

## Enemy\_4.cs

```
1 using UnityEngine;
2 using System.Collections;
3
4 // Part is another serializable data storage class just like WeaponDefinition
5 [System.Serializable]
6 public class Part {
7     // These three fields need to be defined in the Inspector pane
8     public string name; // The name of this part
9     public float health; // The amount of health this part has
10    public string[] protectedBy; // The other parts that protect this
11
12    // These two fields are set automatically in Start().
13    // Caching like this makes it faster and easier to find these later
14    public GameObject go; // The GameObject of this part
15    public Material mat; // The Material to show damage
16 }
17
18 public class Enemy_4 : Enemy {
19     // Enemy_4 will start offscreen and then pick a random point on screen to
20     // move to. Once it has arrived, it will pick another random point and
21     // continue until the player has shot it down.
22
23     public Vector3[] points; // Stores the p0 & p1 for interpolation
24     public float timeStart; // Birth time for this Enemy_4
25     public float duration = 4; // Duration of movement
26
27     public Part[] parts; // The array of ship Parts
28
29     void Start () {
30         points = new Vector3[2];
31         // There is already an initial position chosen by Main.SpawnEnemy()
32         // so add it to points as the initial p0 & p1
33         points[0] = pos;
34         points[1] = pos;
35
36         InitMovement();
37
38         // Cache GameObject & Material of each Part in parts
39         Transform t;
40         foreach(Part prt in parts) {
41             t = transform.Find(prt.name);
42             if (t != null) {
43                 prt.go = t.gameObject;
44                 prt.mat = prt.go.GetComponent<Renderer>().material;
45             }
46         }
47     }
48
49     void InitMovement() {
50         // Pick a new point to move to that is on screen
51         Vector3 p1 = Vector3.zero;
52         float esp = Main.S.enemySpawnPadding;
53         Bounds cBounds = Utils.camBounds;
54         p1.x = Random.Range(cBounds.min.x + esp, cBounds.max.x - esp);
55         p1.y = Random.Range(cBounds.min.y + esp, cBounds.max.y - esp);
56
57         points[0] = points[1]; // Shift points[1] to points[0]
58         points[1] = p1; // Add p1 as points[1]
59
60         // Reset the time
61         timeStart = Time.time;
62     }
63
64 }
```

```

65 public override void Move () {
66     // This completely overrides Enemy.Move() with a linear interpolation
67     float u = (Time.time-timeStart)/duration;
68     if (u>=1) { // if u >=1...
69         InitMovement(); // ...then initialize movement to a new point
70         u=0;
71     }
72
73     u = 1 - Mathf.Pow( 1-u, 2 ); // Apply Ease Out easing to u
74
75     // This line is the same as: pos = (1-u)*points[0] + u*points[1];
76     pos = (1-u)*points[0] + u*points[1];
77 }
78
79 // This will override the OnCollisionEnter that is part of Enemy.cs
80 // Because of the way that MonoBehaviour declares common Unity functions
81 // like OnCollisionEnter(), the override keyword is not necessary.
82 void OnCollisionEnter( Collision coll ) {
83     GameObject other = coll.gameObject;
84     switch (other.tag) {
85     case "ProjectileHero":
86         Projectile p = other.GetComponent<Projectile>();
87         // Enemies don't take damage unless they're on screen
88         // This stops the player from shooting them before they are visible
89         bounds.center = transform.position + boundsCenterOffset;
90         if (bounds.extents == Vector3.zero || Utils.ScreenBoundsCheck(bounds,
91             ↳BoundsTest.offScreen) != Vector3.zero) {
92             Destroy(other);
93             break;
94         }
95
96         // Hurt this Enemy
97         // Find the GameObject that was hit
98         // The Collision coll has contacts[], an array of ContactPoints
99         // Because there was a collision, we're guaranteed that there is at
100        // least a contacts[0], and ContactPoints have a reference to
101        // thisCollider, which will be the collider for the part of the
102        // Enemy_5 that was hit.
103        GameObject goHit = coll.contacts[0].thisCollider.gameObject;
104        Part prtHit = FindPart(goHit);
105        if (prtHit == null) { // If prtHit wasn't found
106            // ...then it's usually because, very rarely, thisCollider on
107            // contacts[0] will be the ProjectileHero instead of the ship
108            // part. If so, just look for otherCollider instead
109            goHit = coll.contacts[0].otherCollider.gameObject;
110            prtHit = FindPart(goHit);
111        }
112        // Check whether this part is still protected
113        if (prtHit.protectedBy != null) {
114            foreach( string s in prtHit.protectedBy ) {
115                // If one of the protecting parts hasn't been destroyed...
116                if (!Destroyed(s)) {
117                    // ...then don't damage this part yet
118                    Destroy(other); // Destroy the ProjectileHero
119                    return; // return before causing damage
120                }
121            }
122        }
123        // It's not protected, so make it take damage
124        // Get the damage amount from the Projectile.type & Main.W_DEFS
125        prtHit.health -= Main.W_DEFS[p.type].damageOnHit;
126        // Show damage on the part
127        ShowLocalizedDamage(prtHit.mat);
128        if (prtHit.health <= 0) {
129            // Instead of Destroying this enemy, disable the damaged part

