RPG Board Project

Andrew Reid East
Student ID: 16280042
20/11/2017
CT3111

Introduction

This game is a visual aid to support a tabletop role playing game that has a grid-based map, character tokens that move on the grid squares, and have attack actions against other tokens. The game master can predefine scenes, such as the Undead Ambush and Kobold Raid scene that have been already made, and then the players and game master will take turns controlling their characters or monsters.

Controls

Control is mainly mouse based: the user will click on the grid (represented as flat, translucent rectangular prisms that fall onto the terrain topography) to move or click on other tokens to attack. There is a button to end a creature's turn (if they don't or can't use up all their actions), which can also be triggered by the spacebar. WASD controls the camera's location, and the mouse tilts and pans the camera slightly. The stats for the currently active creature are displayed on screen. Attacks are represented by an abstract weapon, and a successful hit depends on a virtual dice roll (done in the background).

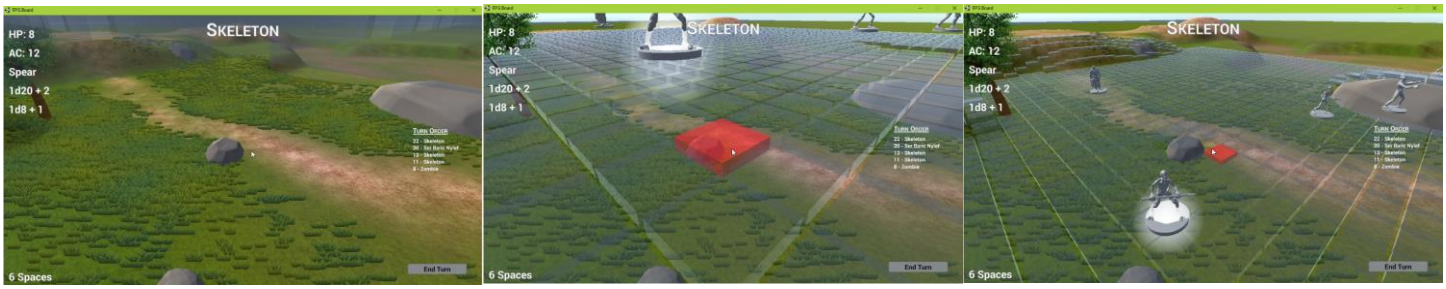
Instructions

The user clicks pre-built scene from the main menu. Turn order is determined by a random roll, and tokens are placed in predefined locations. The active character is displayed at the top of the screen, and their player clicks on game elements to move or attack. The player uses up their available actions or clicks the end turn button, and then play passes to the next character. Turns continue round-robin according to the turn order. The game is over when all the characters for either the human or monster side are killed, and the main menu is displayed again.

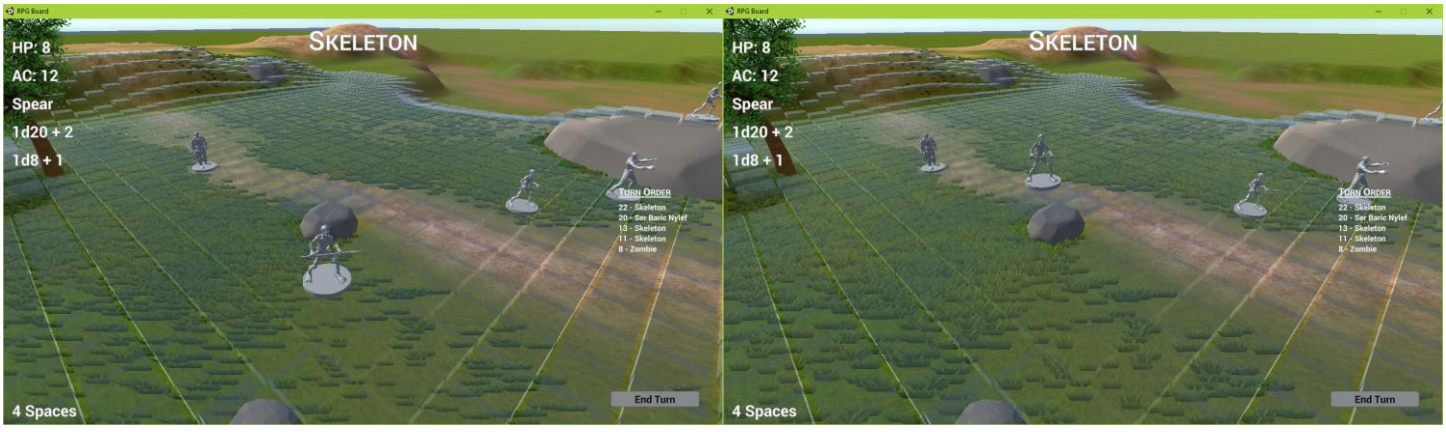
To build a scene, create tokens in the Unity editor and attach scripts to facilitate their walking/attacking behaviour as well as a script to define their in-game stats. In the inspector pane for the GameManager object, add those prefab tokens to either the player or monster list. In the code editor, modify GameManager.cs' Start method to build an actor data structure containing a reference to the prefabs created, the square to drop the token onto, their colour, and which team they are on. Link up several actors to build a scene.

