**Dialog System API**

Course: UWB - CSS452 Game Engine

Author: Peter Chen, Martin Matsudaira, Jonny Robinson

Date: 3/13/2023

## Table of Contents

[**Overview 2**](#_r8y0mq8h5d52)

[**Demo Screenshot 3**](#_moas6kjlonos)

[**Design 4**](#_xj6n32rgiyej)

[General Workflow 6](#_fx6z3t7hcklb)

[**API Usage / Files 8**](#_hblcu3un49nl)

[Dialog object JSON Files 8](#_1ls5oze2o4fw)

[src/engine/dialog.js 10](#_gx0ang2lxq2c)

[**Tutorial 14**](#_c2s2ul2lw025)

[**Instruction 28**](#_o9x36i9wxaal)

ONLINE DEMO LINK:

<https://nausniboar.github.io/nausniboar/>

## Overview

The Dialog System is essential functionality in the video game.

It provides an engaging game experience through text for storytelling and game interaction. It also helps display character in different ways. (see Figure 1)



***Figure 1*** *Fire Emblem Example*

The text also serves as the real-time subtitle so people with hearing disabilities can engage with the game; It also saves the company money from voice acting.

Although the sound is not necessary, it does provide a more satisfying gameplay experience!!!

**Our Dialog System API Support**

* Display textbox
* Update text
* Display associate sprite
* Update display sprite
* Play audio
* Skip Text
* User Define Textbox
* Adjust display speed

## Demo Screenshot

Here is the example outcome for using our API *!!!*



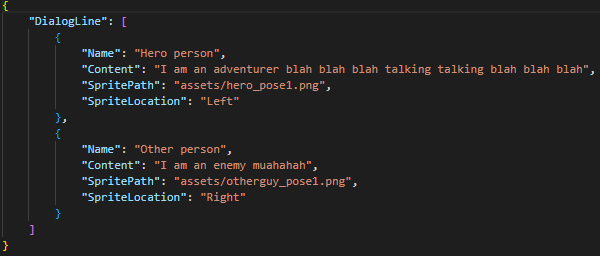


## Design

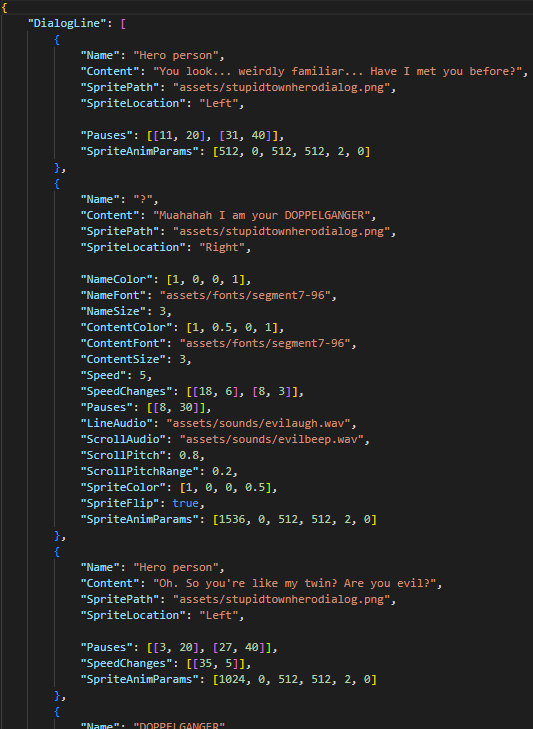
The API consists of two classes: Dialog.js + Scrolling\_Font\_Renderable.js

The API also requires a JSON file for each scene (conversation).

* JSON File
  + Contain information about the dialog display
  + The format is as follows (See Figures 2 & 3)