```

```

129         prtHit.go.SetActive(false);
130     }
131     // Check to see if the whole ship is destroyed
132     bool allDestroyed = true; // Assume it is destroyed
133     foreach( Part prt in parts ) {
134         if (!Destroyed(prt)) { // If a part still exists
135             allDestroyed = false; // ...change allDestroyed to false
136             break; // and break out of the foreach loop
137         }
138     }
139     if (allDestroyed) { // If it IS completely destroyed
140         // Tell the Main singleton that this ship has been destroyed
141         Main.S.ShipDestroyed( this );
142         // Destroy this Enemy
143         Destroy(this.gameObject);
144     }
145     Destroy(other); // Destroy the ProjectileHero
146     break;
147 }
148 }
149
150 // These two functions find a Part in parts based on name or GameObject
151 Part FindPart(string n) {
152     foreach( Part prt in parts ) {
153         if (prt.name == n) {
154             return( prt );
155         }
156     }
157     return( null );
158 }
159 Part FindPart(GameObject go) {
160     foreach( Part prt in parts ) {
161         if (prt.go == go) {
162             return( prt );
163         }
164     }
165     return( null );
166 }
167
168 // These functions return true if the Part has been destroyed
169 bool Destroyed(GameObject go) {
170     return( Destroyed( FindPart(go) ) );
171 }
172 bool Destroyed(string n) {
173     return( Destroyed( FindPart(n) ) );
174 }
175 bool Destroyed(Part prt) {
176     if (prt == null) { // If no real ph was passed in
177         return(true); // Return true (meaning yes, it was destroyed)
178     }
179     // Returns the result of the comparison: prt.health <= 0
180     // If prt.health is 0 or less, returns true (yes, it was destroyed)
181     return (prt.health <= 0);
182 }
183
184 // This changes the color of just one Part to red instead of the whole ship
185 void ShowLocalizedDamage(Material m) {
186     m.color = Color.red;
187     remainingDamageFrames = showDamageForFrames;
188 }
189 }

```

## Main.cs

```
1 using UnityEngine; // Required for Unity
2 using System.Collections; // Required for Arrays & other Collections
3 using System.Collections.Generic; // Required to use Lists or Dictionaries
4
5 public class Main : MonoBehaviour {
6     static public Main S;
7     static public Dictionary<WeaponType, WeaponDefinition> W_DEFS;
8
9     public GameObject[] prefabEnemies;
10    public float enemySpawnPerSecond = 0.5f; // # Enemies/second
11    public float enemySpawnPadding = 1.5f; // Padding for position
12    public WeaponDefinition[] weaponDefinitions;
13
14    public GameObject prefabPowerUp;
15
16    public WeaponType[] powerUpFrequency = new WeaponType[] {
17        WeaponType.blaster, WeaponType.blaster,
18        WeaponType.spread,
19        WeaponType.shield };
20
21    public bool _____;
22
23    public WeaponType[] activeWeaponTypes;
24    private float enemySpawnRate; // Delay between Enemies
25
26
27    void Awake() {
28        S = this;
29        // Set Utils.camBounds
30        Utils.SetCameraBounds(this.GetComponent<Camera>());
31        // 0.5 enemies/second = enemySpawnRate of 2
32        enemySpawnRate = 1f/enemySpawnPerSecond;
33        // Invoke 1 call to SpawnEnemy() in 2 seconds
34        Invoke( "SpawnEnemy", enemySpawnRate );
35
36        // A generic Dictionary with WeaponType as the key
37        W_DEFS = new Dictionary<WeaponType, WeaponDefinition>();
38        foreach( WeaponDefinition def in weaponDefinitions ) {
39            W_DEFS[def.type] = def;
40        }
41    }
42
43    static public WeaponDefinition GetWeaponDefinition( WeaponType wt ) {
44        // Check to make sure that the key exists in the Dictionary
45        if ( W_DEFS.ContainsKey(wt) ) {
46            // Attempting to retrieve a key that doesn't exist, throws an error
47            return( W_DEFS[wt] );
48        }
49        // This will return a definition for WeaponType.none,
50        // which means it has failed to find the WeaponDefinition
51        return( new WeaponDefinition() );
52    }
53
54    void Start() {
55        activeWeaponTypes = new WeaponType[weaponDefinitions.Length];
56        for ( int i=0; i<weaponDefinitions.Length; i++ ) {
57            activeWeaponTypes[i] = weaponDefinitions[i].type;
58        }
59    }
60
61    public void SpawnEnemy() {
62        // Pick a random Enemy prefab to instantiate
63        int ndx = Random.Range(0, prefabEnemies.Length);
64        GameObject go = Instantiate( prefabEnemies[ ndx ] ) as GameObject;
```

```

65     // Position the Enemy above the screen with a random x position
66     Vector3 pos = Vector3.zero;
67     float xMin = Utils.camBounds.min.x+enemySpawnPadding;
68     float xMax = Utils.camBounds.max.x-enemySpawnPadding;
69     pos.x = Random.Range( xMin, xMax );
70     pos.y = Utils.camBounds.max.y + enemySpawnPadding;
71     go.transform.position = pos;
72     // Call SpawnEnemy() again in a couple seconds
73     Invoke( "SpawnEnemy", enemySpawnRate );
74 }
75
76 public void DelayedRestart( float delay ) {
77     // Invoke the Restart() method in delay seconds
78     Invoke("Restart", delay);
79 }
80
81 public void Restart() {
82     // Reload _Scene_0 to restart the game
83     Application.LoadLevel("_Scene_0");
84 }
85
86 public void ShipDestroyed( Enemy e ) {
87     // Potentially generate a PowerUp
88     if (Random.value <= e.powerUpDropChance) {
89         // Random.value generates a value between 0 & 1 (though never == 1)
90         // If the e.powerUpDropChance is 0.50f, a PowerUp will be generated
91         // 50% of the time. For testing, it's now set to 1f.
92
93         // Choose which PowerUp to pick
94         // Pick one from the possibilities in powerUpFrequency
95         int ndx = Random.Range(0,powerUpFrequency.Length);
96         WeaponType puType = powerUpFrequency[ndx];
97
98         // Spawn a PowerUp
99         GameObject go = Instantiate( prefabPowerUp ) as GameObject;
100        PowerUp pu = go.GetComponent<PowerUp>();
101        // Set it to the proper WeaponType
102        pu.SetType( puType );
103
104        // Set it to the position of the destroyed ship
105        pu.transform.position = e.transform.position;
106    }
107 }
108
109 }