Screenshots

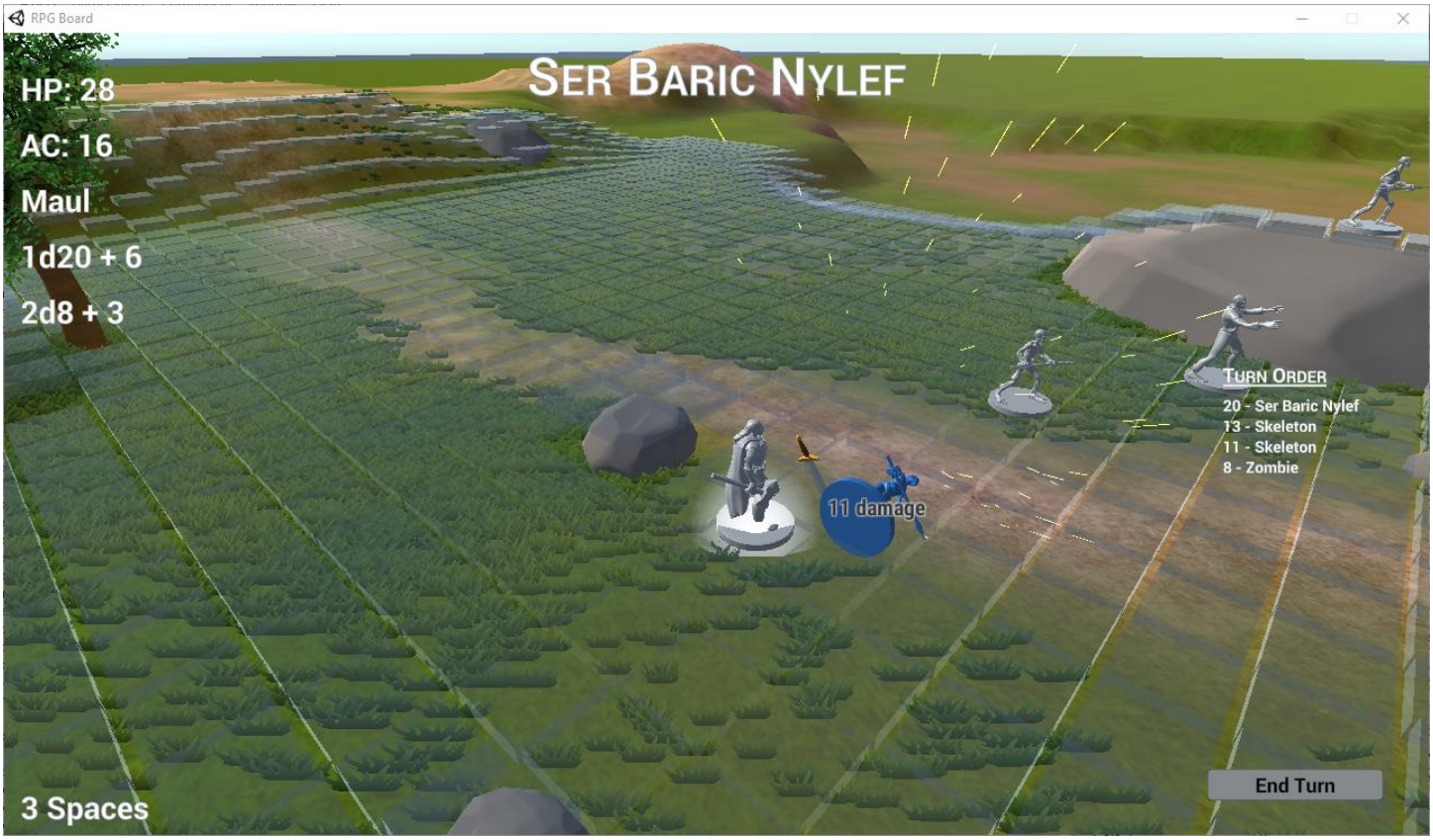
Main Menu



Using physics engine to drop squares upon terrain, mapping the topography



Token "hopping" around the board, as if it was being picked up by a human



Attack animation, after a successful dice role. Shows the floating sword being animated and particle effects. Also demonstrated: blue highlighting on the target token of the attack.

Description of Work Done

Tokens

I imported models for the RPG tokens, which are models sculpted by an artist to be suitable for 3d-printing that I converted to Unity-compatible formats. These models are put inside a container to amortize their size and facing, and given a simple box collider and rigid body, as well five independent scripts to:

1. Walk the token along the board, which given a queue/path of squares to hop between. Uses slerp() to linearly interpolate an arc between two squares and Update to space the animation out smoothly.
2. Animate an attack, which takes a prefab sword (but could be extended with other prefab attacks)
3. Highlight itself on mouse over
4. Send a message to the GameManager when clicked
5. Hold the RPG stats for the character, namely hit points, armour, attack, speed in # of squares/turn, and name

Game Board

The game board is made up of 3d cube primitives. They have a box collider has a simple physics material I created to be slightly bouncy, giving a desired effect when tokens are dropped onto them. Their rigid body is very heavy, so as to stay put when tokens are dropped atop them, and is frozen in rotation and all directions except Y. They have material I made, with no texture but has transparent rendering mode, tweaked to be translucent enough to show the terrain underneath but to sill have the grid be discernible. Each square has 2 scripts, one to send messages to the GameManager when clicked, and one to highlight itself on mouse over—the colour of which is controlled whose turn it is as a static public property of all the squares.

Terrain

I created a game board using Unity's terrain tool, painting on textures and adding rocks, grass, and a lonely (but happy) tree (assets and textures from source). Terrain was designed to give a mainly flat playing surface, but have some areas of interest to navigate through for potential strategy (depending on how creative the players are). Since the game board squares drop down, it maps the topology of the terrain at run-time, giving the potential for complex scenes to be modified very easily with the Unity terrain tools.

Weapon Animation

A weapon, which is attached to a token via the attacker-script, is animated using Unity's animation tool (model is imported from an external source). The animation storyboard has events attached, which call functions in an attached script. About half-way through the animation, it calls a script which finds any ParticleSystems in the object's children and plays them, showing a "hit" effect. At the end of the animation, the script is called to destroy the whole weapon. The ParticleSystem is simple, emitting a spray of sparks (particle texture taken from outside source, but I put together the particle effect in Unity).

Camera

The camera is controlled by the keyboard, and can be tilted slightly as the mouse moves around the screen (camera look script is from an outside source). It is not complex and affords little control, since the game is meant to be turn-based.

Menus and UI

There is a simple menu and in-game UI. Changes to the UI are controlled from the GameManager via public properties. The GameManager also manages which menu is display and what mouse input is currently allowed via a simple finite state machine.

PopupText

The popup text is done according to a tutorial, cited below. It is generic for any message to be shown during play.

GameManager

The GameManager script handles generating the board from prefabs and generating tokens from prefabs based upon an array of tokens/locations called Actors. It builds a scene in a generic way based upon these arrays. It handles taking events from menu buttons and starting the appropriate scene, and changing the state back to the menu when someone wins a scene. It keeps track of which character is currently active, asks a Pathfinder script to run A* (taken from my Java code written last year for a CT255 class assignment) to deduce a path between squares, and determines if attacks are possible based on a character's stats. It applies attack and damage based on random virtual dice rolls, and determines if a character has been killed. The purpose of its Update method is to respond to keyboard input.

Pathfind

A static class to perform pathfinding on the grid. Uses public references to GameManager's data grid structures to determine the size of the grid and what spaces are passable. Has a second class to simply calculate distance (which unfortunately just does the full pathfinding routine and reads the length of the stack of path steps, rather than finding a shortest path and then simply returning that number). This code is work I first did in Java last year.

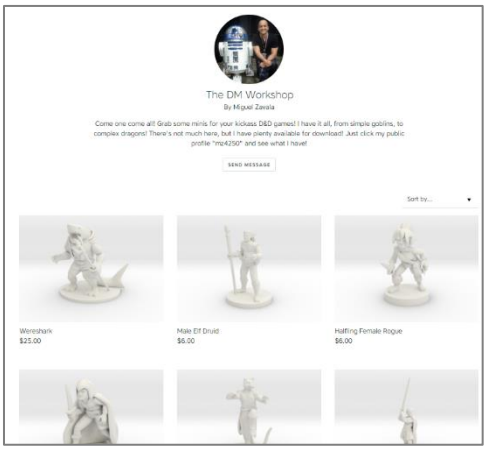
Attribution and Assets

Tabletop token 3d-printable models

by Miguel Zavala on Shapeways

https://www.shapeways.com/shops/dmworkshop

No explicit license listed online, but author's description encouraging sharing and downloading: https://www.shapeways.com/designer/mz4250



