using RpgLibrary.CharacterClasses;
using MGRpgLibrary.SpriteClasses;
using MGRpgLibrary.ItemClasses;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace MGRpgLibrary.CharacterClasses
{
    public class Character
    {
        #region Constant Region

        public const int SpeakingRadius = 32;
        public const int CollisionRadius = 8;

        #endregion

        #region Field Region

        protected Entity entity;
        protected AnimatedSprite sprite;

        // Armor fields
        protected GameItem head;
        protected GameItem body;
        protected GameItem hands;
        protected GameItem feet;

        // Weapon/Shield fields
        protected GameItem mainHand;
        protected GameItem offHand;
        protected int handsFree;
```

```
#endregion

#region Property Region

public Entity Entity
{
    get { return entity; }
}

public AnimatedSprite Sprite
{
    get { return sprite; }
}

#endregion

#region Money Region

private int _gold;

public int Gold { get => _gold; }

public void UpdateGold(int amount)
{
    _gold += amount;
}

#endregion

#region Constructor Region

public Character(Entity entity, AnimatedSprite sprite)
{
    this.entity = entity;
    this.sprite = sprite;
}

#endregion

#region Method Region
#endregion

#region Virtual Method region

public virtual void Update(GameTime gameTime)
{
    entity.Update(gameTime.ElapsedGameTime);
    sprite.Update(gameTime);
}

public virtual void Draw(GameTime gameTime, SpriteBatch spriteBatch)
{
    sprite.Draw(gameTime, spriteBatch);
}

public virtual bool Equip(GameItem gameItem)
{
    bool success = false;
    return success;
```

```
        }

        public virtual bool Unequip(GameItem gameItem)
        {
            bool success = false;
            return success;
        }

        #endregion
    }
}

using MGRpgLibrary.ItemClasses;
using RpgLibrary.EffectClasses;
using RpgLibrary.SkillClasses;
using RpgLibrary.SpellClasses;
using RpgLibrary.TalentClasses;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace RpgLibrary.CharacterClasses
{
    public enum EntityGender { Male, Female, NonBinary, Unknown }
    public enum EntityType { Character, NPC, Monster, Creature, Merchant }

    public class Entity
    {
        #region Vital Field and Property Region

        private string entityName;
        private string entityClass;
        private EntityType entityType;
        private EntityGender gender;

        public string EntityName
        {
            get { return entityName; }
            private set { entityName = value; }
        }

        public string EntityClass
        {
            get { return entityClass; }
            private set { entityClass = value; }
        }

        public EntityType EntityType
        {
            get { return entityType; }
            private set { entityType = value; }
        }

        public EntityGender Gender
        {
            get { return gender; }
            private set { gender = value; }
        }
```

```
    #endregion

    #region Resistance and Weakness Field and Property Region

    private readonly List<Resistance> resistances;

    public List<Resistance> Resistances
    {
        get { return resistances; }
    }

    private readonly List<Weakness> weaknesses;

    public List<Weakness> Weaknesses
    {
        get { return weaknesses; }
    }

    #endregion


#region Basic Attribute and Property Region

    private int strength;
    private int dexterity;
    private int cunning;
    private int willpower;
    private int magic;
    private int constitution;
    private int strengthModifier;
    private int dexterityModifier;
    private int cunningModifier;
    private int willpowerModifier;
    private int magicModifier;
    private int constitutionModifier;

    public int Strength
    {
        get { return strength + strengthModifier; }
        private set { strength = value; }
    }

    public int Dexterity
    {
        get { return dexterity + dexterityModifier; }
        private set { dexterity = value; }
    }

    public int Cunning
    {
        get { return cunning + cunningModifier; }
        private set { cunning = value; }
    }

    public int Willpower
    {
        get { return willpower + willpowerModifier; }
        private set { willpower = value; }
    }
```

```
public int Magic
{
    get { return magic + magicModifier; }
    private set { magic = value; }
}

public int Constitution
{
    get { return constitution + constitutionModifier; }
    private set { constitution = value; }
}

#endregion

#region Calculated Attribute Field and Property Region

private AttributePair health;
private AttributePair stamina;
private AttributePair mana;

public AttributePair Health
{
    get { return health; }
}

public AttributePair Stamina
{
    get { return stamina; }
}

public AttributePair Mana
{
    get { return mana; }
}

private int attack;
private int damage;
private int defense;