```

## PowerUp.cs

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class PowerUp : MonoBehaviour {
5      // An unusual but useful use of Vector2s, x is a min value
6      // and y is a max for a Random.Range() to be called later
7      public Vector2      rotMinMax = new Vector2(15,90);
8      public Vector2      driftMinMax = new Vector2(.25f,2);
9      public float        lifeTime = 6f; // Seconds the PowerUp exist
10     public float        fadeTime = 4f; // Seconds it will then fade
11     public bool          _____;
12     public WeaponType    type; // The type of the PowerUp
13     public GameObject    cube; // Reference to the Cube child
14     public TextMesh      letter; // Reference to the TextMesh
15     public Vector3       rotPerSecond; // Euler rotation speed
16     public float         birthTime;
17
18     void Awake() {
19         // Find the Cube reference
20         cube = transform.Find("Cube").gameObject;
21         // Find the TextMesh
22         letter = GetComponent<TextMesh>();
23
24         // Set a random velocity
25         Vector3 vel = Random.onUnitSphere; // Get Random XYZ velocity
26         // Random.onUnitSphere gives you a vector point that is somewhere on
27         // the surface of the sphere with a radius of 1m around the origin
28         vel.z = 0; // Flatten the vel to the XY plane
29         vel.Normalize(); // Make the length of the vel 1
30         // Normalizing a Vector3 makes it length 1m
31         vel *= Random.Range(driftMinMax.x, driftMinMax.y);
32         // Above sets the velocity length to something between the x and y
33         // values of driftMinMax
34         GetComponent<Rigidbody>().velocity = vel;
35
36         // Set the rotation of this GameObject to R:[0,0,0]
37         transform.rotation = Quaternion.identity;
38         // Quaternion.identity is equal to no rotation.
39
40         // Set up the rotPerSecond for the Cube child using rotMinMax x & y
41         rotPerSecond = new Vector3( Random.Range(rotMinMax.x,rotMinMax.y),
42                                     Random.Range(rotMinMax.x,rotMinMax.y),
43                                     Random.Range(rotMinMax.x,rotMinMax.y) );
44
45         // CheckOffscreen() every 2 seconds
46         InvokeRepeating( "CheckOffscreen", 2f, 2f );
47
48         birthTime = Time.time;
49     }
50
51     void Update () {
52         // Manually rotate the Cube child every Update()
53         // Multiplying it by Time.time causes the rotation to be time-based
54         cube.transform.rotation = Quaternion.Euler( rotPerSecond*Time.time );
55
56         // Fade out the PowerUp over time
57         // Given the default values, a PowerUp will exist for 10 seconds
58         // and then fade out over 4 seconds.
59         float u = (Time.time - (birthTime+lifeTime)) / fadeTime;
60         // For lifeTime seconds, u will be <= 0. Then it will transition to 1
61         // over fadeTime seconds.
62         // If u >= 1, destroy this PowerUp
63         if (u >= 1) {
64             Destroy( this.gameObject );
65         }
66     }
67 }
```

