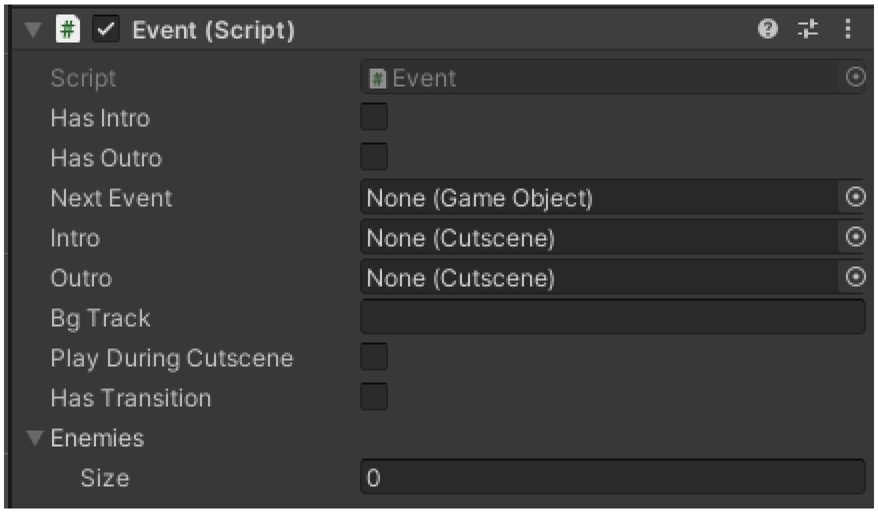
Developer Manual:

You can find scripts within the file Assets/Scripts. Scripts are split up into folders depending on what they are related to. Each one contains a short description at the top of each file. The game takes place in one Unity scene. The game has multiple components that combine together to make it easier for developers to add new content. These components are split like this.

Event	An event is a grouping of enemies and paths, an event may also have an intro cutscene or outro cutscene, as well as a different background music track. Events also have links to the next event in the sequence.
Enemy	Enemies have health as well as bullet patterns and movement paths. Enemies can have multiple movement paths and patterns that are change depending on the damage the enemy has taken.
Enemy Path	An enemy path is an object that contains nodes representing points on the path. The enemy will move towards each node at a pre-determined speed. The enemy may also be set to decelerate after each node.
Enemy Pattern	Enemy patterns are scripts to give each enemy a unique attack. Enemy pattern is an abstract class which gives you default methods and variables to get you started. Each bullet in a pattern can be fired at a specific velocity, direction, and acceleration based on a timeline of keyframes.
Pattern Sequence	If an enemy has multiple different patterns that it uses, a pattern sequence lets you decide the order in which these patterns occur, and at what health will they start at. NOTE: patterns are ordered by how they appear in the inspector.
Ability	An attack from the player. These have cooldowns and are often pretty complex in comparison. When the appropriate button is pressed, the Abilities fire method will be executed.
Cutscene	Cutscene is the entire cutscene as a whole, either played at the beginning or end of an event.
Message	Message is each individual text of a cutscene. This also includes the name of the sprites being used, the names on the nameplate, as well as who is currently speaking. It also has links to the next message in the cutscene.

Adding Events

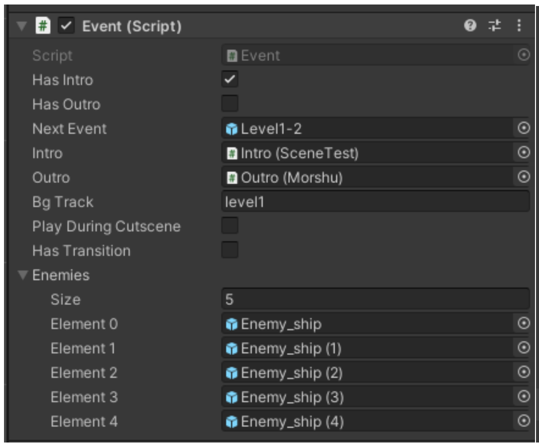
Each event is just an empty game object with the “Event.cs” script attached, and any child object (enemies and paths usually) are part of the event. When an event is finished, the current event and any children are destroyed, and the next event is loaded.



