IPA Firefighter Data Visualizer

System Specification

Version 1.0

Zeeshan Karim

Overview:

The IPA Firefighter Data Visualizer is a cross platform standalone application designed to interface with the IPA data collection and processing backend. It is developed in Unity 5.5 and scripted in C#. The application currently supports Windows, Mac, Linux, and Android devices at any resolution in landscape. The application is designed to be intuitive, and provide robust functionality in a simple, mobile inspired design.

Features:

* Visualizer: Primary functionality of the application. Generates and places firefighter and beacon objects populated by incoming messages from the server. Positions and scales GameObjects and Main Camera to accurately position objects over a digital map. Provides functionality for zoom, data overview, object focus, and a terminal window displaying relevant system information. Users can select from three map styles. Data stream from server displays firefighter locations in real-time.

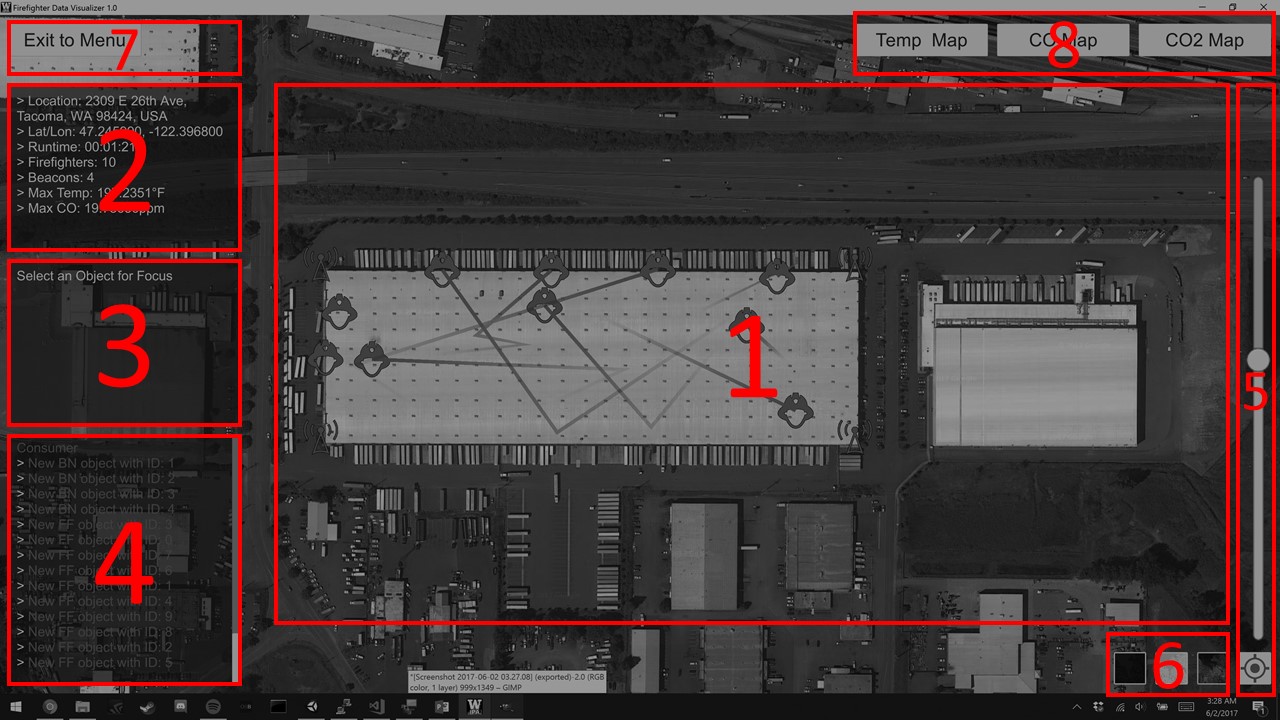


Fig. 1: Visualizer layout diagram

1. Map panel: Provides primary data visualization functionality. Implements a complex object positioning and scaling system to accurately represent GameObjects positions on a zoom enabled map.
   1. Map: Utilizes Infinity Code’s Online Maps API (Available on the Unity Asset Store) to fetch map tiles from various online maps API’s including Google Classic, Google Satellite, and Mapbox Classic Dark. Maps API manages a 4096x4096 texture, updated with map tiles selected using a center latitude/longitude, and zoom level.
   2. Firefighter GameObjects: GameObject Prefab including a 2D firefighter sprite, with a randomly selected color upon instantiation. Includes a line renderer, showing a trail of the path taken by the object over time. Contains a script containing the firefighter’s data as determined by the server, and provides functionality for instantiating a target icon and displaying additional data when the object is clicked.
   3. FF Spawner: Manages firefighter model and instantiates/destroys firefighter GameObjects based on incoming messages from an instruction queue managed by the network controller. Messages addressing existing objects update their positions and metadata. Capable of managing any number of objects dynamically. Implements the automatic positioning functionality of the application, by averaging the position of all objects in the model to update a global center position in Latitude/Longitude coordinates.
   4. Beacon GameObjects: GameObject Prefab including a 2D beacon sprite, colored yellow upon instantiation. Contains a script containing the beacon’s data as determined by the server.
   5. BN Spawner: Manages beacon model and instantiates/destroys beacon GameObjects based on incoming messages from an instruction queue managed by the network controller. Messages addressing existing objects update their positions and metadata.
2. Overview Window: Displays current overview data. Data includes:
   1. Location address: determined by reverse geocoding the center latitude longitude coordinate using Google’s maps API.