Online 3D Converter (STL → OBJ) by Alexander Gessler

http://www.greentoken.de/onlineconv/

Textures, Rock/Tree/Grass from Nature Starter Kit 1 & 2

by Shapes

https://www.assetstore.unity3d.com/en/#!/content/49962

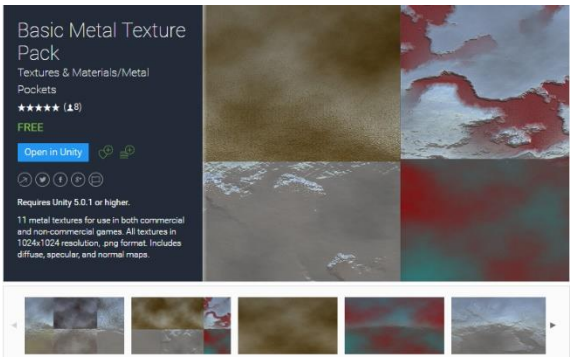
Free license

Pewter metal material from Basic Metal Texture Pack

by Pockets

https://www.assetstore.unity3d.com/en/#!/content/37402

Free license

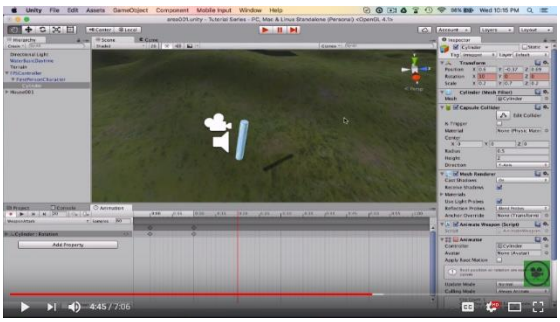


Weapon animations inspired heavily from walkthrough:

How to Make Weapon Animation in Unity 5

by MDL Tutorials

https://www.youtube.com/watch?v=tL3qq-UBbow

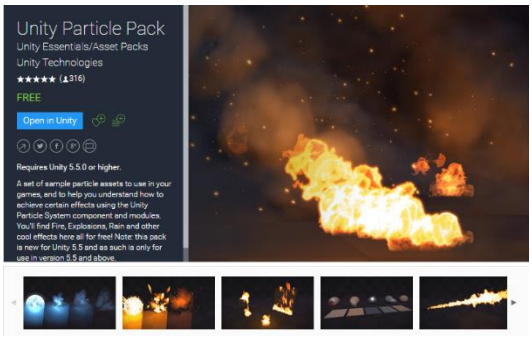


Unity Particle Pack

By Unity Technologies

https://www.assetstore.unity3d.com/#!/content/73777

Free License



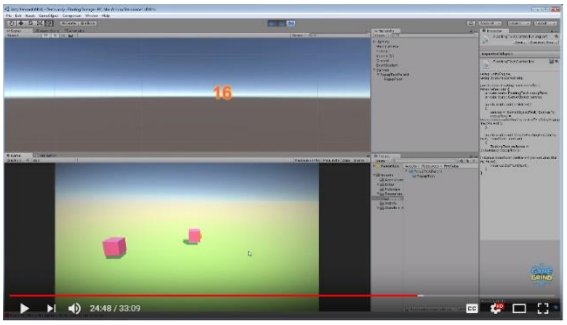
Sparks ParticleSystem tutorial by Unity:

https://unity3d.com/learn/tutorials/topics/graphics/creating-sparks-particle-trails

Popup Text Tutorial

by GameGrind

https://www.youtube.com/watch?v=fbUOG7f3jq8

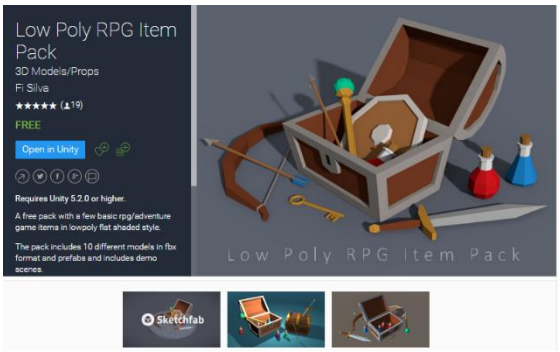


Low Poly RPG Item Pack

by Fi Silva

https://www.assetstore.unity3d.com/en/#!/content/76088

Free License



Text Mesh Pro

by Unity

https://www.assetstore.unity3d.com/en/#!/content/84126

Free License

Mouse look script: https://answers.unity.com/answers/1135844/view.html

Mouse hover highlight colour script: https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnMouseOver.html

LookAt without tilting method: https://answers.unity.com/answers/250578/view.html

Moving an object in an arc using slerp: https://answers.unity.com/questions/11184/moving-player-in-an-arc-from-startpoint-to-endpoin.html

Pathfinding using A* is taken *directly* from the Java code I wrote as an assignment for CT255 at NUIG in 2016

Finally, many inspirations were taken from Dr Sam Redfern's labs for CT3111!

CODE LISTING

Code also available at on GitHub (private repo, please email me if desired so I can share it)
https://github.com/reideast/Unity3dRPGBoard

GameManager.cs

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

public class GameManager : MonoBehaviour {
    // Public GameObjects to be assigned in editor
    public GameObject OneByOnePrefab;
    public Camera Camera;
    public List<GameObject> MonsterPrefabsList;
    public List<GameObject> PlayerPrefabList;
    public GameObject MenuCanvas, InGameCanvas, PlayersWinMessage, MonstersWinMessage;
    public TMP_Text TextCurrentActor, TextHP, TextAC, TextAtkName, TextAtkRoll, TextDmgRoll,
TextSpeedLeft, TextTurnTracker;

    [HideInInspector] public static GameManager instance;

    // Data structures to support running the game
    private List<Actor> actors;
    private int currentActorTurn;
    public Turn currentTurnStats;
    private int playerCount, monsterCount;
    [HideInInspector] public static STATES state = STATES.MENU;
    public enum STATES {
        MENU,
        AWAITING_INPUT,
        ANIMATING_ACTION
    };

    // Predefined Scenarios
    private SceneActor[] undeadScene;
    private SceneActor[] koboldScene;

    // The game board, with useful properties for several scripts
    // each Space object contains a reference to a OneByOne (GameObject). Use these to find actual
world unit coordinates of each game space
    [HideInInspector] public Space[,] spaces;
    public GameObject SpacesHolder; // an empty GameObject to hold all the spaces. Simply to reduce
clutter...doesn't improve performance, I think

    // Properties of the spaces
    public int RowsX = 60, ColsZ = 60;
    private const float DropFromHeight = 10f;
    private const float Margin = 0.05f;
    private const float SpaceHeight = 0.2f;
    public Vector3 SPACE_HEIGHT_MOD;
    private const float cameraSpeed = 4;


    void Start() {
        GameManager.instance = this;
        SPACE_HEIGHT_MOD = new Vector3(0f, SpaceHeight, 0f);
        PopupTextController.Initialize();

        // Build the predefined scenarios
        undeadScene = new SceneActor[] {
            new SceneActor(true, 0, 25, 17, new Color(0, 0.47f, 1f, 0.58f)), // Paladin
            new SceneActor(false, 0, 28, 27, new Color(1f, 0, 0, 0.58f)), // Skeleton
            new SceneActor(false, 0, 13, 30, new Color(1f, 0.5f, 0, 0.58f)),
```

```
            new SceneActor(false, 0, 20, 27, new Color(1f, 0.75f, 0, 0.58f)),
            new SceneActor(false, 1, 17, 29, new Color(0.5f, 0.75f, 0.5f, 0.58f)) // Zombie
        };
        koboldScene = new SceneActor[] {
            new SceneActor(true, 1, 28, 27, new Color(0, 0.8f, 0.5f, 0.58f)), // Heavy Weapon
Fighter
            new SceneActor(true, 2, 26, 29, new Color(0, 0.47f, 0.5f, 0.58f)), // Bow Ranger
            new SceneActor(true, 3, 27, 26, new Color(0, 0.0f, 0.5f, 0.58f)), // Rogue
            new SceneActor(false, 2, 13, 30, new Color(1f, 0.7f, 0.8f, 0.58f)), // Basic kobold
            new SceneActor(false, 2, 15, 24, new Color(1f, 0.3f, 0.8f, 0.58f)),
            new SceneActor(false, 2, 17, 20, new Color(1f, 0.5f, 0.0f, 0.58f)),
            new SceneActor(false, 2, 18, 22, new Color(1f, 0.1f, 0.4f, 0.58f)),
            new SceneActor(false, 3, 12, 28, new Color(1f, 0.75f, 0.2f, 0.58f)), // Kobold rogue
            new SceneActor(false, 4, 18, 25, new Color(1f, 0, 0, 0.58f)), // Kobold sorcerer
        };