```

65     return;
66 }
67 // Use u to determine the alpha value of the Cube & Letter
68 if (u>0) {
69     Color c = cube.GetComponent<Renderer>().material.color;
70     c.a = 1f-u;
71     cube.GetComponent<Renderer>().material.color = c;
72     // Fade the Letter too, just not as much
73     c = letter.color;
74     c.a = 1f - (u*0.5f);
75     letter.color = c;
76 }
77 }
78
79 public void SetType( WeaponType wt ) {
80     // Grab the WeaponDefinition from Main
81     WeaponDefinition def = Main.GetWeaponDefinition( wt );
82     // Set the color of the Cube child
83     cube.GetComponent<Renderer>().material.color = def.color;
84     //letter.color = def.color; // We could colorize the letter too
85     letter.text = def.letter; // Set the letter that is shown
86     type = wt; // Finally actually set the type
87 }
88
89 public void AbsorbedBy( GameObject target ) {
90     // This function is called by the Hero class when a PowerUp is collected
91     // We could tween into the target and shrink in size,
92     // but for now, just destroy this.gameObject
93     Destroy( this.gameObject );
94 }
95
96 void CheckOffscreen() {
97     // If the PowerUp has drifted entirely off screen
98     if ( Utils.ScreenBoundsCheck( cube.GetComponent<Collider>().bounds,
99         -BoundsTest.offScreen ) != Vector3.zero ) {
100         // ...then destroy this GameObject
101         Destroy( this.gameObject );
102     }
103 }

```

## Projectile.cs

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class Projectile : MonoBehaviour {
5      [SerializeField]
6      private WeaponType  _type;
7
8      // This public property masks the field _type & takes action when it is set
9      public WeaponType  type {
10         get {
11             return( _type );
12         }
13         set {
14             SetType( value );
15         }
16     }
17
18     void Awake() {
19         // Test to see whether this has passed off screen every 2 seconds
20         InvokeRepeating( "CheckOffscreen", 2f, 2f );
21     }
22
23     public void SetType( WeaponType eType ) {
24         // Set the _type
25         _type = eType;
26         WeaponDefinition def = Main.GetWeaponDefinition( _type );
27         GetComponent<Renderer>().material.color = def.projectileColor;
28     }
29
30     void CheckOffscreen() {
31         if ( Utils.ScreenBoundsCheck( GetComponent<Collider>().bounds,
32             ↳BoundsTest.offScreen ) != Vector3.zero ) {
33             Destroy( this.gameObject );
34         }
35     }
36 }
```



## Shield.cs

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class Shield : MonoBehaviour {
5      public float   rotationsPerSecond = 0.1f;
6      public bool   _____;
7      public int    levelShown = 0;
8
9      void Update () {
10         // Read the current shield level from the Hero Singleton
11         int currLevel = Mathf.FloorToInt( Hero.S.shieldLevel );
12         // If this is different from levelShown...
13         if (levelShown != currLevel) {
14             levelShown = currLevel;
15             Material mat = this.GetComponent<Renderer>().material;
16             // Adjust the texture offset to show different shield level
17             mat.mainTextureOffset = new Vector2( 0.2f*levelShown, 0 );
18         }
19         // Rotate the shield a bit every second
20         float rZ = (rotationsPerSecond*Time.time*360) % 360f;
21         transform.rotation = Quaternion.Euler( 0, 0, rZ );
22     }
23
24 }
```

## Utils.cs

```
1  using UnityEngine;
2  using System.Collections;
3  using System.Collections.Generic;
4
5  // This is actually OUTSIDE of the Utils Class
6  public enum BoundsTest {
7      center,          // Is the center of the GameObject on screen
8      onScreen,       // Are the bounds entirely on screen
9      offScreen      // Are the bounds entirely off screen
10 }
11
12 public class Utils : MonoBehaviour {
13
14
15 //===== Bounds Functions =====\|
16
17 // Creates bounds that encapsulate of the two Bounds passed in.
18 public static Bounds BoundsUnion( Bounds b0, Bounds b1 ) {
19     // If the size of one of the bounds is Vector3.zero, ignore that one
20     if ( b0.size==Vector3.zero && b1.size!=Vector3.zero ) {
21         return( b1 );
22     } else if ( b0.size!=Vector3.zero && b1.size==Vector3.zero ) {
23         return( b0 );
24     } else if ( b0.size==Vector3.zero && b1.size==Vector3.zero ) {
25         return( b0 );
26     }
27     // Stretch b0 to include the b1.min and b1.max
28     b0.Encapsulate(b1.min);
29     b0.Encapsulate(b1.max);
30     return( b0 );
31 }
32
33 public static Bounds CombineBoundsOfChildren(GameObject go) {
34     // Create an empty Bounds b
35     Bounds b = new Bounds(Vector3.zero, Vector3.zero);
36     // If this GameObject has a Renderer Component...
37     if (go.GetComponent<Renderer>() != null) {
38         // Expand b to contain the Renderer's Bounds
39         b = BoundsUnion(b, go.GetComponent<Renderer>().bounds);
40     }
41     // If this GameObject has a Collider Component...
42     if (go.GetComponent<Collider>() != null) {
43         // Expand b to contain the Collider's Bounds
44         b = BoundsUnion(b, go.GetComponent<Collider>().bounds);
45     }
46     // Iterate through each child of this gameObject.transform
47     foreach( Transform t in go.transform ) {
48         // Expand b to contain their Bounds as well
49         b = BoundsUnion( b, CombineBoundsOfChildren( t.gameObject ) );
50     }
51
52     return( b );
53 }
54
55 // Make a static read-only public property camBounds
56 static public Bounds camBounds {
57     get {
58         // if _camBounds hasn't been set yet
59         if ( _camBounds.size == Vector3.zero ) {
60             // SetCameraBounds using the default Camera
61             SetCameraBounds();
62         }
63         return( _camBounds );
64     }
65 }
```