Event (Script)

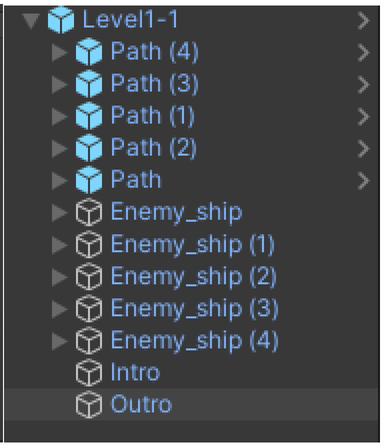
Script	Event
Has Intro	☐
Has Outro	☐
Next Event	None (Game Object)
Intro	None (Cutscene)
Outro	None (Cutscene)
Bg Track	
Play During Cutscene	☐
Has Transition	☐
Enemies	
Size	0

Above is the inspector for an event. Has intro and has outro are marked for whether an event has one. Next event is the next event loaded when this is over. Intro and outro are links to the cutscene objects that hold the dialogue data for the cutscene. BgTrack is the name of the track playing during the event. Play during cutscene is checked if you want the music to play while in the cutscene, and has transition marks whether there is a fade in or out of the event after it finishes. Enemies is a list of all the enemies in the event, this must be populated by the player themselves as they add enemies.

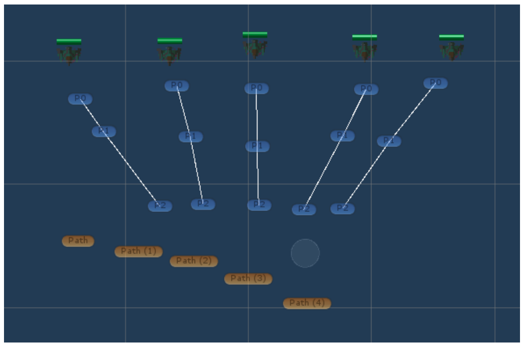


Event (Script)

Script	Event
Has Intro	✓
Has Outro	☐
Next Event	Level1-2
Intro	Intro (SceneTest)
Outro	Outro (Morshu)
Bg Track	level1
Play During Cutscene	☐
Has Transition	☐
Enemies	
Size	5
Element 0	Enemy_ship
Element 1	Enemy_ship (1)
Element 2	Enemy_ship (2)
Element 3	Enemy_ship (3)
Element 4	Enemy_ship (4)

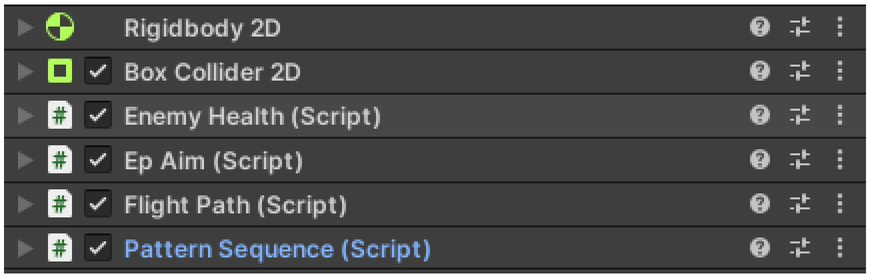


```
Level1-1
  Path (4)
  Path (3)
  Path (1)
  Path (2)
  Path
  Enemy_ship
  Enemy_ship (1)
  Enemy_ship (2)
  Enemy_ship (3)
  Enemy_ship (4)
  Intro
  Outro
```



This is what a typical event will look like. Note that cutscenes must be held in empty objects. Events are saved as prefabs within unity. This is necessary to be able to load the next event.