#endregion

#region Level Field and Property Region

private int level;
private long experience;

public int Level
{
    get { return level; }
    private set { level = value; }
}

public long Experience
{
    get { return experience; }
    private set { experience = value; }
}

#endregion
```

```
#region Skill Field and Property Region

readonly Dictionary<string, Skill> skills;
readonly List<Modifier> skillModifiers;

public Dictionary<string, Skill> Skills
{
    get { return skills; }
}

public List<Modifier> SkillModifiers
{
    get { return skillModifiers; }
}

#endregion

#region Spell Field and Property Region

readonly Dictionary<string, Spell> spells;
readonly List<Modifier> spellModifiers;

public Dictionary<string, Spell> Spells
{
    get { return spells; }
}

public List<Modifier> SpellModifiers
{
    get { return spellModifiers; }
}

#endregion

#region Talent Field and Property Region

readonly Dictionary<string, Talent> talents;
readonly List<Modifier> talentModifiers;

public Dictionary<string, Talent> Talents
{
    get { return talents; }
}

public List<Modifier> TalentModifiers
{
    get { return talentModifiers; }
}

public string HealthCalculation { get; set; }
public string StaminaCalculation { get; set; }
public string ManaCalculation { get; set; }

#endregion

#region Item Region

// Armor fields
protected GameItem head;
protected GameItem body;
```

```
protected GameItem hands;
protected GameItem feet;

// Weapon/Shield fields
protected GameItem mainHand;
protected GameItem offHand;
protected int handsFree;


// Armor properties
public GameItem Head
{
    get { return head; }
}

public GameItem Body
{
    get { return body; }
}

public GameItem Hands
{
    get { return hands; }
}

public GameItem Feet
{
    get { return feet; }
}

// Weapon/Shield properties

public GameItem MainHand
{
    get { return mainHand; }
}

public GameItem OffHand
{
    get { return offHand; }
}

public int HandsFree
{
    get { return handsFree; }
}

#endregion
#region Constructor Region

private Entity()
{
    Strength = 10;
    Dexterity = 10;
    Cunning = 10;
    Willpower = 10;
    Magic = 10;
    Constitution = 10;

    health = new AttributePair(0);
```

```
        stamina = new AttributePair(0);
        mana = new AttributePair(0);

        skills = new Dictionary<string, Skill>();
        spells = new Dictionary<string, Spell>();
        talents = new Dictionary<string, Talent>();

        skillModifiers = new List<Modifier>();
        spellModifiers = new List<Modifier>();
        talentModifiers = new List<Modifier>();
        resistances = new List<Resistance>();
        weaknesses = new List<Weakness>();
    }

    public Entity(
        string name,
        EntityData entityData,
        EntityGender gender,
        EntityType entityType) : this()
    {
        EntityName = name;
        Level = entityData.Level;
        EntityClass = entityData.EntityName;
        Gender = gender;
        EntityType = entityType;
        Strength = entityData.Strength;
        Dexterity = entityData.Dexterity;
        Cunning = entityData.Cunning;
        Willpower = entityData.Willpower;
        Magic = entityData.Magic;
        Constitution = entityData.Constitution;

        if (!string.IsNullOrEmpty(entityData.HealthFormula))
        {
            health = new AttributePair(CalculateAttribute(entityData.HealthFormula));
        }
        else
        {
            health = new AttributePair(entityData.MaximumHealth);
        }

        if (!string.IsNullOrEmpty(entityData.StaminaFormula))
        {
            stamina = new AttributePair(CalculateAttribute(entityData.StaminaFormula));
        }
        else
        {
            stamina = new AttributePair(entityData.MaximumStamina);
        }

        if (!string.IsNullOrEmpty(entityData.MagicFormula))
        {
            mana = new AttributePair(CalculateAttribute(entityData.MagicFormula));
        }
        else
        {
            mana = new AttributePair(entityData.MaximumMana);
        }
    }
```

```
        public int CalculateAttribute(string formula)
        {
            int value = 0;

            string[] parts = formula.Split('|');

            value = int.Parse(parts[0]);

            for (int i = 1; i < level; i++)
            {
                value += int.Parse(parts[2]);
            }

            return value;
        }

        #endregion

        #region Method Region

        public void Update(TimeSpan elapsedTime)
        {
            foreach (Modifier modifier in skillModifiers)
                modifier.Update(elapsedTime);

            foreach (Modifier modifier in spellModifiers)
                modifier.Update(elapsedTime);

            foreach (Modifier modifier in talentModifiers)
                modifier.Update(elapsedTime);
        }

