# **Classes**

## **Cage Configuration Setup**

### **CageConfigDropdown : MonoBehaviour**

* Handles the user inputs in the cage configuration setup menu

Properties

* ValueChanged: UnityEvent
  + A UnityEvent allowing other classes to run code when the dropdown menu is changed
  + See [UnityEvent.AddListener()](https://docs.unity3d.com/ScriptReference/Events.UnityEvent.AddListener.html) for how to listen for changes in the dropdown menu

Methods

* GetCurrentConfig(): CageConfiguration
  + Returns current state of drone cage

### 

### **Saver : MonoBehaviour**

* Manages saving cage configurations for future runs of the application

Methods

* static ForceSave(): void
  + Will save the data in the SaveState to the computer
* GetState(): SaveState
  + Returns a reference to the SaveState

### 

### **SaveState**

* Stores data in a manner allowing it to be saved to the computer for future runs of the application

Properties

* ConfigPrefabs: List<CageConfiguration>
  + List of cage configurations to save between runs of the application

### 

### **CageConfiguration (struct)**

* Holds the data necessary to construct a drone cage in the GUI

Properties

* configName: string
  + The name assigned to a cage
* cageWidth: float
  + The width of a cage
* cageLength: float
  + The length of a cage. The length spans the blue team area and the red team area
* sectorLength: float
  + The length of a red team sector (aligned with the length of the cage)
* sectorCountX: uint
  + The number of sectors along the length of the cage
* sectorCountY: uint
  + The number of sectors along the width of the cage

### 

### **Ruler : MonoBehaviour**

* Manages and maintains a ruler graphic to the left and below the drone cage visual

### 

### **CageRenderer : MonoBehaviour**

* Displays cage visual on screen.
* Provides mathematical support for finding coordinates on the screen

Properties

* MAX\_X: float
  + The largest possible value of x the right side of the cage can reach on the current screen
* MIN\_X: float
  + The smallest possible value of x the left side of the cage can reach on the current screen
* MAX\_Y: float
  + The largest possible value of y the top side of the cage can reach on the current screen
* MIN\_Y: float
  + The smallest possible value of y the bottom side of the cage can reach on the current screen
* left: float
  + The actual x position of the left side of the drone cage on the screen
* right: float
  + The actual x position of the right side of the drone cage on the screen
* up: float
  + The actual y position of the top side of the drone cage on the screen
* down: float
  + The actual y position of the bottom side of the drone cage on the screen

Methods

* FeetToPixels(float feet): float
  + Converts a given value in feet to Unity units (not pixels as the name implies) by applying the scale of the drone cage visual

### 

### **QuitListener : MonoBehaviour**

* Listens for ESC key press then quits the application

## 

## **User Input Validation**

### **SensorDropdown : MonoBehaviour**

* Manages the Sensor Selection Dropdown and edge cases related to it

Mathods

* GetID(): uint
  + Returns the sensor id of the currently selected sensor
* AddNew(uint id): void
  + Adds the sensor corresponding to the given id to the dropdown menu (unless it already exists) and sets dropdown menu to be at this option
* DeleteID(uint id): void
  + Deletes the option in the dropdown menu if it exists that corresponds with the sensor with the matching id

### 

### **AddingButton : MonoBehaviour**

* Adds a number (positive or negative) to a specific input field when clicked by the user

### 

### **ConfirmButton : MonoBehaviour**

* Loads the Ping Display scene when the Confirm button is clicked

### 

### **abstract Constrainer : MonoBehaviour**

* Class that allows for code to run when an input field is modified in order to restrict the data in the input field to a set format

### 

### **FloatConstrainer : Constrainer**

* Constrains an input field’s data to be in a format that can be converted to the float data type

### 

### **HAngleConstrainer : FloatConstrainer**

* Constrains the horizontal rotation input field’s data to be in a format that can be converted to the float data type while remaining within the [0, 360) range

### 

### **VAngleConstrainer : FloatConstrainer**

* Constrains the vertical rotation input field’s data to be in a format that can be converted to the float data type while remaining within the [-90, 90] range

### 

### **PosFloatConstrainer : FloatConstrainer**

* Constrains an input field’s data to be in a format that can be converted to the float data type while being positive

### 

### **XConstrainer : PosFloatConstrainer**

* Constrains the sensor x input field’s data to be in a format that can be converted to the float data type while being positive and not being larger than the length of the red team area of the drone cage in feet

### 

### **YConstrainer : PosFloatConstrainer**

* Constrains the sensor y input field’s data to be in a format that can be converted to the float data type while being positive and not being larger than the width of the drone cage in feet