Adding Enemies



Rigidbody 2D
Box Collider 2D
Enemy Health (Script)
Ep Aim (Script)
Flight Path (Script)
Pattern Sequence (Script)

An enemy will typically have these scripts on them. Rigidbody 2d and box collider 2d are unity scripts. Enemy health will hold how much health they have left, Ep Aim is the one (and only) shooting pattern of this specific enemy. Flight path is the script that lets the enemy change paths, and lets you specify which path it goes along. Pattern sequence lets you write at what points the enemy bullet pattern will change. NOTE: in order for this to work, the patterns must be part of the inspector as components. They are loaded in order that they appear, top to bottom. Enemies are saved as prefabs. It is highly recommended to use the Enemy_Ship prefab as a base for an enemy. This can be found in Assets/Prefabs/Enemy_Ship.

Adding enemy paths

Each path will have a path head with the script "MovementPath.cs" Movement path holds an array of each node in the path. Nodes are just empty game objects that represent where each point in the path is. Paths go into the Flight Path script of enemies.

Adding Enemy Patterns

Enemy patterns are individual scripts that derive from the EnemyPattern.cs class. The enemy pattern class has the abstract method Fire() which needs to be added to all children of this class. It also has Begin(), which is called at the start of the pattern's period, determined by the "pattern sequence" script in enemy. And End is called when it finishes. Begin() typically invokes Fire() repeatedly. SingleFire() fires a singular bullet in a direction, at a velocity. A bullet can have a "timeline" and "speeds", both arrays of floats, passed to it through SingleFire(). These two arrays hold the "keyframes" of speed the bullet will take. Between points in "timeline" the bullet will gradually change speed to match that of the next speed in the timeline. The first speed is determined by the initial speed. And the final speed will be a constant.

Adding Abilities

Abilities are individual scripts that derive from the Ability.cs class. Each ability has a Begin() and End() method which is called when the user presses a key or lets go of a key respectively. They also have a cooldown and cooldownTimer variables, to represent it's cooldown. Abilities are pretty open when it comes to what they do, as all methods are virtual. As of right now, abilities need to be added to the “fire trigger” script directly so that they’re called when the user presses the button. However, it is planned to let the user select their own abilities.

Adding Cutscenes.

```
public class SceneTest : Cutscene
{
    public override DialogueTree dS { get; protected set; }

    private void Start()
    {
        dS = new DialogueTree();
        dS.Add(new Message("Lorem ipsum dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor",
            "Cherry", "",
            Message.DialogueSprite.Cherry, Message.DialogueSprite.NegativeCherry));
        dS.Add(new Message(
            "incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
quis nostrud exercitation", "",
            "Negative Cherry",
            Message.DialogueSprite.Cherry, Message.DialogueSprite.NegativeCherryMad,
false));
        dS.Add(new Message("ullamco laboris nisi ut aliquip ex ea commodo consequat.",
"Cherry", "",
            Message.DialogueSprite.CherryMad, Message.DialogueSprite.NegativeCherry));
        dS.Add(new Message("Duis aute irure dolor in reprehenderit in voluptate velit
esse cillum", "",
            "Negative Cherry",
            Message.DialogueSprite.Cherry, Message.DialogueSprite.NegativeCherry,
false));
        dS.Add(new Message("dolore eu fugiat nulla pariatur.", "Cherry", "",
            Message.DialogueSprite.CherryMad, Message.DialogueSprite.NegativeCherry));
        dS.Add(new Message(
            "Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia
deserunt mollit anim id est laborum.",
            "", "Negative Cherry",
            Message.DialogueSprite.Cherry, Message.DialogueSprite.NegativeCherryMad,
false));
    }
}
```

Dialogue trees are relatively simple to create. First you create an empty dialogueTree object. Then you add messages to it. The constructor for a message looks like this.

```
public Message(string body, string lName, string rName, DialogueSprite lSprite,
DialogueSprite rSprite,
    bool leftSpeaking = true)
```

Body is the main part of the message. Lname and rname refer to the nameplates. lSprite and rSprite refer to the sprites being used, and leftSpeaking refers to whether it’s the person on the left speaking or the person on the right.