   2. Latitude/Longitude: truncated float representation of the current center map position.
   3. Runtime: time elapsed since the beginning of the current visualization.
   4. Firefighters: Total number of firefighters in the current model.
   5. Beacons: Total number of beacons in the current model.
   6. Max Temp: Maximum temperature reading across the entire model.
   7. Max CO: Max CO reading across the entire model.
3. Focus Window: Displays existing model data upon selection of an object to focus on. Only one object can be the focus subject at a time. Displayed data includes:
   1. Type: Object type
   2. ID: Object ID
   3. Lat/Lon: Position of object
   4. Temp: Temperature of object
   5. CO: CO reading of object
   6. Humidity: Humidity reading of object
   7. Light: Light reading of object
   8. REST data: Selection of a object to focus on invokes a REST API request in the format “/[object type][ID]”, eg. “/firefighter3”. This request is sent to the server. It asynchronously waits for a reply, and on return appends the additional data to the focus window following the data above.
4. Terminal Window: Displays messages regarding system information in a familiar terminal format. Caps number of displayed messages at 100, and allows users to scroll using a scroll bar. Displayed messages include:
   1. Server Connection status (Connecting, Connected, Disconnected)
   2. Instantiation of new objects
   3. REST API requests
   4. Any message sent by server using the “terminal” WebMessage type (See Data Specification)
5. Map Controls: Provides UI for controlling map zoom and position
   1. Zoom Slider: Invokes the “UpdateZoom” function in the Zoom Controller script attached to the Map Panel. Function scales model object sprites, camera size, map size, and modifies map zoom to provide smooth zoom functionality.
   2. Center Map Button: Invokes the “UpdateMap” function in the Location Controller script function attached to the Map Panel. Function recalculates the center of data in the model, centers the map and data to the updated values, and resets the zoom to max.
6. Map Style Buttons: Provides UI for selecting a map style. Functionality implemented in the UI Controls script attached to the UI GameObject.
   1. Dark Map Button: Sets map provider to MapBox and fetches tiles from a custom dark tileset designed in MapBox Classic.
   2. Light Map Button: Sets map provider to Google Maps with classic tileset.
   3. Satellite Map Button: Sets map provider to Google Maps with satellite tileset.
7. Exit to Menu Button: Allows user to return to menu scene. Functionality implemented in the UI Controls script attached to the UI GameObject.
   1. Destroys objects in model and disconnects from server
   2. Unloads visualizer scene and loads menu scene
8. Heatmap control buttons:
   1. Placeholder UI for future implementation of heatmaps

* Network Manager: Implementation of WebSocket and REST network connections. Provides application with data from server. See “Data Specification” for detailed information on data format.