        #endregion
    }
}
```

I did four things here:

1. I removed the code from the character class that has to do with equipment.
2. I moved that code to the Entity class.
3. I also added a using statement to bring the GameItem classes into the scope of this class.
4. I made it so that the Entity class is no longer sealed.

The next step will be to move things from the player's backpack to be an equipped item. To do that we are going to require some assets. Fortunately, OpenGameArt.org is always open and has just what we need! Following are a list of links to the content that I used in my version.

1. https://opengameart.org/content/simple-rpg-gui – it is a PhotoShop image but I converted it to PNG and extracted the portion that I used. You can download it from https://cynthiamcmahon.ca/blog/downloads/inventory-gui.png.
2. https://opengameart.org/content/dungeon-crawl-32x32-tiles-supplemental – a ton of useful graphics that we will make use off. Make sure and download all four files.

Once you have downloaded the graphics you need to add them to the project. Open the MonoGame Pipeline Tool. Browse to the files you downloaded and add the inventory-gui.png,

ProjectUtumno_full.png, and ProjectUtumno_supplemental.png files to the GUI folder. When prompted choose the option to copy them to the project folder.

Because of the size of the sheets holding the items I decided to have a texture manager to hold the sheets and update the GameItem class to use the texture manager. First, I will add the texture manager. Right click the MGRpgLibrary project, select Add and then Class. Name this new class TextureManager. Here is the code for that class.

```
using Microsoft.Xna.Framework.Graphics;
using System;
using System.Collections.Generic;
using System.Text;

namespace MGRpgLibrary
{
    public class TextureManager
    {
        private static Dictionary<string, Texture2D> _textures = new Dictionary<string,
Texture2D>();

        public static void AddTexture(string name, Texture2D texture)
        {
            if (!_textures.ContainsKey(name))
            {
                _textures.Add(name, texture);
            }
        }

        public static Texture2D GetTexture(string name)
        {
            if (_textures.ContainsKey(name))
            {
                return _textures[name];
            }

            return null;
        }
    }
}
```

A very basic class. It has a static field, _textures, that holds all of the textures to be managed. It has two static methods: AddTexture and GetTexture. AddTexture, as the name implies, adds a new texture to the texture manager. It takes a string that is the name of the texture in the manager and the texture as arguments. GetTexture, as the name implies, gets a texture from the texture manager. It takes as an argument the name of the texture. If the texture exists it is returned, otherwise null is returned.

Now, I need to update the GameItem class to use the texture manager instead of each item having its own texture. Change the GameItem class to the following code.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
```

```
using RpgLibrary.ItemClasses;

namespace MGRpgLibrary.ItemClasses
{
    public class GameItem
    {
        #region Field Region

        public Vector2 Position;
        private string image;
        private Rectangle? sourceRectangle;
        private readonly BaseItem baseItem;
        private Type type;

        #endregion

        #region Property Region

        public string Image
        {
            get { return image; }
        }

        public Rectangle? SourceRectangle
        {
            get { return sourceRectangle; }
            set { sourceRectangle = value; }
        }

        public BaseItem Item
        {
            get { return baseItem; }
        }

        public Type Type
        {
            get { return type; }
        }

        #endregion

        #region Constructor Region

        public GameItem(BaseItem item, string texture, Rectangle? source)
        {
            baseItem = item;
            image = texture;
            sourceRectangle = source;
            type = item.GetType();
        }

        #endregion

        #region Method Region

        public void Draw(SpriteBatch spriteBatch)
        {
            spriteBatch.Draw(TextureManager.GetTexture(image), Position, sourceRectangle,
Color.White);
        }
```

```
        #endregion

        #region Virtual Method region
        #endregion
    }
}
```

All I did was change the image field and Image property from a Texture2D to a string. Then, in the Draw method I use the GetTexture method of the TextureManager class to get the texture using the image field as the argument.