        // Generate game board made of one-by-one squares
        spaces = new Space[RowsX, ColsZ];
        GenerateSquares();
    }

    public void SetState(STATES newSate) {
        state = newSate;
        if (newSate == STATES.AWAITING_INPUT) {
            MouseHoverHighlight.isEffectActive = true;
            ((Behaviour) actors[currentActorTurn].tokenRef.GetComponent("Halo")).enabled = true;
        } else if (newSate == STATES.ANIMATING_ACTION) {
            MouseHoverHighlight.isEffectActive = false;
            ((Behaviour) actors[currentActorTurn].tokenRef.GetComponent("Halo")).enabled = false;
        } else if (newSate == STATES.MENU) {
            MouseHoverHighlight.isEffectActive = false;
            ((Behaviour) actors[currentActorTurn].tokenRef.GetComponent("Halo")).enabled = false;
            InGameCanvas.SetActive(false);
            MenuCanvas.SetActive(true);
        }
    }

    private static int RollDice(int numDice, int diceMagnitude, int mod) {
        int diceTotal = mod;
        for (int i = 0; i < numDice; ++i) {
            diceTotal += Random.Range(1, diceMagnitude);
        }
        return diceTotal;
    }

    public void OnClickStartUndead() {
        // Reset the scene and place the new scene's tokens
        ResetBuildAndStartScene(undeadScene);
    }

    public void OnClickStartKobold() {
        ResetBuildAndStartScene(koboldScene);
    }

    private void ResetBuildAndStartScene(SceneActor[] predefinedSceneActors) {
        // Reset the scene to blank
        ResetBoard(); // put the squares back in their reset position
        ReleaseBoard(); // Drop the squares
        actors = new List<Actor>();
        playerCount = 0;
        monsterCount = 0;
        currentActorTurn = -1; // -1 so turns actually start a 0

        // Build scene objects from predefined
        foreach (SceneActor actorData in predefinedSceneActors) {
            // Create GameObject and place it in the correct square
            GameObject newGameObject;
            if (actorData.IsPlayer) {
                newGameObject = (GameObject)
Instantiate(instance.PlayerPrefabList[actorData.PrefabIndex]);
```

```
                playerCount++;
            } else {
                newGameObject = (GameObject)
Instantiate(instance.MonsterPrefabsList[actorData.PrefabIndex]);
                monsterCount++;
            }
            Space spaceToPlace = spaces[actorData.x, actorData.z];
            Vector3 squareBasis = spaceToPlace.gameSpace.transform.position;
            newGameObject.transform.position = new Vector3(squareBasis.x, DropFromHeight + 1,
squareBasis.z);

            TokenStats stats = newGameObject.GetComponent<TokenStats>();
            Actor newActor = new Actor(newGameObject, actorData.x, actorData.z,
actorData.ActorColor, actorData.IsPlayer, stats.characterName, stats.HP, stats.AC,
                stats.InitativeMod, stats.Speed, stats.AttackName, stats.AttackRange,
stats.AttackMod, stats.DamageDiceNum,
                stats.DamageDiceMagnitude, stats.DamageMod);
            spaces[actorData.x, actorData.z].isBlocked = true;

            actors.Add(newActor);
        }

        // Show UI
        InGameCanvas.SetActive(true);

        // Roll init and sort
        RollInit();

        // Start the action!
        NextTurn();
    }

    // Instantiate square objects, but don't make them active yet
    private void GenerateSquares() {
        // Set up X,Z containers
        for (int x = 0; x < RowsX; x++) {
            for (int z = 0; z < ColsZ; z++) {
                spaces[x, z] = new Space(x, z, false);
            }
        }

        // Block any spaces that are impassible
        // A tree!
        spaces[29, 14].isBlocked = true;
        // A big rock!
        spaces[12, 32].isBlocked = true;
        spaces[13, 25].isBlocked = true;
        spaces[13, 26].isBlocked = true;
        spaces[13, 32].isBlocked = true;
        spaces[14, 26].isBlocked = true;
        spaces[14, 27].isBlocked = true;
        spaces[14, 28].isBlocked = true;
        spaces[14, 29].isBlocked = true;
        spaces[14, 30].isBlocked = true;
        spaces[14, 31].isBlocked = true;
        spaces[14, 32].isBlocked = true;
        spaces[15, 27].isBlocked = true;
        spaces[15, 28].isBlocked = true;
        spaces[15, 29].isBlocked = true;
        spaces[15, 30].isBlocked = true;
        spaces[15, 31].isBlocked = true;

        for (int x = 0; x < RowsX; x++) {
            for (int z = 0; z < ColsZ; z++) {
                if (!spaces[x, z].isBlocked) {
                    spaces[x, z].gameSpace = (GameObject) Instantiate(instance.OneByOnePrefab,
SpacesHolder.transform);
                }
            }
        }
```

```
    }

    // Place squares back in the original position for a new game scenario
    private void ResetBoard() {
        // Hide menu
        MenuCanvas.SetActive(false);

        // Remove any actors that are still on the board
        if (actors != null) {
            foreach (Actor actor in actors) {
                Destroy(actor.tokenRef);
                spaces[actor.x, actor.z].isBlocked = false;
            }
        }

        // Reset the squares back to their position, ready to be dropped
        for (int x = 0; x < RowsX; x++) {
            for (int z = 0; z < ColsZ; z++) {
                if (!spaces[x, z].isBlocked) {
                    spaces[x, z].gameSpace.transform.position = new Vector3(x + Margin,
DropFromHeight, z + Margin);
                    spaces[x, z].gameSpace.SetActive(false);
                }
            }
        }
    }

    // Re-activate all squares so they fall
    private void ReleaseBoard() {
        for (int x = 0; x < RowsX; x++) {
            for (int z = 0; z < ColsZ; z++) {
                if (!spaces[x, z].isBlocked) {
                    spaces[x, z].gameSpace.SetActive(true);
                }
            }
        }
    }

    // Establish the turn order of all monsters
    private void RollInit() {
        foreach (Actor actor in actors) {
            actor.RollInit();
        }
        actors.Sort((a, b) => b.Initative.CompareTo(a.Initative));
        UpdateTurnTracker();
    }

    // Recreate the list of tokens shon to the user
    private void UpdateTurnTracker() {
        string turnTrackerList = "";
        foreach (Actor actor in actors) {
            if (actor.IsAlive) {
                turnTrackerList += actor.Initative + " - " + actor.ActorName + "\n";
            }
        }
        TextTurnTracker.text = turnTrackerList;
    }

    // Advance play to the next turn
    public void NextTurn() {
        // Turn off highlight for previous token
        if (currentActorTurn >= 0) { // skip for first turn
            ((Behaviour) actors[currentActorTurn].tokenRef.GetComponent("Halo")).enabled = false;
        }