1. WebSocket Network Manager: Implements the primary data connection to the server.
   1. Uses the WebSocket Sharp library available at: <https://github.com/sta/websocket-sharp>
   2. Connects to server at startup. Upon successful connection begins listening for incoming messages
   3. Upon receipt of a message, the message is unpacked into a WebMessage object. Its type is determined, and its body is placed into its respective instruction queue.
   4. Instruction types include “ff” for firefighters, “bn” for beacons, and “terminal” for terminal messages.
2. REST Network Manager: Implements additional data requests on object focus
   1. Uses the System.Net library to implement HTTP requests from the same IP being used in the WebSocket manager.
   2. Requests are placed into coroutines which allow the application to continue running while waiting for a response.
   3. Upon the return of a message, the message contents are appended to the focus window.

* Menu: Scene viewed on launch of application. Contains application title, UWT logo, dark map background, and developer name and year. Menu button toggles application settings overlay. Start visualization button loads visualizer scene.

1. Application Settings: Allows user to input application settings that persist when application is terminated using Unity PlayerPrefs.
   1. Server IP: Populates contents from existing value. On change the saved value is edited with the new value. Used by the network manager to connect to the server.
   2. Server Port: Populates contents from existing value. On change the saved value is edited with the new value. Used by the network manager to connect to the server.
   3. Firefighter Icon Size: Allows user to set scale of firefighter icons using a slider. Size is visualized with an example icon in realtime.
   4. Beacon Icon Size: Allows user to set scale of beacon icons using a slider. Size is visualized with an example icon in realtime.

Data Specification:

1. WebMessages: Data specification for server – client data
   1. WebMessages are sent by the server are formatted as serialized JSON strings.
   2. WebMessage objects contain string parameters for “type” and “body”.
   3. On receipt of a message from the server, it is unpacked using a JSON un-serializer into a WebMessage object. Messages must be serialized JSON, with these two parameters. Additional parameters are ignored.
   4. WebMessages are separated by “type” and their “body” values are placed in the respective instruction queues.
2. Acceptable values of WebMessage “type” and “body” parameters:
   1. “ff” or “FF”, denotes that message body is a firefighter object in serialized JSON format.
   2. “bn” or “BN”, denotes that message body is a beacon object in serialized JSON format.
   3. “terminal”, denotes that message body is a string to be displayed in terminal
3. Firefighter and Beacon Object Parameters:
   1. “id”: (Integer) (Required) Represents the object ID. Used to select and identify objects in the model. New ID’s instantiate a new GameObject. Existing ID’s update an existing GameObject with the new data.
   2. “lat”: (Float)(Required) Represents the latitude value of the object
   3. “lon”: (Float)(Required) Represents the longitude value of the object.
   4. “alt”: (Float)(Required) Represents the altitude value of the object.
   5. “time”: (String)(Optional) Timestamp associated with the message
   6. “temp”: (Float)(Optional) Temperature of object
   7. “humidity”: (Integer)(Optional) Humidity of object
   8. “light”: (Integer)(Optional) Light reading of object
   9. “co”: (Integer)(Optional) CO reading of object
4. Example WebMessages:
   1. { type: 'ff', body: '{"id":0,"lat":47.24568080764683,"lon":-122.39656656912942,"alt":1}' }
   2. { type: 'bn', body: '{"id":0,"lat":47.24568080764683,"lon":-122.39656656912942,"alt":1}' }
   3. { type: 'terminal', body: 'This text will appear on the application terminal' }

Software Specification:

Scripting for this application was performed in C# using Visual Studio tools for Unity. These scripts extend MonoBehavior, and all run concurrently. Most scripts implement a start and update function, that run at startup and once per frame respectively. Following are the class diagrams representing the software structure of the application.

1. Visualizer Class Diagram: Visual representation of the references between GameObjects, Scripts, and prefabs that make up the functionality of the visualizer.