***Figure 2*** *Basic Dialog Information Requirement*



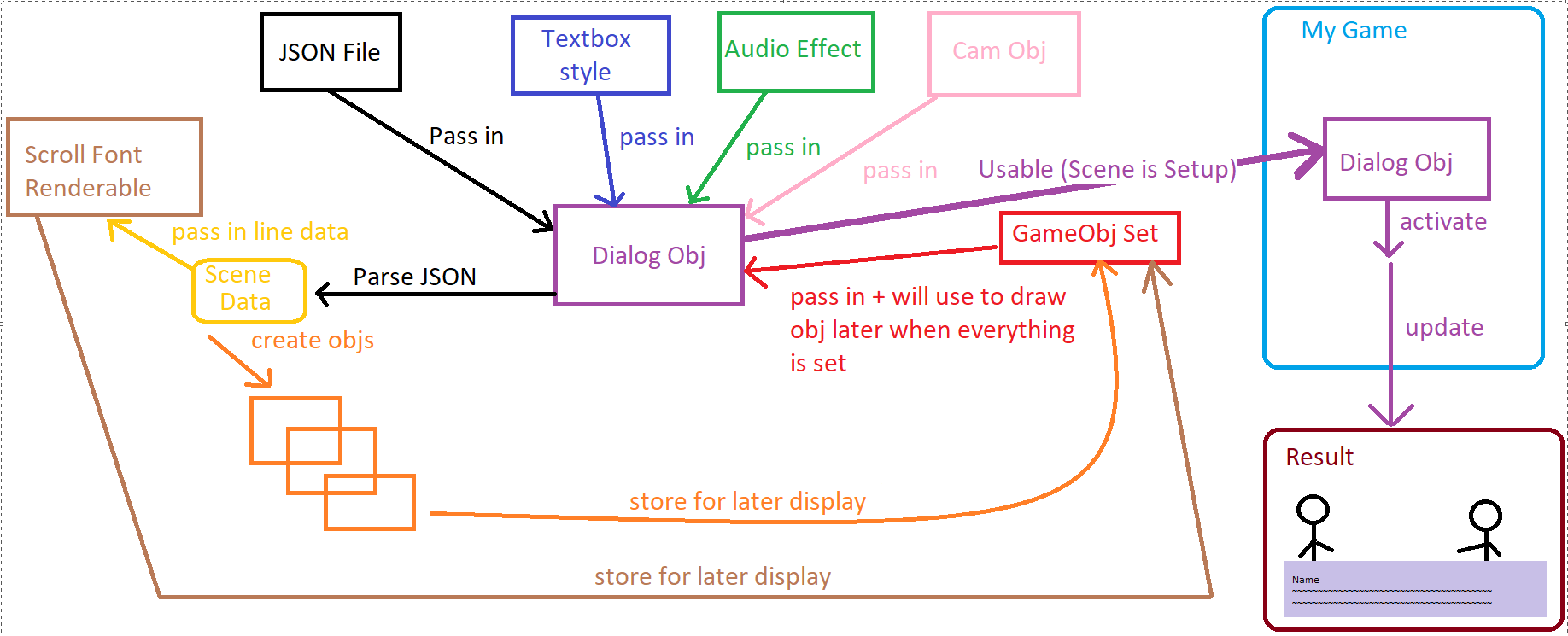
***Figure 3*** *Full Advance Dialog Information Usage*

* Dialog.js
  + Create a Dialog Obj (Setting up Scene to display)
    - Take in the JSON file → Scene info
    - Take in GameObjSet → Store all Scene Objs for drawing
    - Take in Cam Obj → where to draw
    - Take in audio Effect path → Audio Effect when displaying
    - Take in user define textbox → Textbox style for display
* Once everything is setup
  + Activate it will start the dialog system (display scene)
  + Update will update the scene display (conversation)
* Scrolling\_Font\_Renderable.js
  + Create a ScrollingFontRenderable Obj (Setup Text attribute)
    - Take in the scene text related data from Dialog obj (JSON parsed)
* Once the Scrolling\_Font\_Renderable obj finish creates text obj
  + It will store in Dialog GameObjSet for drawing text

### General Workflow

1. Creates textbox (Texture Renderable)
2. Create Dialog Obj
   1. Pass in JSON, Audio Path, Cam, textbox
   2. Parse JSON → pass text-related data to Scrolling Font Renderable Obj
   3. Finish setup Scene in this Dialog Obj
3. User call activate Dialog Obj
   1. Dialog Start Display (draw)
4. User call update Dialog Obj
   1. Dialog updates the next line scene. (draw)

**Here is the API Design Diagram (See Figure 4)**



***Figure 4*** *Dialog System Design*

## API Usage / Files

### Dialog object JSON Files

JSON files are used to store content such as text, layout, text speed, and sprite paths. These JSON files will be parsed by **“engine.json”** and used by dialog.js in the constructor.