        // Update counter for new turn (skipping killed actors)
        int infinteLoopGuard = actors.Count + 1; // paranoid that Unity will crash on me again....
        do {
            currentActorTurn = (currentActorTurn + 1) % actors.Count;
            infinteLoopGuard--;
```

```
        } while (!actors[currentActorTurn].IsAlive || infinteLoopGuard < 0);
        if (infinteLoopGuard < 0) {
            Debug.Log("INFINTE LOOP!");
        }

        // Set text for this actor
        TextCurrentActor.text = actors[currentActorTurn].ActorName;
        TextHP.text = "HP: " + actors[currentActorTurn].HP;
        TextAC.text = "AC: " + actors[currentActorTurn].AC;
        TextAtkName.text = actors[currentActorTurn].AttackName;
        TextAtkRoll.text = "1d20 + " + actors[currentActorTurn].AttackMod;
        TextDmgRoll.text = actors[currentActorTurn].DamageDieNum + "d" +
actors[currentActorTurn].DamageDieMagnitude + " + " + actors[currentActorTurn].DamageMod;
        TextSpeedLeft.text = actors[currentActorTurn].Speed + " Spaces";

        // Define struct to keep track what'll be happening this turn
        currentTurnStats = new Turn {MovementLeft = actors[currentActorTurn].Speed};

        // Change visuals for this actor's turn
        MouseHoverHighlight.MouseOverColor = actors[currentActorTurn].ActorColor;

        // Set state
        SetState(STATES.AWAITING_INPUT);
    }
    // Contains the information for a current turn. Temporary: will be deleted after one turn is
done
    public class Turn {
        public int MovementLeft;
        public bool HasAttackHappened = false;
    }

    public void CheckForTurnCompleted() {
        if (currentTurnStats.MovementLeft == 0 && currentTurnStats.HasAttackHappened) {
            // Current turn actor is out of movement and has already attacked
            NextTurn();
        }
    }


    // Resolve an attack action
    // Recevied from any arbitrary GameObject with the OnClick-Message script attached
    public void MessageClickedToken(GameObject attackee) {
        SetState(STATES.ANIMATING_ACTION);
        if (currentTurnStats.HasAttackHappened) {
            PopupTextController.PopupText("Already attacked", attackee.transform);
        } else {
            GameObject attacker = actors[currentActorTurn].tokenRef;
            if (attackee == attacker) {
                PopupTextController.PopupText("Can't attack self", attackee.transform);
            } else {
                Actor victim = actors.Find(actor => { return actor.tokenRef == attackee; });
                if (victim == null) {
                    PopupTextController.PopupText("ERROR FINDING ACTOR", attackee.transform);
                } else {
                    if (!victim.IsAlive) {
                        PopupTextController.PopupText("Creature is already dead",
attackee.transform);
                    } else {
                        // Check if attack is possible, using A* pathfinding to find range in num
squares, manhattan distance
                        if (Pathfind.FindDistance(actors[currentActorTurn].x,
actors[currentActorTurn].z, victim.x, victim.z) > actors[currentActorTurn].AttackRange) {
                            PopupTextController.PopupText("Out of range", attackee.transform);
                        } else {
                            // Roll to hit
                            int attackResult = RollDice(1, 20, actors[currentActorTurn].AttackMod);
                            if (attackResult >= victim.AC) {
                                PopupTextController.PopupText("Hit: " + attackResult + " vs. " +
victim.AC, attacker.transform);
```

```
                                // Animate attack

attacker.GetComponent<TokenAttacker>().AttackTowards(attackee.transform);

                                int damageResult = RollDice(actors[currentActorTurn].DamageDieNum,
actors[currentActorTurn].DamageDieMagnitude, actors[currentActorTurn].DamageMod);
                                victim.HP -= damageResult;

                                delayedMessage = damageResult + " damage";
                                delayedActor = victim;
                                Invoke("DelayDamagePopup", 0.5f);
                                return;
                            } else {
                                PopupTextController.PopupText("Miss: " + attackResult + " vs. " +
victim.AC, attackee.transform);
                            }

                            // Finalise attack
                            currentTurnStats.HasAttackHappened = true;
                        }
                    }
                }
            }
        }
        SetState(STATES.AWAITING_INPUT);
        CheckForTurnCompleted();
    }
    private Actor delayedActor;
    private string delayedMessage;
    private void DelayDamagePopup() {
        PopupTextController.PopupText(delayedMessage, delayedActor.tokenRef.transform);
        CheckForDeath(delayedActor);
        SetState(STATES.AWAITING_INPUT);
        CheckForTurnCompleted();
    }

    public void CheckForDeath(Actor actor) {
        if (actor.HP <= 0) {
            actor.IsAlive = false; // Note: still blocking its space, which is fine!
            UpdateTurnTracker();
            KillAnimation(actor.tokenRef);
            if (actor.IsPlyaer) {
                playerCount--;
            } else {
                monsterCount--;
            }
            Invoke("CheckForGameOver", 1.1f);
        }
    }

    private void KillAnimation(GameObject actorTokenRef) {
        actorTokenRef.transform.position += new Vector3(0.3f, 0.5f, 0);
        toResetFreeze = actorTokenRef.GetComponent<Rigidbody>();

        // allow only Z rotation
        toResetFreeze.constraints = RigidbodyConstraints.FreezePositionX |
RigidbodyConstraints.FreezePositionZ | RigidbodyConstraints.FreezeRotationX |
                                    RigidbodyConstraints.FreezeRotationY;

        // Tap! Fall down
        toResetFreeze.AddTorque(new Vector3(0, 0, 1.5f)); // rotate along Z axis;

        // Lock back in place after it has a chance to fall down
        Invoke("ReFreeze", 1f);
    }
    private Rigidbody toResetFreeze;
    private void ReFreeze() {
        toResetFreeze.constraints = RigidbodyConstraints.FreezeRotation |
RigidbodyConstraints.FreezePositionX | RigidbodyConstraints.FreezePositionZ;
    }
```

```
    private void CheckForGameOver() {
        if (playerCount < 1) {
            MonstersWinMessage.SetActive(true);
            PlayersWinMessage.SetActive(false);
            SetState(STATES.MENU);
        } else if (monsterCount < 1) {
            MonstersWinMessage.SetActive(false);
            PlayersWinMessage.SetActive(true);
            SetState(STATES.MENU);
        }
    }


    // Resolve a walk action
    // Recevied from any arbitrary GameObject with the OnClick-Message script attached
    public void MessageClickedSpace(Vector2 coord) {
        WalkActor(actors[currentActorTurn], (int) coord.x, (int) coord.y);
    }

    // Walk a player or monster token to a space
    private void WalkActor(Actor actor, int xTo, int zTo) {
        // Find a path to the desired square, by getting a queue of sqaures to hop over
        LinkedList<TokenWalker.Hop> hopsQueue = Pathfind.FindPath(actor.x, actor.z, xTo, zTo);

        if (hopsQueue != null) {
            if (hopsQueue.Count > currentTurnStats.MovementLeft) {
                PopupTextController.PopupText("Not Enough Movement", spaces[xTo,
zTo].gameSpace.transform);
            } else {
                // change the token's stored properties to its final position
                spaces[actor.x, actor.z].isBlocked = false;
                actor.x = xTo;
                actor.z = zTo;
                spaces[xTo, zTo].isBlocked = true;

                SetState(STATES.ANIMATING_ACTION);