These changes broke the CharacterGeneratorScreen where we created a Merchant. That is because we add GameItems to the backpack and the constructor requires a string, not a Texture2D. Also, we need to load the textures for the items. Update the LoadWorld method of CharacterGeneratorScreen class to the following.

```
private void LoadWorld()
{
    RpgLibrary.WorldClasses.LevelData levelData =
        Game.Content.Load<RpgLibrary.WorldClasses.LevelData>(@"Game\Levels\Starting Level");

    RpgLibrary.WorldClasses.MapData mapData =
        Game.Content.Load<RpgLibrary.WorldClasses.MapData>(@"Game\Levels\Maps\" +
levelData.MapName);


    string[] fileNames = Directory.GetFiles(
        Path.Combine("Content/Game/Items", "Armor"),
        "*.xnb");

    foreach (string a in fileNames)
    {
        string path = "Game/Items/Armor/" + Path.GetFileNameWithoutExtension(a);

        ArmorData armorData = Game.Content.Load<ArmorData>(path);
        ItemManager.AddArmor(new Armor(armorData));
    }

    fileNames = Directory.GetFiles(
        Path.Combine("Content/Game/Items", "Shield"),
        "*.xnb");

    foreach (string a in fileNames)
    {
        string path = "Game/Items/Shield/" + Path.GetFileNameWithoutExtension(a);

        ShieldData shieldData = Game.Content.Load<ShieldData>(path);
        ItemManager.AddShield(new Shield(shieldData));
    }

    fileNames = Directory.GetFiles(
        Path.Combine("Content/Game/Items", "Weapon"),
        "*.xnb");

    foreach (string a in fileNames)
    {
```

```
        string path = "Game/Items/Weapon/" + Path.GetFileNameWithoutExtension(a);

        WeaponData weaponData = Game.Content.Load<WeaponData>(path);
        ItemManager.AddWeapon(new Weapon(weaponData));
    }

    CharacterLayerData charData =
        Game.Content.Load<CharacterLayerData>(@"Game\Levels\Chars\Starting Level");
    CharacterLayer characterLayer = new CharacterLayer();

    TileMap map = TileMap.FromMapData(mapData, Game.Content);

    foreach (var c in charData.Characters)
    {
        Character character;

        if (c.Value is NonPlayerCharacterData)
        {
            Entity entity = new Entity(c.Value.Name, c.Value.EntityData, c.Value.Gender,
EntityType.NPC);

            using (Stream stream = new FileStream(c.Value.TextureName, FileMode.Open,
FileAccess.Read))
            {
                Texture2D texture = Texture2D.FromStream(GraphicsDevice, stream);
                AnimatedSprite sprite = new AnimatedSprite(texture,
AnimationManager.Instance.Animations)
                {
                    Position = new Vector2(c.Key.X * Engine.TileWidth, c.Key.Y *
Engine.TileHeight)
                };

                character = new NonPlayerCharacter(entity, sprite);

                ((NonPlayerCharacter)character).SetConversation(
                    ((NonPlayerCharacterData)c.Value).CurrentConversation);
            }

            characterLayer.Characters.Add(c.Key, character);
        }
    }

    map.AddLayer(characterLayer);

    Level level = new Level(map);

    ChestData chestData = Game.Content.Load<ChestData>(@"Game\Chests\Plain Chest");

    Chest chest = new Chest(chestData);

    BaseSprite chestSprite = new BaseSprite(
        containers,
        new Rectangle(0, 0, 32, 32),
        new Point(10, 10));

    ItemSprite itemSprite = new ItemSprite(
        chest,
        chestSprite);

    level.Chests.Add(itemSprite);
```

```
    World world = new World(GameRef, GameRef.ScreenRectangle);

    world.Levels.Add(level);
    world.CurrentLevel = 0;

    AnimatedSprite s = new AnimatedSprite(
        GameRef.Content.Load<Texture2D>(@"SpriteSheets\Eliza"),
        AnimationManager.Instance.Animations);

    s.Position = new Vector2(0 * Engine.TileWidth, 5 * Engine.TileHeight);

    EntityData ed = new EntityData("Eliza", 1, 10, 10, 10, 10, 10, 10, "20|CON|12", "16|WIL|
16",
        "0|0|0");

    Entity e = new Entity("Eliza", ed, EntityGender.Female, EntityType.NPC);

    NonPlayerCharacter npc = new NonPlayerCharacter(e, s);

    npc.SetConversation("eliza1");
    //world.Levels[world.CurrentLevel].Characters.Add(npc);

    s = new AnimatedSprite(
        GameRef.Content.Load<Texture2D>(@"SpriteSheets\Eliza"),
        AnimationManager.Instance.Animations);
    s.Position = new Vector2(10 * Engine.TileWidth, 0);

    ed = new EntityData("Barbra", 2, 10, 10, 10, 10, 10, 10, "20|CON|12", "16|WIL|16", "0|0|
0");

    e = new Entity("Barbra", ed, EntityGender.Female, EntityType.Merchant);

    Merchant m = new Merchant(e, s);
    Texture2D items = Game.Content.Load<Texture2D>("ObjectSprites/roguelikeitems");
    m.Backpack.AddItem(new GameItem(ItemManager.GetWeapon("Long Sword"), "FullSheet", new
Rectangle(1696, 1408, 32, 32)));
    m.Backpack.AddItem(new GameItem(ItemManager.GetWeapon("Short Sword"), "FullSheet", new
Rectangle(800, 1504, 32, 32)));

    world.Levels[world.CurrentLevel].Characters.Add(m);
    ((CharacterLayer)world.Levels[world.CurrentLevel].Map.Layers.Find(x => x is
CharacterLayer)).Characters.Add(new Point(10, 0), m);
    GamePlayScreen.World = world;

    CreateConversation();

    //
((NonPlayerCharacter)world.Levels[world.CurrentLevel].Characters[0]).SetConversation("eliza1");
}
```