### 

### **FloatParser : MonoBehaviour**

* Allows for reading of data in an input field into a float

Methods

* ReadField(): float
  + Reads the data in the input field as a float

### 

### **IntParser : MonoBehaviour**

* Allows for reading of data in an input field into an integer

Methods

* ReadField(): int
  + Reads the data in the input field as an integer

### 

### **TypeParser : MonoBehaviour**

* Allows for reading and setting the selected type of the Sensor Type dropdown menu

Methods

* ReadValue(): string
  + Return the text of the currently selected sensor type dropdown menu option
* SetValue(string text): void
  + Set the currently selected sensor type dropdown menu option to match the given text (assuming the given text is an existing sensor type dropdown menu option)
* SetValue(uint index): void
  + Set the currently selected sensor type dropdown menu option with the given index for the list of options

### 

### **VerticalRotationPreview : MonoBehaviour**

* Displays an indicator on screen corresponding with the vertical rotation input field

## 

## **Sensor Management**

### 

### **SensorRenderer : MonoBehaviour**

Properties

* Config: SensorConfiguration
  + The configuration of the sensor the renderer is displaying

Methods

* UpdateConfig(SensorConfiguration newConfig): void
  + Gives the renderer a new config (or a changed version of an old config) and tells it to display it’s sensor visual to match

### 

### **SensorConfiguration**

Properties

* type: string
  + The type of sensor being represented
* id: uint
  + The id of the sensor being represented
* x: float
  + The position of the sensor being represented in feet along the length of the drone cage from the left side of the red team’s area
* y: float
  + The position of the sensor being represented in feet along the width of the drone cage from the bottom side of the cage
* hRotation: float
  + The rotation of the sensor being represented horizontally in degrees where 0 degrees is facing towards the right side of the drone cage and rotating counter-clockwise increases the angle
* vRotation: float
  + The rotation of the sensor being represented vertically in degrees where 0 degrees is pointing straight up, 90 degrees is facing parallel to the floor, etc.

### 

### **SensorConfigReader : MonoBehaviour**

* Interface for reading data from user input

Methods

* ReadSensor(): SensorConfiguration
  + Reads the user input fields and dropdowns into a SensorConfiguration object

### 

### **SensorManager : MonoBehaviour**

Methods

* SaveSensor(uint id, SensorConfiguration newConfig): void
  + Saving a sensor displays it visually with a specific id at a location constructed from a SensorConfiguration. Saving a sensor with an id another sensor was already saved with will overwrite the older sensor
* DeleteSensor(uint id): void
  + Deletes the sensor matching the given id if it exists.
* FindSensor(uint id): SensorRenderer
  + Returns the SenserRenderer (visual representation of sensor) corresponding with the given id if it exists. Will return null if no SensorRenderer matching the id was found
* UpdateSensors(): void
  + Causes all visual sensors (SensorRenderers) to recalculate their positions/rotation/etc. on screen without changing their configuration. Useful when cage dimensions change.
* GetSensors(): List<SensorRenderer>
  + Returns a list of all SensorRenderers (visual representations of sensors) currently being managed.

### 

### **SensorPreview : MonoBehaviour**

* Manages the preview display of the sensor currently being modified. This allows a real-time visual to update as the user changes the data making up the sensor’s position, rotation, and type

Methods

* ResetPreview()
  + Removes the sensor preview visual until user modifies inputs again

### 

### **SensorDeleter : MonoBehaviour**

* Deletes the currently selected sensor when the Delete button is clicked by the user

### 

### **SensorSaver : MonoBehaviour**

* Saves the current user input as a sensor when the Save button is clicked by the user

## 

## **Ping Displaying**

**COMSelector : MonoBehaviour**

* Attached to same object as ArduinoListener
* Reads number key input and attempts connection to an Arduino Coordinator on the corresponding COM port

Properties

* DefaultCom: int
* Selects the default com port to attempt connection with

### **ArduinoListener : MonoBehaviour**

* Reads input from the Coordinator Arduino and sends packets to corresponding sensor types

### 

### **Jammer : MonoBehaviour**

* Handles Jam packets by causing large amounts of false positive pings on the GUI

Methods

* DisplayJammed(): void
  + Creates large amounts of false positive pings when called

### 

### **Sensor : MonoBehaviour**

* Parent class for all sensor types which handle displaying pings for different types of sensors
* See SensorTemplate.cs for a template for adding custom sensors to the GUI

Properties

* Cage: CageRenderer (write only)
  + Reference to the CageRenderer for calculations. Typically set when the Sensor is created
* PingTemplate: GameObject (write only)
  + Template GameObject for pings. Typically set when the Sensor is created
* Manager: SensorManager (write only)
  + Manager of the sensors being displayed. Typically set when the Sensor is created
* Type: uint (read only)
  + Id for the type of the sensor. For individual sensor types, expect this number to match the number in the first byte of ping packets from this type of sensor