                // Use the script attached to the token to walk the path
                actor.tokenRef.GetComponent<TokenWalker>().WalkPath(hopsQueue);
            }
        } else {
            PopupTextController.PopupText("Pathfinding failed", spaces[xTo,
zTo].gameSpace.transform);
        }
    }

    void Update() {
        // Move the camera along the diagonals
        float deltaX = 0f, deltaZ = 0f;
        if (Input.GetKey(KeyCode.A)) {
            deltaX += cameraSpeed * Time.deltaTime;
            deltaZ -= cameraSpeed * Time.deltaTime;
        } else if (Input.GetKey(KeyCode.D)) {
            deltaX -= cameraSpeed * Time.deltaTime;
            deltaZ += cameraSpeed * Time.deltaTime;
        }
        if (Input.GetKey(KeyCode.W)) {
            deltaX -= cameraSpeed * Time.deltaTime;
            deltaZ -= cameraSpeed * Time.deltaTime;
        } else if (Input.GetKey(KeyCode.S)) {
            deltaX += cameraSpeed * Time.deltaTime;
            deltaZ += cameraSpeed * Time.deltaTime;
        }
        if (deltaX != 0f || deltaZ != 0f) {
            Camera.transform.position = new Vector3(Camera.transform.position.x + deltaX,
Camera.transform.position.y, Camera.transform.position.z + deltaZ);
        }

        if (state == STATES.AWAITING_INPUT) {
```

```
            if (Input.GetKey(KeyCode.Space) && lastInputTime + 1f < Time.time) {
                lastInputTime = Time.time;
                NextTurn();
            }
        }
    }
    private float lastInputTime = 0f; // Used to limit turn skipping, because hitting the spacebar
can sometimes be read as holding it down, and skips several turns

    // A struct to hold information about the game board spaces
    public class Space {
        public GameObject gameSpace = null; // public reference to the OneByOne GameObject pointed
to by this space
        public int x, z; // public reference to this object's position in the grid
        public bool isBlocked; // Define if this space is impassible

        public Space(int x, int z, bool isBlocked) {
            this.x = x;
            this.z = z;
            this.isBlocked = isBlocked;
        }
    }

    // A class to define a Prebuilt Scenario, stored as an array of SceneActors
    // Stores each token's initial position and properties
    public class SceneActor {
        public bool IsPlayer; // grab GameObject from player list or monster list
        public int PrefabIndex; // which item in the list of players/monsters does this Actor refer
to?
        public int x, z; // location on the grid to start the token
        public Color ActorColor;

        public SceneActor(bool isPlayer, int prefabIndex, int x, int z, Color actorColor) {
            IsPlayer = isPlayer;
            PrefabIndex = prefabIndex;
            this.x = x;
            this.z = z;
            ActorColor = actorColor;
        }
    }

    // A struct to hold an actor on the game board
    // A list of these will make up a scene
    // They are generated at the start of every game from a combination of a GameObject's
TokenStats struct and a SceneActor struct
    public class Actor {
        public GameObject tokenRef;
        public int x, z;
        public bool IsPlyaer;
        public bool IsAlive = true;
        public readonly string ActorName;
        public int HP, AC, InitativeMod, Speed;
        public int Initative;
        public string AttackName;
        public int AttackRange, AttackMod, DamageDieNum, DamageDieMagnitude, DamageMod;
        public Color ActorColor; // the colour to surround this token with indicating it is the
active Actor, and to use as the cursor highlight

        public Actor(GameObject tokenRef, int x, int z, Color actorColor, bool isPlyaer, string
actorName, int hp, int ac, int initativeMod, int speed, string attackName,
            int attackRange, int attackMod, int damageDieNum, int damageDieMagnitude, int
damageMod) {
            this.tokenRef = tokenRef;
            this.x = x;
            this.z = z;
            ActorColor = actorColor;
            IsPlyaer = isPlyaer;
            ActorName = actorName;
            HP = hp;
            AC = ac;
```

```
            InitativeMod = initativeMod;
            Speed = speed;
            AttackName = attackName;
            AttackRange = attackRange;
            AttackMod = attackMod;
            DamageDieNum = damageDieNum;
            DamageDieMagnitude = damageDieMagnitude;
            DamageMod = damageMod;
        }

        public void RollInit() {
            Initative = RollDice(1, 20, InitativeMod);
        }
    }
}
```

TokenWalker.cs

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

public class TokenWalker : MonoBehaviour {
    private static readonly float HOP_ANIMATION_TIME = 0.5f;

    private LinkedList<Hop> hopsQueue;
    private GameObject tokenToAnimate;
    private Vector3 startPos, endPos, relativeStartPos, relativeEndPos, center;
    private float startTime, endTime;
    private bool isWalking;

    public void WalkPath(LinkedList<Hop> hopsQueue) {
       this.hopsQueue = hopsQueue;
        NextHop();
   }

    // Hop from one space to another space (probably right next to it)
    // start the hopping at the first one. will continue until hopsQueue is empty
    private void NextHop() {
        // Pop first hop off the queue
        if (hopsQueue.First != null) {
            // Get next hop out of queue
            Hop nextHop = hopsQueue.First.Value;
            hopsQueue.RemoveFirst(); // pop

            // Update UI to match one space moved
            GameManager.instance.currentTurnStats.MovementLeft -= 1;
            GameManager.instance.TextSpeedLeft.text =
GameManager.instance.currentTurnStats.MovementLeft + " Spaces";

            // Set up global variables for next hop. (Requried to be global to use InvokeRepeating
to loop through the animation.)
            startPos = GameManager.instance.spaces[nextHop.xFrom,
nextHop.zFrom].gameSpace.transform.position + GameManager.instance.SPACE_HEIGHT_MOD * 2;
            endPos = GameManager.instance.spaces[nextHop.xTo,
nextHop.zTo].gameSpace.transform.position + GameManager.instance.SPACE_HEIGHT_MOD * 2;
            center = (startPos + endPos) * 0.5f;
            center -= new Vector3(0, 0.1f, 0); // make circular movment a bit flatter (also, this
line is necessary to have the arc be along the Y plane)
            relativeStartPos = startPos - center;
            relativeEndPos = endPos - center;
            tokenToAnimate = gameObject;
            startTime = Time.time;
            endTime = startTime + HOP_ANIMATION_TIME;
            isWalking = true;
        } else {
            GameManager.instance.SetState(GameManager.STATES.AWAITING_INPUT);
            GameManager.instance.CheckForTurnCompleted();
        }
    }
```

```
    // Update is called once per frame
    void Update () {
        if (isWalking) {
            if (Time.time < endTime) {
                // Using Slerp to make an arc of movement is from unity manual:
https://docs.unity3d.com/ScriptReference/Vector3.Slerp.html
                //     and this post: https://answers.unity.com/questions/11184/moving-player-in-
an-arc-from-startpoint-to-endpoin.html
                tokenToAnimate.transform.position = Vector3.Slerp(relativeStartPos, relativeEndPos,
(Time.time - startTime) / (endTime - startTime));
                tokenToAnimate.transform.position += center;
            } else {
                isWalking = false;
                NextHop();
            }
        }
    }

    public class Hop {
        public int xFrom, zFrom, xTo, zTo;
        public Hop(int xFrom, int zFrom, int xTo, int zTo) { this.xFrom = xFrom; this.zFrom =
zFrom; this.xTo = xTo; this.zTo = zTo; }
    }

}
```