That was a lot of code but you know most of it from previous tutorials. There were really only the two changes: loading the textures and updating the call to the constructor of GameItem.

Now I am going to add a new state to handle inventory, and eventually equipping items. First, lets add a basic state that renders the inventory screen. Then, gradually, we will add in the functionality for the state. Right click the GameScreens folder in the EyesOfTheDragon project, select Add and then Class. Name this new class InventoryScreen. Here is the code for the initial class.

```
using MGRpgLibrary;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;
using System.Collections.Generic;
using System.Text;

namespace EyesOfTheDragon.GameScreens
{
    public class InventoryScreen : BaseGameState
    {
        private Texture2D _background;
        private Rectangle _destination;

        public InventoryScreen(Game game, GameStateManager manager) : base(game, manager)
        {
        }

        protected override void LoadContent()
        {
            base.LoadContent();
            _background = Game.Content.Load<Texture2D>(@"GUI/inventory-gui");
            _destination = new Rectangle(0, 0, Game1.ScreenWidth, Game1.ScreenHeight);
        }

        public override void Update(GameTime gameTime)
        {
            if (InputHandler.CheckMouseReleased(MouseButton.Right) ||
InputHandler.KeyReleased(Keys.Escape))
            {
                StateManager.PopState();
            }

            base.Update(gameTime);
        }

        public override void Draw(GameTime gameTime)
        {
            base.Draw(gameTime);

            GameRef.SpriteBatch.Begin();

            GameRef.SpriteBatch.Draw(_background, _destination, Color.White);

            GameRef.SpriteBatch.End();
        }
    }
}
```

A fairly simple class for now. There are fields for the destination rectangle that describes where the background will be drawn, which is of course a Texture2D. The LoadContent method loads the texture for the scene and creates the destination rectangle. The Update method checks to see if the Escape key has been pressed or the right mouse button has been clicked. If either are true it pops the state off the stack. The escape key will not work at the moment because it is still set in the Game1 class to exit the game. The Draw method just draws the texture using the destination rectangle.

The next thing that I'm going to do is add the state to the Game1 class and set the Update method to

not close when the Escape key is pressed. I will just show you the code for the entire class.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using MGRpgLibrary;
using EyesOfTheDragon.Components;
using EyesOfTheDragon.GameScreens;

namespace EyesOfTheDragon
{
    public class Game1 : Game
    {
        #region Frames Per Second Field Region

        private float fps;
        private float updateInterval = 1.0f;
        private float timeSinceLastUpdate = 0.0f;
        private float frameCount = 0;