Methods

* PlotPing(Ping ping): void
  + Displays the provided ping object on the screen
* Jam(): void
  + Generates fake pings for all sensors matching the type

### 

### **LIDAR : Sensor**

* Handles calculation of where on the screen to display pings from LIDAR sensors

### 

### **Ultrasonic : LIDAR**

* Handles calculation of where on the screen to display pings from Ultrasonic sensors
* LIDAR and Ultrasonic pings are calculated the same way so Ultrasonic uses the same code as LIDAR through inheritance

### 

### **OMNISonic : Sensor**

* Handles calculation of where on the screen to display pings from OMNISonic sensors

### 

### **PingFader : MonoBehaviour**

* Fades ping images out over time and eventually deletes them

### 

### **Ping**

* Stores data from a ping packet

Constructors

* Ping(string packet)
  + Creates a Ping object out of a packet received from the Arduino

Properties

* type: uint (read only)
  + Number corresponding to the type of sensor the ping is from.
* id: uint (read only)
  + Number corresponding to the id of the sensor the ping is from
* data: List<string> (read only)
  + List of all data entries extracted from the packet received from the Arduino (except the type and id)

### 

### **PingTextFader : MonoBehaviour**

* Fades ping images out over time and eventually deletes them

# **Scene Hierarchy**

## **CageConfig**

* Saver (has Saver.cs attached. Manages saving cage configurations between runs of the application)
* Canvas (manages UI objects)
  + Side Panel (the visual of the panel on the side)
  + Main Panel (visual panel covering the entire screen in the background
  + Dropdown (has CageConfigDropdown.cs attached. Manages the dropdown for selecting cage configurations)
  + EditToggleButton (button that toggles the editability of user inputs when clicked)
  + Name Label (text saying “Name:”)
  + Name Input Field (input field for inputting the name of a cage configuration)
  + Cage Length Label (text saying “Cage Length:”)
  + Cage Length Input Field (input field for inputting the cage length of a cage)
  + Cage Width Label (text saying “Cage Width:”)
  + Cage Width Input Field (input field for inputting the cage width of a cage)
  + Sector Length Label (text saying “Sector Length:”)
  + Sector Length Input Field (input field for inputting the length of sectors in a cage configuration)
  + Sector Count X Label (text saying “Sector Count X:”)
  + Sector Count X Input Field (input field for inputting the number of sectors along the length of the cage)
  + Sector Count Y Label (text saying “Sector Count Y:”)
  + Sector Count Y Input Field (input field for inputting the number of sectors along the width of the cage)
  + Save Button (button that causes the current cage configuration to be saved)
  + Delete Button (button that causes the current cage configuration to be deleted)
  + Select Button (button that proceeds to the SensorPlacement scene with the current cage configuration selected)
* Drone Cage (has CageRenderer.cs attached. manages visual of the drone cage)
  + Vertical Sector Edge (template for the lines separating sectors in the cage visual)
  + Cage Edge (template for the lines forming the edges of the cage visual)
  + Drone Starting Area (template object for the blue colored area in the cage visual)
* Ruler (has Ruler.cs attached. manages the ruler visuals on the sides of the drone cage. will persist between scene changes)
  + Directional Light (necessary to brighten ruler visual elements)
  + RulerMark (enabled by scripts. template visual component representing a single ruler mark)
  + BottomRulerMain (visual component of the main part of the bottom ruler)
  + LeftRulerMain (visual component of the main part of the left ruler)
* QuitListener (has QuitListener.cs attached. will persist between scene changes. listens for ESC to be pressed then quits the application)

