New System: Lasers and Mirrors

The way it works:

**Core components:**

* The terminal (Script: LaserTerminal.cs) awaits input from the player. If the player is close, on button press (‘space’ by default), the terminal calls the LoadBeam() method from a laser caster it has been assigned (on the inspector);
* The laser caster (Script: LaserCaster.cs) receives input from the terminal and begins a number of loading sound effects (4 repetitions by default) and spawns wave object with a merely aesthetic purpose. After all repetitions are done, it instantiates a Laser Beam that inherits several properties from the script, including speed and duration (note, the duration of the beam is currently given by a percentage [105%] of the duration of the sound that plays when the beam is shot);
* The instantiated laser beam (Script: LaserBeam.cs) receives an initial force based on the speed it inherited from the Laser Caster, bounces off mirrors, and stops on collision with non-mirror objects. The puzzle is a success if the collider is tagged as “Receiver”, or a failure otherwise. The laser:
  + Exists on the scene for a set duration (life) inherited from the caster;
  + Generates a particle trail whose width changes over time to simulate light;
  + Instantiates Laser Particle Generators on collision with mirror objects, which inherit specific duration values based on a timer, so they match the duration of the trail;

**Mirror Components:**

* The mirrors (Script: Mirror.cs) are specially tagged object the Laser Beam is expected to bounce off of. The script they contain serves two purposes:
  + Regulate their interaction with Elizabeth’s Wind Spell;
  + Store their original position and the method to restore if a pressure plate is activated;

The mirrors’ reaction to the Wind Spell is almost identical to that of enemies. However, they mitigate some of the push projected onto them to make them move more precisely (and realistically\*);

Note: Mirrors do not collide with Walls right now, since this functionality is not part of the Wind Spell;

* The Mirror Controller (Script: MirrorController.cs) is a parent object to which all the mirrors of a given puzzle should be children. When a pressure plate is pressed, the controller will iterate through a list containing references to each of its children, and call the reset() method on them, which will restore their position. It also calls the FadeIn() and FadeOut() functions from the Transition prefab (see below) to make the transition smoother.

**Additional components:**

* Laser Waves (Script: LaserWave.cs) are instantiated by the Caster and Receiver objects for aesthetic purposes. Their scale increases towards a set amount, its alpha gets reduced toward zero, and the object is destroyed;
* Laser Particles (Script: LaserParticles.cs), or rather particle generator objects, are instantiated by the Laser Beam when it collides with mirrors. They exist for a set amount meant to match the exact moment the nearest trail particle disappears (the duration of the generator is obtained from the Laser Beam). The object stops emitting slightly before the timer ends, and waits until all particles have effectively faded before destroying itself;
* The Receiver (Script: Receiver), like the mirror, is specially tagged to incite a specific response from the Laser Beam. The Receiver scripts serves merely to instantiate Laser Waves and play a sound effect on beam collision for aesthetic purposes;

**Code Updates Not Specific to the New System:**

* AudioControl.cs (Audio Manager): The PlaySFX() method now returns the duration of the sound effect being played, to allow for sound effect iteration.
* Transition.cs: Two new methods included, FadeIn() and FadeOut(), to fade the screen to black and back in, respectively. Can be called from anywhere as long as a reference to the Transition prefab is obtained [using GameObject.Find(“Transition”)];

Note: The Transition object was added to the UI prefab, so it is now a global component that can be accessed from every room. This means two things:

* + A fade-in transition will occur every time a new level is loaded.
  + Crossfade animations can be called by any object that has a reference to the prefab.

**Assets**

Sprites:

* Five terminal sprites to reflect the On, Off, In-Progress, Success, and Failure states.
* Placeholder sprite for the laser caster.
* Placeholder sprite for the mirrors (please replace this).
* Placeholder sprite for the receiver.
* Placeholder sprite for the laser wave.

Art Components:

* Particle generator for laser beams.
* Particle generator for mirror collisions.
* Trail renderer for the laser beam.

Materials:

* Bouncy physics material for the laser beam.
* Simple red-line material for the laser beam trail.

Audio:

* Three sound effects for terminal interactions (On, Success, Failure).
* Three sound effects for the laser mechanics (Loading Shot on the Caster, Receiving Energy on the Receiver, and Shooting the Laser).

Additional Changes:

Changed the location of the files in the Carlos scene folder to the corresponding project folder. Updated the name of the ‘tranMode.cs’ script to ‘TranMode.cs’ to obey naming practices. Added the new sound effects to the AudioManager.

**Things To Do When Implementing/Integrating the Changes:**

- Two objects need external references to work, and they must be assigned manually from the inspector:

* Every Terminal must be connected to a Laser Caster (drag and drop the **Editor Instance** of the caster into the Laser Caster variable field on the inspector).
* Every Laser Caster needs a reference to a Terminal (drag and drop the **Editor Instance** of the terminal into the Parent Terminal variable field on the inspector).

- The position the camera will follow when the terminal is interact with must also be **set manually**, by moving the empty **CameraTarget** object (included in the Terminal prefab) to the location you’d like to focus the camera on. Make sure you are moving an **Instance** of the CameraTarget object on the scene editor, **NOT the prefab one**, doing this will displace **ALL** instances of CameraTarget.

- All mirrors should be listed as children of a Mirror Controller object.

- If there is more than one Virtual Cinemachine Camera on the scene, or if the Virtual Camera has been renamed, you must specify the name of the Virtual Camera you want to transition on the Virtual Camera Name variable field of the terminal. Otherwise, it will be set to the default Virtual Camera name (CM vcam1).

- To set up a **Pressure Plate** for a puzzle, reference the corresponding **Mirror Controller** object and call this instance method: **pressPlate()**

**Other Technical Information about the Mirror/Laser System**

- There are no restrictions to the number of mirrors and receivers a puzzle can have, but every puzzle must have a single instance of a laser caster, a mirror controller, and a terminal.

- There is no limit to the number of puzzles you can have in a room. They will operate independently from each other. I recommend creating a named Empty Parent Object to organize the Instances of each puzzle.

- There are no restrictions to the position/rotation of any of the parts of the mirror system. The caster will cast the Laser Beam in the direction of the red markings (which signifies the front face of the caster) and the Beam will bounce off the environment as expected, using the built-in physics engine.

- The beam will only last for a certain amount of time. If it must travel for a time longer than 6.1 seconds (approximately), the laser will be destroyed and return a failed attempt. The trail will also thin out and disappear after 6.1 seconds, regardless of whether it was consumed or not by then.

- Though the speed of the Laser Beam can be adjusted on the Laser Caster instance, massive speeds will make the Beam clip through mirrors and walls, whereas low speeds risk a higher chance of ending the puzzle unpleasantly.

- Player Input is not yet restricted while the system is active. The player can collide with the Beam and break the puzzle while it’s running but, after 6.2 seconds or so, it should reset without consequences and return a failed attempt.

A build showcasing the new features and their intended implementation can be found in Carlos’ scene folder: MirrorTesting.unity