```

65     }
66     // This is the private static field that camBounds uses
67     static private Bounds _camBounds;
68
69     public static void SetCameraBounds(Camera cam=null) {
70         // If no Camera was passed in, use the main Camera
71         if (cam == null) cam = Camera.main;
72         // This makes a couple important assumptions about the camera!:
73         // 1. The camera is Orthographic
74         // 2. The camera is at a rotation of R:[0,0,0]
75
76         // Make Vector3s at the topLeft and bottomRight of the Screen coords
77         Vector3 topLeft = new Vector3( 0, 0, 0 );
78         Vector3 bottomRight = new Vector3( Screen.width, Screen.height, 0 );
79
80         // Convert these to world coordinates
81         Vector3 boundTLN = cam.ScreenToWorldPoint( topLeft );
82         Vector3 boundBRF = cam.ScreenToWorldPoint( bottomRight );
83
84         // Adjust the z to be at the near and far Camera clipping planes
85         boundTLN.z += cam.nearClipPlane;
86         boundBRF.z += cam.farClipPlane;
87
88         // Find the center of the Bounds
89         Vector3 center = (boundTLN + boundBRF)/2f;
90         _camBounds = new Bounds( center, Vector3.zero );
91         // Expand _camBounds to encapsulate the extents.
92         _camBounds.Encapsulate( boundTLN );
93         _camBounds.Encapsulate( boundBRF );
94     }
95
96
97
98     // Test to see whether Bounds are on screen.
99     public static Vector3 ScreenBoundsCheck(Bounds bnd, BoundsTest test =
100     ↳BoundsTest.center) {
101         // Call the more generic BoundsInBoundsCheck with camBounds as bigB
102         return( BoundsInBoundsCheck( camBounds, bnd, test ) );
103     }
104
105     // Tests to see whether lilB is inside bigB
106     public static Vector3 BoundsInBoundsCheck( Bounds bigB, Bounds lilB, BoundsTest test
107     ↳= BoundsTest.onScreen ) {
108         // Get the center of lilB
109         Vector3 pos = lilB.center;
110
111         // Initialize the offset at [0,0,0]
112         Vector3 off = Vector3.zero;
113
114         switch (test) {
115             // The center test determines what off (offset) would have to be applied to lilB to move
116             ↳its center back inside bigB
117             case BoundsTest.center:
118                 // if the center is contained, return Vector3.zero
119                 if ( bigB.Contains( pos ) ) {
120                     return( Vector3.zero );
121                 }
122                 // if not contained, find the offset
123                 if (pos.x > bigB.max.x) {
124                     off.x = pos.x - bigB.max.x;
125                 } else if (pos.x < bigB.min.x) {
126                     off.x = pos.x - bigB.min.x;
127                 }
128                 if (pos.y > bigB.max.y) {
129                     off.y = pos.y - bigB.max.y;
130                 } else if (pos.y < bigB.min.y) {
131                     off.y = pos.y - bigB.min.y;

```