        #endregion

        private GraphicsDeviceManager _graphics;
        private SpriteBatch _spriteBatch;
        private GameStateManager _gameStateManager;

        public const int ScreenWidth = 1280;
        public const int ScreenHeight = 720;

        public readonly Rectangle ScreenRectangle = new Rectangle(
            0,
            0,
            ScreenWidth,
            ScreenHeight);

        public SpriteBatch SpriteBatch { get { return _spriteBatch; } }
        public TitleScreen TitleScreen { get; private set; }
        public StartMenuScreen StartMenuScreen { get; private set; }
        public GamePlayScreen GamePlayScreen { get; private set; }
        public CharacterGeneratorScreen CharacterGeneratorScreen { get; private set; }
        public SkillScreen SkillScreen { get; private set; }
        public LoadGameScreen LoadGameScreen { get; private set; }
        public InventoryScreen InventoryScreen { get; private set; }
        public ConversationScreen ConversationScreen { get; private set; }
        public ShopState ShopScreen { get; }

        public Game1()
        {
            _graphics = new GraphicsDeviceManager(this);
            ScreenRectangle = new Rectangle(
                0,
                0,
                ScreenWidth,
                ScreenHeight);
            IsMouseVisible = true;

            Content.RootDirectory = "Content";

            Components.Add(new InputHandler(this));
```

```
            _gameStateManager = new GameStateManager(this);
            Components.Add(_gameStateManager);

            TitleScreen = new TitleScreen(this, _gameStateManager);
            StartMenuScreen = new StartMenuScreen(this, _gameStateManager);
            GamePlayScreen = new GamePlayScreen(this, _gameStateManager);
            CharacterGeneratorScreen = new CharacterGeneratorScreen(this, _gameStateManager);
            SkillScreen = new SkillScreen(this, _gameStateManager);
            LoadGameScreen = new LoadGameScreen(this, _gameStateManager);
            ConversationScreen = new ConversationScreen(this, _gameStateManager);
            ShopScreen = new ShopState(this, _gameStateManager);
            InventoryScreen = new InventoryScreen(this, _gameStateManager);

            _gameStateManager.ChangeState(TitleScreen);

            IsFixedTimeStep = false;
            _graphics.SynchronizeWithVerticalRetrace = false;
        }

        protected override void Initialize()
        {
            // TODO: Add your initialization logic here
            _graphics.PreferredBackBufferWidth = ScreenWidth;
            _graphics.PreferredBackBufferHeight = ScreenHeight;
            _graphics.ApplyChanges();

            base.Initialize();
        }

        protected override void LoadContent()
        {
            _spriteBatch = new SpriteBatch(GraphicsDevice);

            DataManager.ReadEntityData(Content);
            DataManager.ReadArmorData(Content);
            DataManager.ReadShieldData(Content);
            DataManager.ReadWeaponData(Content);
            DataManager.ReadChestData(Content);
            DataManager.ReadKeyData(Content);
            DataManager.ReadSkillData(Content);

            FontManager.AddFont("testfont", Content.Load<SpriteFont>("Fonts/scenefont"));
        }

        protected override void Update(GameTime gameTime)
        {
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed ||
Keyboard.GetState().IsKeyDown(Keys.F10))
                Exit();

            // TODO: Add your update logic here

            base.Update(gameTime);
        }

        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);

            base.Draw(gameTime);
```

```
            float elapsed = (float)gameTime.ElapsedGameTime.TotalSeconds;
            frameCount++;
            timeSinceLastUpdate += elapsed;

            if (timeSinceLastUpdate > updateInterval)
            {
                fps = frameCount / timeSinceLastUpdate;

                System.Diagnostics.Debug.WriteLine("FPS: " + fps.ToString());
                this.Window.Title = "FPS: " + fps.ToString();

                frameCount = 0;
                timeSinceLastUpdate -= updateInterval;
            }
        }
    }
}
```

Nothing here that you haven't seen before. There is a property for the InventoryScreen and it is created in the constructor. In the Update method I replaced the Escape key with the F10 key to check it the game should be closed.

The next step is to display the screen when a condition is true. I set it to the I key being pressed. In the GamePlayScreen set the Update method to the following.