Pathfind.cs

```
using System.Collections.Generic;

public class Pathfind { // Does not use Unity at all, so don't extend MonoBehaviour

    private static bool careIfPathIsBlocked = true;
    public static int FindDistance(int xFrom, int zFrom, int xTo, int zTo) {
        careIfPathIsBlocked = false; // Distance doesn't care if the path is path is blocked!
        LinkedList<TokenWalker.Hop> path = FindPath(xFrom, zFrom, xTo, zTo);
        careIfPathIsBlocked = true;
        if (path == null) {
            return -1;
        } else {
            return path.Count;
        }
    }

    /**
     * Find a path using A*, and return it as a "stack" (i.e. LinkedList, but please pop off the
Front)
     * NOTE: This uses my code that I submitted for assignment 10 from CT255 that I completed in
Spring of 2016.
     *          It is largely unchanged, except converting from Java -> C#
     */
    public static LinkedList<TokenWalker.Hop> FindPath(int xFrom, int zFrom, int xTo, int zTo) {
        // **** do not pathfind to own square ****
        if (xTo == xFrom && zTo == zFrom) {
            return null;
        }

        // **** create data structures ****
        Node[,] nodes = new Node[GameManager.instance.RowsX, GameManager.instance.ColsZ];
        LinkedList<Node> openList = new LinkedList<Node>();

        // **** set initial conditions ****
        // create node objects and set walls to closed
        for (int row = 0; row < GameManager.instance.RowsX; ++row) {
            for (int col = 0; col < GameManager.instance.ColsZ; ++col) {
                nodes[col, row] = new Node { x = col, z = row };
                if (careIfPathIsBlocked && GameManager.instance.spaces[col, row].isBlocked) {
                    nodes[col, row].isClosed = true;
```

```
                }
            }
        }

        // **** add initial node to open list ****
        Node initialNode = nodes[xFrom, zFrom];
        initialNode.g = 0; // condition of the initial node
        initialNode.parent = null; // leaving this null will be the termination signal for the
found path
        openList.AddLast(initialNode);

        // **** loop through nodes on open list until a path is found or list is empty ****
        Node curr; // the node we've just popped off the open list
        Node nearby; // hold nodes to compare to the open node
        bool isPathFound = false;
        bool isMazeSolvable = true;
        while (!isPathFound && isMazeSolvable) {
            // 1. find the open node with lowest f
            curr = openList.First.Value;
            foreach (Node openNode in openList) {
            //for (Node openNode : openList) {
                if (openNode.f <= curr.f) { // by doing less or EQUAL, this biases towards items
examined last, i.e. the newer ones added to the open list
                    curr = openNode;
                }
            }
            // curr is now node with lowest f

            // 2. close node
            curr.isClosed = true;
            openList.Remove(curr);

            // 3. test for termination condition: if this node is the target, then quit,
successfully
            if (curr.x == xTo && curr.z == zTo) {
                isPathFound = true;
            }

            // 4. add all nodes surrounding current to open list, pointing back to current
            for (int deltaRow = -1; deltaRow <= 1; ++deltaRow) {
                if (curr.z + deltaRow == -1 || curr.z + deltaRow == GameManager.instance.RowsX) {
                    continue;
                }
                for (int deltaCol = -1; deltaCol <= 1; ++deltaCol) {
                    if (curr.x + deltaCol == -1 || curr.x + deltaCol == GameManager.instance.ColsZ)
{
                        continue;
                    }
                    nearby = nodes[curr.x + deltaCol, curr.z + deltaRow];

                    if (!nearby.isClosed) {
                        if (nearby.g == 0) { // first time examining this node
                            nearby.g = curr.g + 1;
                            nearby.h = System.Math.Abs(xTo - nearby.x) + System.Math.Abs(zTo -
nearby.z);
                            nearby.f = nearby.g + nearby.h;
                            nearby.parent = curr;
                            openList.AddLast(nearby);
                        } else { // have already examined this node, but it's not yet closed
                            if (curr.g + 1 < nearby.g) { // if already on open list, yet current
square would give it a better g: make it part of the current path instead
                                nearby.g = curr.g + 1;
                                // temp.h = this.heuristic(temp.x, temp.y); // don't need to recalc
heuristic
                                nearby.f = nearby.g + nearby.h;
                                nearby.parent = curr; // do need to change parent
                            }
                        }
                    }
                }
```

```
            }

            // 5. test for termination condition
            if (openList.Count == 0) {
                isMazeSolvable = false;
            }
        } // end algorithmic loop

        // **** if a path was found, save that path externally ****
        if (isMazeSolvable) {
            LinkedList<TokenWalker.Hop> pathStack = new LinkedList<TokenWalker.Hop>();
            //path = new Stack<>();
            curr = nodes[xTo, zTo];
            // Skip the first square in the path (the destination). Start the loop at the square
just before the end. (For Unity, I want "hops", but the original Java was designed to return the
whole path.)
            if (curr != null) {
                curr = curr.parent;
            }

            // then traverse into path
            // first item on stack should be xTo,zTo
            int prevX = xTo, prevZ = zTo;
            while (curr != null) {
                pathStack.AddFirst(new TokenWalker.Hop(curr.x, curr.z, prevX, prevZ));
                prevX = curr.x;
                prevZ = curr.z;

                curr = curr.parent;
            }
            return pathStack;
        } else {
            // didn't find a path, so let monster just do a dumb run towards the player, into the
dead end
            return null;
        }
    }

    // helper class for the A* algorithm
    // all fields are simply publicly accessible!
    private class Node {
        public int x, z;
        public Node parent = null;
        public int g, h, f;
        public bool isClosed = false;
    }
}
```

TokenStats.cs

```
using UnityEngine;

// A struct to define the stats of this Token. Set in the Inspector
public class TokenStats : MonoBehaviour {
    public string characterName;
    public int HP;
    public int AC;
    public int InitativeMod;
    public int Speed;

    public string AttackName;
    public int AttackRange;
    public int AttackMod;
    public int DamageDiceNum;
    public int DamageDiceMagnitude;
    public int DamageMod;
}
```

TokenAttacker.cs

```
using UnityEngine;

public class TokenAttacker : MonoBehaviour {
    public GameObject animatedWeaponPrefab;

    public void AttackTowards(Transform victim) {
        // Face the GO that's being attacked by creating a point that is in the victim's X and Z,
but locked to the current Y (so token doesn't tilt).
        //    See: https://answers.unity.com/answers/250578/view.html
        Vector3 victimSpotLevelled = new Vector3(victim.position.x, this.transform.position.y,
victim.position.z);
        transform.LookAt(victimSpotLevelled);

        // Spawn a floating weapon, and face it the right way
        GameObject floatingWeapon = (GameObject) Instantiate(animatedWeaponPrefab, transform);
        floatingWeapon.transform.position = transform.position + 0.5f * transform.forward +
Vector3.up; // token's position + token's facing (scaled back) + 1m high
        floatingWeapon.transform.LookAt(victimSpotLevelled + Vector3.up);

        // Weapon will animate itself, and animation contains the routine to start the particle
effects
        // Weapon also takes care of removing itself
    }
}
```

WeaponSelfActions.cs

```
using UnityEngine;

public class WeaponSelfActions : MonoBehaviour {
    // Fire off attack effects. (ParticleSystem should not be playing on start)
    // This function will be assigned to an Animation Event
    private void AttackEffects() {
        GetComponentInChildren<ParticleSystem>().Play(); // Not looped, so no need to Stop()
    }

    // Delete the parent GO
    // This function will be assigned to an Animation Event
    private void RemoveWeapon() {
        Destroy(transform.parent.gameObject);
    }
}
```