```

129     }
130     if (pos.z > bigB.max.z) {
131         off.z = pos.z - bigB.max.z;
132     } else if (pos.z < bigB.min.z) {
133         off.z = pos.z - bigB.min.z;
134     }
135     return( off );
136
137     // The onScreen test determines what off would have to be applied to keep all of lilB
138     -inside bigB
139     case BoundsTest.onScreen:
140         // find whether bigB contains all of lilB
141         if ( bigB.Contains( lilB.min ) && bigB.Contains( lilB.max ) ) {
142             return( Vector3.zero );
143         }
144         // if not, find the offset
145         if (lilB.max.x > bigB.max.x) {
146             off.x = lilB.max.x - bigB.max.x;
147         } else if (lilB.min.x < bigB.min.x) {
148             off.x = lilB.min.x - bigB.min.x;
149         }
150         if (lilB.max.y > bigB.max.y) {
151             off.y = lilB.max.y - bigB.max.y;
152         } else if (lilB.min.y < bigB.min.y) {
153             off.y = lilB.min.y - bigB.min.y;
154         }
155         if (lilB.max.z > bigB.max.z) {
156             off.z = lilB.max.z - bigB.max.z;
157         } else if (lilB.min.z < bigB.min.z) {
158             off.z = lilB.min.z - bigB.min.z;
159         }
160         return( off );
161
162     // The offScreen test determines what off would need to be applied to move any tiny part
163     -of lilB inside of bigB
164     case BoundsTest.offScreen:
165         // find whether bigB contains any of lilB
166         bool cMin = bigB.Contains( lilB.min );
167         bool cMax = bigB.Contains( lilB.max );
168         if ( cMin || cMax ) {
169             return( Vector3.zero );
170         }
171         // if not, find the offset
172         if (lilB.min.x > bigB.max.x) {
173             off.x = lilB.min.x - bigB.max.x;
174         } else if (lilB.max.x < bigB.min.x) {
175             off.x = lilB.max.x - bigB.min.x;
176         }
177         if (lilB.min.y > bigB.max.y) {
178             off.y = lilB.min.y - bigB.max.y;
179         } else if (lilB.max.y < bigB.min.y) {
180             off.y = lilB.max.y - bigB.min.y;
181         }
182         if (lilB.min.z > bigB.max.z) {
183             off.z = lilB.min.z - bigB.max.z;
184         } else if (lilB.max.z < bigB.min.z) {
185             off.z = lilB.max.z - bigB.min.z;
186         }
187         return( off );
188     }
189     return( Vector3.zero );
190 }
191
192

```

```

193 //===== Transform Functions =====\
194
195 // This function will iteratively climb up the transform.parent tree
196 // until it either finds a parent with a tag != "Untagged" or no parent
197 public static GameObject FindTaggedParent(GameObject go) {
198     // If this gameObject has a tag
199     if (go.tag != "Untagged") {
200         // then return this gameObject
201         return(go);
202     }
203     // If there is no parent of this Transform
204     if (go.transform.parent == null) {
205         // We've reached the end of the line with no interesting tag
206         // So return null
207         return( null );
208     }
209     // Otherwise, recursively climb up the tree
210     return( FindTaggedParent( go.transform.parent.gameObject ) );
211 }
212 // This version of the function handles things if a Transform is passed in
213 public static GameObject FindTaggedParent(Transform t) {
214     return( FindTaggedParent( t.gameObject ) );
215 }
216
217
218
219
220 //===== Materials Functions =====
221
222 // Returns a list of all Materials in this GameObject or its children
223 static public Material[] GetAllMaterials( GameObject go ) {
224     List<Material> mats = new List<Material>();
225     if (go.GetComponent<Renderer>() != null) {
226         mats.Add(go.GetComponent<Renderer>().material);
227     }
228     foreach( Transform t in go.transform ) {
229         mats.AddRange( GetAllMaterials( t.gameObject ) );
230     }
231     return( mats.ToArray() );
232 }
233
234
235
236
237 //===== Linear Interpolation =====
238
239 // The standard Vector Lerp functions in Unity don't allow for extrapolation
240 // (which is input u values <0 or >1), so we need to write our own functions
241 static public Vector3 Lerp (Vector3 vFrom, Vector3 vTo, float u) {
242     Vector3 res = (1-u)*vFrom + u*vTo;
243     return( res );
244 }
245 // The same function for Vector2
246 static public Vector2 Lerp (Vector2 vFrom, Vector2 vTo, float u) {
247     Vector2 res = (1-u)*vFrom + u*vTo;
248     return( res );
249 }
250 // The same function for float
251 static public float Lerp (float vFrom, float vTo, float u) {
252     float res = (1-u)*vFrom + u*vTo;
253     return( res );
254 }
255
256

```

```

257 //===== Bézier Curves =====
258
259 // While most Bézier curves are 3 or 4 points, it is possible to have
260 // any number of points using this recursive function
261 // This uses the Utils.Lerp function because it needs to allow extrapolation
262 static public Vector3 Bezier( float u, List<Vector3> vList ) {
263     // If there is only one element in vList, return it
264     if (vList.Count == 1) {
265         return( vList[0] );
266     }
267     // Otherwise, create vListR, which is all but the 0th element of vList
268     // e.g. if vList = [0,1,2,3,4] then vListR = [1,2,3,4]
269     List<Vector3> vListR = vList.GetRange(1, vList.Count-1);
270     // Remove the last element of vList, leaving one fewer
271     // e.g. if vList = [0,1,2,3,4] then vList = [0,1,2,3]
272     vList.RemoveAt(vList.Count-1);
273     // The result is the Lerp of these two shorter Lists
274     Vector3 res = Lerp( Bezier(u, vList), Bezier(u, vListR), u );
275     return( res );
276 }
277
278 // This version allows an Array or a series of Vector3s as input
279 static public Vector3 Bezier( float u, params Vector3[] vecs ) {
280     return( Bezier( u, new List<Vector3>(vecs) ) );
281 }
282
283
284 // The same two functions for Vector2
285 static public Vector2 Bezier( float u, List<Vector2> vList ) {
286     // If there is only one element in vList, return it
287     if (vList.Count == 1) {
288         return( vList[0] );
289     }
290     // Otherwise, create vListR, which is all but the 0th element of vList
291     // e.g. if vList = [0,1,2,3,4] then vListR = [1,2,3,4]
292     List<Vector2> vListR = vList.GetRange(1, vList.Count-1);
293     // Remove the last element of vList, leaving one fewer
294     // e.g. if vList = [0,1,2,3,4] then vList = [0,1,2,3]
295     vList.RemoveAt(vList.Count-1);
296     // The result is the Lerp of these two shorter Lists
297     Vector2 res = Lerp( Bezier(u, vList), Bezier(u, vListR), u );
298     return( res );
299 }
300
301 // This version allows an Array or a series of Vector2s as input
302 static public Vector2 Bezier( float u, params Vector2[] vecs ) {
303     return( Bezier( u, new List<Vector2>(vecs) ) );
304 }
305
306 }
307
308
309
310
311
312
313 //===== Easing Classes =====
314 [System.Serializable]
315 public class EasingCachedCurve {
316     public List<string> curves = new List<string>();
317     public List<float> mods = new List<float>();
318 }
319
320

```