{

"DialogLine": [

{

"Name": "Hero person 2",

"Content": "I am an adventurer blah blah blah",

"SpritePath": "assets/hero\_pose1.png",

"SpriteLocation": "Left"

},

{

"Name": "Other person 2",

"Content": "I am an enemy muahahah",

"SpritePath": "assets/otherguy\_pose1.png",

"SpriteLocation": "Right"

}

]

}

In order to create a dialog object and render text, a JSON file must be created. Each JSON file must contain data inside of it:

{

"DialogLine" : [

// Data goes here

]

}

“DialogLine” is an array that can take in multiple dictionaries that will be used in the dialog object. Each new dictionary will represent dialogue from a certain character/object. The following are the different keys that can be defined within each dictionary in the DialogLine array.

**Required *(See Figure 2)***

| Key | Value Type | Description |
| --- | --- | --- |
| Name | String | Represents the object in focus  (ex. Main character) |
| Content | String | Dialogue text to be displayed |
| SpritePath | String | Absolute path to the sprite asset to be displayed with the text box. |
| SpriteLocation | String | Relative position of the sprite on screen when displaying.  **Valid inputs:** “Right”, “Left”, null |

**Optional *(See Figure3)***

| Key | Value Type | Description |
| --- | --- | --- |
| NameColor | Float Array | A 4-digit normalized array that represents the RGBA of the objects name text color |
| NameFont | String | The absolute path to the font file that will be used for the name text. |
| NameSize | Integer | Determines the size of the name text. |
| ContentColor | Float Array | Determines the color of the content text.  **(See NameColor for use)** |
| ContentFont | String | Determines the font of the content text.  **(See NameFont for use)** |
| ContentSize | Integer | Determines the size of the content font. |
| Speed | Integer | Determines how fast the text will scroll |
| SpeedChanges | 2D Integer Array | An array containing an array of [x, y] pairs that determines that index “x” will be set to the speed of “y” |
| Key | Value Type | Description |
| Pauses | 2D Integer Array | Pauses the text with [x, y] where x is the index and y is the duration in fps |
| LineAudio | String | Absolute path to the audio file that will be played on triggering the next line. |
| ScrollAudio | String | Absolute path to the audio file that will be played upon scrolling to the next line of text. |
| ScrollPitch | Float | Determines the pitch of the scroll audio |
| ScrollPitchRange | Float | Determines the variance in pitch of the scroll audio |
| SpriteColor | Float Array | A 4-digit normalized array that represents the RGBA of the tint for the sprite being used. |
| SpriteFlip | Boolean | Determines if a sprite will be mirrored |
| SpriteAnimParams | Integer Array | The parameters for the frames in a spritesheet  **(See engine.SpriteAnimateParamRenderable)** |
| SpriteAnimPauses | Boolean | Determines if a sprite animation will end after the line is complete or paused. |
| SpriteRepeats | Boolean | Determines if the sprite animation will loop |
| Skippable | Boolean | Determines if the user can load the full line of text at once |

### src/engine/dialog.js

constructor (json, renderSet, cam, scrollBeep = null, textBoxStyle)

**Description:**

A public function that will be called to create a Dialog object using “new Dialog()”.

**Parameters:**

| Parameter | Object Type | Description |
| --- | --- | --- |
| json | json (parsed) | Contains data to be used in the dialogue |
| renderSet | GameObjectSet | Set of objects that will contain all objects of the dialogue  (ex. Text and scroll arrow) |
| cam | Camera | Camera to be displayed |
| scrollBeep | Audio | Sound that will be triggered upon animating the dialogue text |
| textBoxStyle | TextureRenderable | Object that allows custom designs for the dialogue text box |

**Returns:**

| This function does not return anything. |
| --- |

update()

**Description:**

Updates the animation of the dialogue and handles user input. This function should be called in the update function in the game loop.

**Parameters:**

| This function takes in no parameters |
| --- |

**Returns:**

| This function does not return anything. |
| --- |

nextLine()

**Description:**

This function is called to advance the dialogue to the next line of text. This function is also called in the update() function of dialog.js.

**Parameters:**

| This function takes in no parameters |
| --- |

**Returns:**

| This function does not return anything. |
| --- |

isActive()

**Description:**

This getter function returns a boolean value to check if the object being used is in an active state.

**Parameters:**

| This function takes in no parameters |
| --- |

**Returns:**

| Object Type | Description |
| --- | --- |
| Boolean | Value of this.active |

activate()

**Description:**

Sets the dialog object’s this.active to true if the dialog has not been played or is repeatable.