PopupTextController.cs

```
using UnityEngine;

// Method of creating a popup text is by GameGrind on: https://www.youtube.com/watch?v=fbUOG7f3jq8
public class PopupTextController : MonoBehaviour {
    private static PopupText popupText;
    private static GameObject canvas;

    public static void Initialize() {
        popupText = Resources.Load<PopupText>("PopupTextParent");
//        canvas = GameObject.Find("InGameCanvas");
        canvas = GameManager.instance.InGameCanvas;
    }

    public static void PopupText(string text, Transform attachTo) {
        PopupText textGameObject = Instantiate(popupText);
        textGameObject.SetText(text);

        // Set position
        textGameObject.transform.SetParent(canvas.transform, false);
        // Map 3d position of transform we are attaching to into flat camera/screen position
        textGameObject.transform.position = (Vector2)
Camera.main.WorldToScreenPoint(attachTo.position);
```

```
    }
}
```

PopupText.cs

```
using UnityEngine;
using UnityEngine.UI;

// Method of creating a popup text is by GameGrind on: https://www.youtube.com/watch?v=fbUOG7f3jq8
public class PopupText : MonoBehaviour {
    public Text textObject;

    private void OnEnable() {
        Destroy(gameObject, 0.8f);
    }

    public void SetText(string text) {
        textObject.text = text;
    }
}
```

OnClickMsgClickedSpace.cs

```
using UnityEngine;
using UnityEngine.EventSystems;

public class OnClickMsgClickedSpace : MonoBehaviour {
//    [HideInInspector] public int x, z;

    private void OnMouseDown() {
        if (!EventSystem.current.IsPointerOverGameObject() && MouseHoverHighlight.isEffectActive) {
            int x = (int) gameObject.transform.position.x; // Use Integer truncation to find X, Z
of space
            int z = (int) gameObject.transform.position.z;
            GameManager.instance.SendMessage("MessageClickedSpace", new Vector2(x, z)); // Using a
Vector2 to hold an X,Z because SendMessage can only handle ONE param
        }
    }
}
```

OnClickMsgClickedToken.cs

```
using UnityEngine;
using UnityEngine.EventSystems;

public class OnClickMsgClickedToken : MonoBehaviour {
    private void OnMouseDown() {
        if (!EventSystem.current.IsPointerOverGameObject() && MouseHoverHighlight.isEffectActive) {
            //GameManager.instance.SendMessage("MessageClickedToken", GetComponent<TokenStats>());
            GameManager.instance.SendMessage("MessageClickedToken", gameObject);
        }
    }
}
```

MouseHoverHighlight.cs

```
using UnityEngine;

// Script is taken from Unity Docs:
https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnMouseOver.html
// Modifed to have its hovor colour modified from an outside script
public class MouseHoverHighlight : MonoBehaviour {

    public static bool isEffectActive = false;

    //This second example changes the GameObject's color to red when the mouse hovers over it
    //Ensure the GameObject has a MeshRenderer
```

```
    //When the mouse hovers over the GameObject, it turns to this color (red)
    [HideInInspector] public static Color MouseOverColor;
    //This stores the GameObject's original color
    private Color m_OriginalColor;
    //Get the GameObject's mesh renderer to access the GameObject's material and color
    private MeshRenderer m_Renderer;

    void Start()
    {
        //Fetch the mesh renderer component from the GameObject
        m_Renderer = GetComponent<MeshRenderer>();
        //Fetch the original color of the GameObject
        m_OriginalColor = m_Renderer.material.color;
    }

    void OnMouseOver()
    {
        //Change the color of the GameObject to red when the mouse is over GameObject
        if (isEffectActive) {
            m_Renderer.material.color = MouseOverColor;
        } else {
            m_Renderer.material.color = m_OriginalColor;
        }
    }

    void OnMouseExit()
    {
        //Reset the color of the GameObject back to normal
        m_Renderer.material.color = m_OriginalColor;
    }
}
```

MouseHoverHighlightChildren.cs

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

// Script is taken from Unity Docs:
https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnMouseOver.html
// Modifed to work with multiple MeshRenderers on one object
public class MouseHoverHighlightChildren : MouseHoverHighlight {


    // uses MouseHoverHighlight.MouseOverColor to change colour

    //This stores the GameObject's original color
    private List<Color> m_OriginalColorList = new List<Color>();
    //Get the GameObject's mesh renderer to access the GameObject's material and color
    private List<MeshRenderer> m_RendererList = new List<MeshRenderer>();

    void Start()
    {
        //Fetch the mesh renderer component from the GameObject
        GetComponentsInChildren<MeshRenderer>(m_RendererList);
        //Fetch the original color of the GameObject
        IEnumerator<MeshRenderer> i = m_RendererList.GetEnumerator();
        while (i.MoveNext()) {
            m_OriginalColorList.Add(i.Current.material.color);
        }
    }

    void OnMouseOver()
    {
        //Change the color of the GameObject to red when the mouse is over GameObject
        if (MouseHoverHighlight.isEffectActive) {
            IEnumerator<MeshRenderer> i = m_RendererList.GetEnumerator();
            while (i.MoveNext()) {
                i.Current.material.color = MouseHoverHighlight.MouseOverColor;
```

```
            }
        } else {
            IEnumerator<MeshRenderer> i = m_RendererList.GetEnumerator();
            IEnumerator<Color> c = m_OriginalColorList.GetEnumerator();
            while (i.MoveNext() && c.MoveNext()) {
                i.Current.material.color = c.Current;
            }
        }
    }

    void OnMouseExit()
    {
        //Reset the color of the GameObject back to normal
        IEnumerator<MeshRenderer> i = m_RendererList.GetEnumerator();
        IEnumerator<Color> c = m_OriginalColorList.GetEnumerator();
        while (i.MoveNext() && c.MoveNext()) {
            i.Current.material.color = c.Current;
        }
    }
}
```

MouseLooks.cs

```
using UnityEngine;

// This MouseLook script is taken directly from a Unity Answer:
https://answers.unity.com/answers/1135844/view.html
// THIS IS NOT MY CODE
public class MouseLook : MonoBehaviour
{
    public float mouseSensitivity = 100.0f;
    public float clampAngle = 80.0f;

    private float rotY = 0.0f; // rotation around the up/y axis
    private float rotX = 0.0f; // rotation around the right/x axis

    void Start()
    {
        // Start camera a bit above the ground, and pointing at the middle (MY CODE)
        transform.position = new Vector3(30f, 10f, 30f);
        transform.LookAt(new Vector3(20f, 0f, 20f));
        // END OF MY CODE

        Vector3 rot = transform.localRotation.eulerAngles;
        rotY = rot.y;
        rotX = rot.x;
    }

    void Update()
    {
        float mouseX = Input.GetAxis("Mouse X");
        float mouseY = -Input.GetAxis("Mouse Y");

        rotY += mouseX * mouseSensitivity * Time.deltaTime;
        rotX += mouseY * mouseSensitivity * Time.deltaTime;

        rotX = Mathf.Clamp(rotX, -clampAngle, clampAngle);

        Quaternion localRotation = Quaternion.Euler(rotX, rotY, 0.0f);
        transform.rotation = localRotation;
    }
}
```