```

321 public class Easing {
322     static public string Linear =      ",Linear|";
323     static public string In =         ",In|";
324     static public string Out =        ",Out|";
325     static public string InOut =      ",InOut|";
326     static public string Sin =        ",Sin|";
327     static public string SinIn =      ",SinIn|";
328     static public string SinOut =     ",SinOut|";
329
330     static public Dictionary<string,EasingCachedCurve> cache;
331     // This is a cache for the information contained in the complex strings
332     // that can be passed into the Ease function. The parsing of these
333     // strings is most of the effort of the Ease function, so each time one
334     // is parsed, the result is stored in the cache to be recalled much
335     // faster than a parse would take.
336     // Need to be careful of memory leaks, which could be a problem if several
337     // million unique easing parameters are called
338
339     static public float Ease( float u, params string[] curveParams ) {
340         // Set up the cache for curves
341         if (cache == null) {
342             cache = new Dictionary<string, EasingCachedCurve>();
343         }
344
345         float u2 = u;
346         foreach ( string curve in curveParams ) {
347             // Check to see if this curve is already cached
348             if (!cache.ContainsKey(curve)) {
349                 // If not, parse and cache it
350                 EaseParse(curve);
351             }
352             // Call the cached curve
353             u2 = EaseP( u2, cache[curve] );
354         }
355         return( u2 );
356     }
357
358     static private void EaseParse( string curveIn ) {
359         EasingCachedCurve ecc = new EasingCachedCurve();
360         // It's possible to pass in several comma-separated curves
361         string[] curves = curveIn.Split(',');
362         foreach (string curve in curves) {
363             if (curve == "") continue;
364             // Split each curve on | to find curve and mod
365             string[] curveA = curve.Split('|');
366             ecc.curves.Add(curveA[0]);
367             if (curveA.Length == 1 || curveA[1] == "") {
368                 ecc.mods.Add(float.NaN);
369             } else {
370                 float parseRes;
371                 if ( float.TryParse(curveA[1], out parseRes) ) {
372                     ecc.mods.Add( parseRes );
373                 } else {
374                     ecc.mods.Add( float.NaN );
375                 }
376             }
377         }
378         cache.Add(curveIn, ecc);
379     }
380
381     static public float Ease( float u, string curve, float mod ) {
382         return( EaseP( u, curve, mod ) );
383     }
384 }

```

```

385 static private float EaseP( float u, EasingCachedCurve ec ) {
386     float u2 = u;
387     for (int i=0; i<ec.curves.Count; i++) {
388         u2 = EaseP( u2, ec.curves[i], ec.mods[i] );
389     }
390     return( u2 );
391 }
392
393 static private float EaseP( float u, string curve, float mod ) {
394     float u2 = u;
395
396     switch (curve) {
397     case "In":
398         if (float.IsNaN(mod)) mod = 2;
399         u2 = Mathf.Pow(u, mod);
400         break;
401
402     case "Out":
403         if (float.IsNaN(mod)) mod = 2;
404         u2 = 1 - Mathf.Pow( 1-u, mod );
405         break;
406
407     case "InOut":
408         if (float.IsNaN(mod)) mod = 2;
409         if ( u <= 0.5f ) {
410             u2 = 0.5f * Mathf.Pow( u*2, mod );
411         } else {
412             u2 = 0.5f + 0.5f * ( 1 - Mathf.Pow( 1-(2*(u-0.5f)), mod ) );
413         }
414         break;
415
416     case "Sin":
417         if (float.IsNaN(mod)) mod = 0.15f;
418         u2 = u + mod * Mathf.Sin( 2*Mathf.PI*u );
419         break;
420
421     case "SinIn":
422         // mod is ignored for SinIn
423         u2 = 1 - Mathf.Cos( u * Mathf.PI * 0.5f );
424         break;
425
426     case "SinOut":
427         // mod is ignored for SinOut
428         u2 = Mathf.Sin( u * Mathf.PI * 0.5f );
429         break;
430
431     case "Linear":
432     default:
433         // u2 already equals u
434         break;
435     }
436
437     return( u2 );
438 }
439
440 }

```



## Weapon.cs