```
public override void Update(GameTime gameTime)
{
    world.Update(gameTime);
    player.Update(gameTime);
    player.Camera.LockToSprite(player.Sprite);

    if (InputHandler.KeyReleased(Keys.Space) ||
        InputHandler.ButtonReleased(Buttons.A, PlayerIndex.One))
    {
        foreach (ILayer layer in World.Levels[World.CurrentLevel].Map.Layers)
        {
            if (layer is CharacterLayer)
            {
                foreach (Character c in ((CharacterLayer)layer).Characters.Values)
                {
                    float distance = Vector2.Distance(
                        player.Sprite.Center,
                        c.Sprite.Center);

                    if (distance < Character.SpeakingRadius && c is NonPlayerCharacter)
                    {
                        NonPlayerCharacter npc = (NonPlayerCharacter)c;

                        if (npc.HasConversation)
                        {
                            StateManager.PushState(GameRef.ConversationScreen);

                            GameRef.ConversationScreen.SetConversation(
                                player,
                                npc,
                                npc.CurrentConversation);
```

```
                        GameRef.ConversationScreen.StartConversation();
                    }
                }
                else if (distance < Character.SpeakingRadius && c is Merchant)
                {
                    StateManager.PushState(GameRef.ShopScreen);
                    GameRef.ShopScreen.SetMerchant(c as Merchant);
                }
            }
        }
    }

    if (InputHandler.KeyReleased(Keys.I))
    {
        StateManager.PushState(GameRef.InventoryScreen);
    }

    base.Update(gameTime);
}
```

All the new code does is check if the I key has been pressed. If it has, the InventoryScreen is pushed onto the stack.

Next, I am going to add an overload of the Draw method of the GameItem class. This overload will take a Rectangle that is the destination of the game item on the screen. Add this overload of the Draw method of the GameItem class.

```
public void Draw(SpriteBatch spriteBatch, Rectangle destination)
{
    spriteBatch.Draw(TextureManager.GetTexture(Image), destination, sourceRectangle,
        Color.White);
}
```

Now I am going to fix a bug in the ShopState. In the Buy action in the Update method I call GetItem to get the item and then PeekItem to get the price. The order should be reversed. You should peek and then get. Change the Update method of the ShopState to the following.

```
public override void Update(GameTime gameTime)
{
    base.Update(gameTime);

    scene.Update(gameTime, playerIndexInControl);

    switch (State)
    {
        case ShopStateType.Buy:
            if (isFirst)
            {
                isFirst = false;
                break;
            }

            if (InputHandler.KeyReleased(Keys.Down) ||
                InputHandler.KeyReleased(Keys.S))
            {
```

```
            selected++;
            if (selected >= merchant.Backpack.Items.Count)
            {
                selected = 0;
            }
        }

        if (InputHandler.KeyReleased(Keys.Up) ||
            InputHandler.KeyReleased(Keys.W))
        {
            selected--;
            if (selected < 0)
            {
                selected = merchant.Backpack.Items.Count - 1;
            }
        }


        if (InputHandler.KeyReleased(Keys.Space) ||
            InputHandler.KeyReleased(Keys.Enter) ||
            (InputHandler.CheckMouseReleased(MouseButton.Left) && mouseOver))
        {
            if (selected != -1 &&
                GamePlayScreen.Player.Gold >=
                merchant.Backpack.PeekItem(
                    merchant.Backpack.Items[selected].Item.Name).Price)
            {
                if (selected >= merchant.Backpack.Items.Count)
                {
                    return;
                }

                GamePlayScreen.Player.Gold -=
                    merchant.Backpack.PeekItem(
                        merchant.Backpack.Items[selected].Item.Name).Price;
                GamePlayScreen.Player.Backpack.AddItem(
                    merchant.Backpack.GetItem(
                        merchant.Backpack.Items[selected].Item.Name));
            }
        }
        break;
    case ShopStateType.Sell:
        if (isFirst)
        {
            isFirst = false;
            break;
        }
        if (InputHandler.KeyReleased(Keys.Down) ||
            InputHandler.KeyReleased(Keys.S))
        {
            selected++;

            if (selected >= GamePlayScreen.Player.Backpack.Items.Count)
            {
                selected = 0;
            }
        }

        if (InputHandler.KeyReleased(Keys.Up) ||
            InputHandler.KeyReleased(Keys.W))
```