## **SensorPlacement**

* Drone Cage (has CageRenderer.cs attached. manages visual of the drone cage)
  + Vertical Sector Edge (template for the lines separating sectors in the cage visual)
  + Cage Edge (template for the lines forming the edges of the cage visual)
  + Drone Starting Area (template object for the blue colored area in the cage visual)
* Canvas (manages UI objects)
  + Side Panel (the visual of the panel on the side)
  + Main Panel (visual panel covering the entire screen in the background
  + Dropdown (has SensorDropdown.cs attached. handles switching sensors via the dropdown menu)
  + Save Button (has SensorSave.cs attached. handles a button that saves a sensor when clicked)
  + Delete Button (has SensorDeleter.cs attached. manages a button that deletes a sensor when clicked)
  + Confirm Button (has ConfirmButton.cs attached. manages a button that swaps scenes to SensorReadings with saved sensors when clicked)
  + ID Label (text saying “ID:”)
  + ID InputField (has IDConstrainer.cs and IntParser.cs attached. an input field for inputting the id of a sensor)
  + Sensor X Label (text saying “Sensor X:”)
  + Sensor X Input Field (has XConstrainer.cs and FloatParser.cs attached. an input field for inputting the X coordinate of a sensor)
  + Sensor Y Label (text saying “Sensor Y:”)
  + Sensor Y Input Field (has YConstrainer.cs and FloatParser.cs attached. an input field for inputting the Y coordinate of a sensor)
  + Horizontal Rotation Label (text saying “Horizontal Rotation:”)
  + Horizontal Rotation Input Field (has HAngleConstrainer.cs and FloatParser.cs attached. an input field for inputting the horizontal rotation of a sensor)
  + Vertical Rotation Label (text saying “Vertical Rotation:”)
  + Vertical Rotation Input Field (has VAngleConstrainer.cs and FloatParser.cs attached. an input field for inputting the vertical rotation of a sensor)
  + -X Button (has AddingButton.cs attached. a button that decreases the value in Sensor X Input Field when clicked)
  + +X Button (has AddingButton.cs attached. a button that increases the value in Sensor X Input Field when clicked)
  + -Y Button (has AddingButton.cs attached. a button that decreases the value in Sensor Y Input Field when clicked)
  + +Y Button (has AddingButton.cs attached. a button that increases the value in Sensor X Input Field when clicked)
  + -HRotation Button (has AddingButton.cs attached. a button that decreases the value in Horizontal Rotation Input Field when clicked)
  + +HRotation Button (has AddingButton.cs attached. a button that increases the value in Horizontal Rotation Input Field when clicked)
  + -VRotation Button (has AddingButton.cs attached. a button that decreases the value in Vertical Rotation Input Field when clicked)
  + Type Label (text saying “Sensor Type:”)
  + Sensor Type Dropdown (has TypeParser.cs attached. a dropdown menu for selecting a sensor type)
* Sensors (has SensorManager.cs attached. will persist between scene changes. manages sensors on screen)
  + Sensor (has SensorRenderer.cs attached. enabled by scripts. a single sensor visual)
* SensorPreview (has SensorPreview.cs attached. Manages the preview visual on screen)
* Vertical Rotation Indicator (has VerticalRotationPreview.cs attached. displays an indicator of the vertical rotation of the selected sensor)
  + Arrow (the arrow that is pointed by the Vertical Rotation Indicator)

## **SensorReadings**

* Drone Cage
  + Vertical Sector Edge (template for the lines separating sectors in the cage visual)
  + Cage Edge (template for the lines forming the edges of the cage visual)
  + Drone Starting Area (template object for the blue colored area in the cage visual)
* PingManager (has ArduinoListener.cs and COMSelector.cs attached. displays pings corresponding to incoming packets from the coordinator Arduino)
  + Ping (has PingFader.cs attached. is enabled by scripts. a visual element of a sensor ping)

# Asset Hierarchy

* Resources
  + Images
    - Arrow.png
    - LIDAR.png
    - OMNISonic.png
    - ping.png
    - Protractor.png
    - Ultrasonic.png
  + Materials
    - CageEdge.mat
    - RulerMaterial.mat
    - SectorEdge.mat
* Scenes
  + CageConfig.unity
  + SensorPlacement.unity
  + SensorReadings.unity
* Scripts
  + Cage Configuration
    - CageConfigDropdown.cs
    - CageRenderer.cs
    - ConfigStorage.cs
    - QuitListener.cs
    - Ruler.cs
    - Saver.cs
    - SaveState.cs
  + Ping Reading
    - ArduinoListener.cs
    - LIDAR.cs
    - OMNISonic.cs
    - Ping.cs
    - PingFader.cs
    - Sensor.cs
    - SensorTemplate.cs
    - Ultrasonic.cs
  + Sensor Placement
    - Sensor Management
      * SensorConfigReader.cs
      * SensorConfiguration.cs
      * SensorDeleter.cs
      * SensorManager.cs
      * SensorPreview.cs
      * SensorSaver.cs
    - User Input
      * AddingButton.cs
      * ConfirmButton.cs
      * Constrainer.cs
      * FloatConstrainer.cs
      * FloatParser.cs
      * HAngleConstrainer.cs
      * IDConstrainer.cs
      * IntParser.cs
      * PosFloatConstrainer.cs
      * SensorDropdown.cs
      * TypeParser.cs
      * VAngleConstrainer.cs
      * XConstrainer.cs
      * YConstrainer.cs
    - Visual Elements
      * SensorRenderer.cs
      * VerticalRotationPreview.cs