```
1  using UnityEngine;
2  using System.Collections;
3
4  // This is an enum of the various possible weapon types
5  // It also includes a "shield" type to allow a shield power-up
6  // Items marked [NI] below are Not Implemented in the book
7  public enum WeaponType {
8      none,          // The default / no weapon
9      blaster,       // A simple blaster
10     spread,        // Two shots simultaneously
11     phaser,        // Shots that move in waves [NI]
12     missile,       // Homing missiles [NI]
13     laser,         // Damage over time [NI]
14     shield         // Raise shieldLevel
15 }
16
17 // The WeaponDefinition class allows you to set the properties
18 // of a specific weapon in the Inspector. Main has an array
19 // of WeaponDefinitions that makes this possible.
20 [System.Serializable]
21 // System.Serializable tells Unity to try to view WeaponDefinition
22 // in the Inspector pane. It doesn't work for everything, but it
23 // will work for simple classes like this!
24 public class WeaponDefinition {
25     public WeaponType type = WeaponType.none;
26     public string letter; // The letter to show on the power-up
27     public Color color = Color.white; // Color of Collar & power-up
28     public GameObject projectilePrefab; // Prefab for projectiles
29     public Color projectileColor = Color.white;
30     public float damageOnHit = 0; // Amount of damage caused
31     public float continuousDamage = 0; // Damage per second (Laser)
32     public float delayBetweenShots = 0;
33     public float velocity = 20; // Speed of projectiles
34 }
35
36 // Note: Weapon prefabs, colors, etc. are set in the class Main.
37
38 public class Weapon : MonoBehaviour {
39     static public Transform PROJECTILE_ANCHOR;
40
41     public bool _____;
42     [SerializeField]
43     private WeaponType _type = WeaponType.blaster;
44     public WeaponDefinition def;
45     public GameObject collar;
46     public float lastShot;
47
48     void Awake() {
49         collar = transform.Find("Collar").gameObject;
50     }
51
52     void Start() {
53         // Call SetType() properly for the default _type
54         SetType( _type );
55         if (PROJECTILE_ANCHOR == null) {
56             GameObject go = new GameObject("_Projectile_Anchor");
57             PROJECTILE_ANCHOR = go.transform;
58         }
59         // Find the fireDelegate of the parent
60         GameObject parentGO = transform.parent.gameObject;
61         if (parentGO.tag == "Hero") {
62             Hero.S.fireDelegate += Fire;
63         }
64     }
65 }
```

```

65
66 public WeaponType type {
67     get { return( _type ); }
68     set { SetType( value ); }
69 }
70
71 public void SetType( WeaponType wt ) {
72     _type = wt;
73     if (type == WeaponType.none) {
74         this.gameObject.SetActive(false);
75         return;
76     } else {
77         this.gameObject.SetActive(true);
78     }
79     def = Main.GetWeaponDefinition(_type);
80     collar.GetComponent<Renderer>().material.color = def.color;
81     lastShot = 0; // You can always fire immediately after _type is set.
82 }
83
84 public void Fire() {
85     // If this.gameObject is inactive, return
86     if (!gameObject.activeInHierarchy) return;
87     // If it hasn't been enough time between shots, return
88     if (Time.time - lastShot < def.delayBetweenShots) {
89         return;
90     }
91     Projectile p;
92     switch (type) {
93     case WeaponType.blaster:
94         p = MakeProjectile();
95         p.GetComponent<Rigidbody>().velocity = Vector3.up * def.velocity;
96         break;
97
98     case WeaponType.spread:
99         p = MakeProjectile();
100        p.GetComponent<Rigidbody>().velocity = Vector3.up * def.velocity;
101        p = MakeProjectile();
102        p.GetComponent<Rigidbody>().velocity = new Vector3( -.2f, 0.9f, 0 ) *
            ↳def.velocity;
103        p = MakeProjectile();
104        p.GetComponent<Rigidbody>().velocity = new Vector3( .2f, 0.9f, 0 ) *
            ↳def.velocity;
105        break;
106
107     }
108 }
109
110
111 public Projectile MakeProjectile() {
112     GameObject go = Instantiate( def.projectilePrefab ) as GameObject;
113     if ( transform.parent.gameObject.tag == "Hero" ) {
114         go.tag = "ProjectileHero";
115         go.layer = LayerMask.NameToLayer("ProjectileHero");
116     } else {
117         go.tag = "ProjectileEnemy";
118         go.layer = LayerMask.NameToLayer("ProjectileEnemy");
119     }
120     go.transform.position = collar.transform.position;
121     go.transform.parent = PROJECTILE_ANCHOR;
122     Projectile p = go.GetComponent<Projectile>();
123     p.type = type;
124     lastShot = Time.time;
125     return( p );
126 }
127 }
128 }

```