```
            {
                selected--;
                if (selected < 0)
                {
                    selected = GamePlayScreen.Player.Backpack.Items.Count - 1;
                }
            }


            if ((InputHandler.KeyReleased(Keys.Space) ||
                InputHandler.KeyReleased(Keys.Enter) ||
                (InputHandler.CheckMouseReleased(MouseButton.Left) && mouseOver)))
            {
                if (selected >= 0)
                {
                    GameItem item = GamePlayScreen.Player.Backpack.GetItem(
                        GamePlayScreen.Player.Backpack.Items[selected].Item.Name);
                    GamePlayScreen.Player.Gold += item.Item.Price * 3 / 4;
                }
            }
            break;
        case ShopStateType.Talk:
            if (InputHandler.KeyReleased(Keys.Space) ||
                InputHandler.KeyReleased(Keys.Enter))
            {
                if (scene.SelectedIndex == 0)
                {
                    isFirst = true;
                    State = ShopStateType.Buy;
                    //selected = -1;
                    return;
                }

                if (scene.SelectedIndex == 1)
                {
                    isFirst = true;
                    State = ShopStateType.Sell;
                    //selected = -1;
                    return;
                }

                if (scene.SelectedIndex == 2 && State == ShopStateType.Talk)
                {
                    StateManager.PopState();
                }
            }
            break;
    }
    if (InputHandler.CheckMouseReleased(MouseButton.Right) ||
InputHandler.KeyReleased(Keys.Escape))
    {
        switch (State)
        {
            case ShopStateType.Buy:
            case ShopStateType.Sell:
                State = ShopStateType.Talk;
                break;
        }
    }
}
```

Next up, we need to load the texture for items. I did that in the Game1 class in the LoadContent method, of course. Update the LoadContent method of the Game1 class to the following.

```
protected override void LoadContent()
{
    _spriteBatch = new SpriteBatch(GraphicsDevice);

    DataManager.ReadEntityData(Content);
    DataManager.ReadArmorData(Content);
    DataManager.ReadShieldData(Content);
    DataManager.ReadWeaponData(Content);
    DataManager.ReadChestData(Content);
    DataManager.ReadKeyData(Content);
    DataManager.ReadSkillData(Content);

    FontManager.AddFont("testfont", Content.Load<SpriteFont>("Fonts/scenefont"));
    TextureManager.AddTexture("FullSheet", Content.Load<Texture2D>("GUI/ProjectUtumno_full"));
}
```

The last thing that I am going to tackle in this tutorial is rendering items if the player has any in their backpack. I will tackle equipping in a second part to this tutorial. First, we need the destination to draw the items on the inventory screen. Add the following field to the InventoryScreen and this override of the Initialize method.

```
private List<Rectangle> _backpackDestinations = new List<Rectangle>();

public override void Initialize()
{
    int x = 622, y = 96;

    for (int i = 0; i < 30; i++)
    {
        _backpackDestinations.Add(new Rectangle(x, y, 106, 78));

        x += 123;

        if (x >= 1220)
        {
            x = 622;
            y += 78;
        }
    }

    base.Initialize();
}
```

What the Initialize method does is set the X and Y coordinate of the first box for the backpack. Since there are 30 boxes it loops thirty times. It creates a rectangle for that box and adds it to the list of destination rectangles for the backpack. It then increments the X coordinate by the spacing between items. I calculated that using an image editor. If the X coordinate is greater than the width of the row, it resets the X coordinate back to the first column and increments the Y coordinate to the next row.

The last thing to do is draw the items in the backpack. Update the Draw method of the InventoryScreen class to the following.

```
public override void Draw(GameTime gameTime)
{
    base.Draw(gameTime);

    GameRef.SpriteBatch.Begin();

    GameRef.SpriteBatch.Draw(_background, _destination, Color.White);

    int c = 0;

    foreach (GameItem item in GamePlayScreen.Player.Backpack.Items)
    {
        item.Draw(GameRef.SpriteBatch, _backpackDestinations[c]);
        c++;
        if (c >= _backpackDestinations.Count)
        {
            break;
        }
    }

    GameRef.SpriteBatch.End();
}
```

After drawing the background, I set a local variable to 0. I then loop over the items in the Player's backpack. I draw the item using the new overload of the Draw method passing in the destination rectangle that corresponds to the index. I then increment the index. If the end of the list of destinations is reached I break out of the loop.

Though not as long as I had originally thought, I'm going to wrap up the tutorial here because I don't want to start something new at this point. Visit my blog, https://cynthiamcmahon.ca/blog/, for the latest news on my tutorials and other goodness.

Good luck in your Game Programming Adventures!

Cynthia