**Parameters:**

| This function takes in no parameters |
| --- |

**Returns:**

| This function does not return anything. |
| --- |

deactivate()

**Description:**

Sets the dialog object’s this.active to false.

**Parameters:**

| This function takes in no parameters |
| --- |

**Returns:**

| This function does not return anything. |
| --- |

setRepeatable(repeatable)

**Description:**

Sets the dialog object’s this.repeatable to indicate if the dialogue can be triggered more than once.

**Parameters:**

| Parameter | Object Type | Description |
| --- | --- | --- |
| repeatable | Boolean | Value to set this.repeatable |

**Returns:**

| This function does not return anything. |
| --- |

isReplayable()

**Description:**

Gets the boolean value of the dialog object’s this.repeatable.

**Parameters:**

| This function takes in no parameters |
| --- |

**Returns:**

| Object Type | Description |
| --- | --- |
| Boolean | Value of this.repeatable |

## Tutorial

Let us create a simple demo using our Dialog API

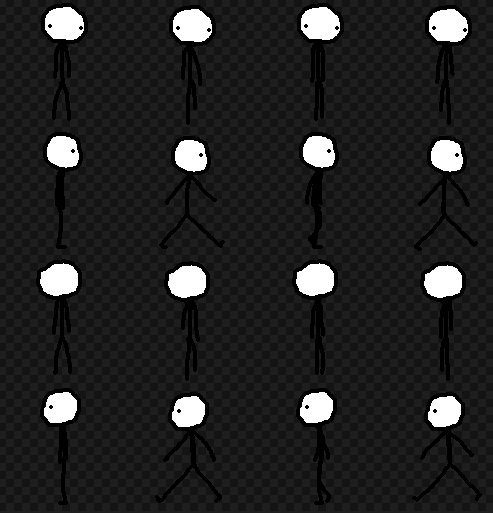
* Make sure you are using the game engine that includes bounding box collision support. (For interaction trigger)

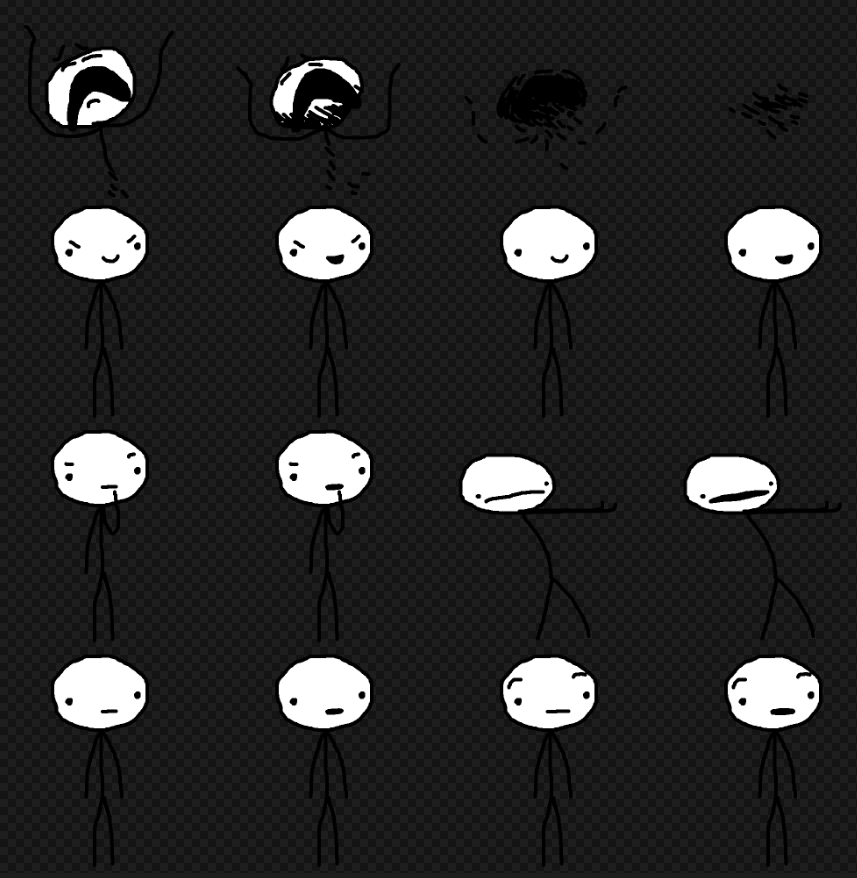
1. **Make Sure you have all the assets you need (Below are examples)**



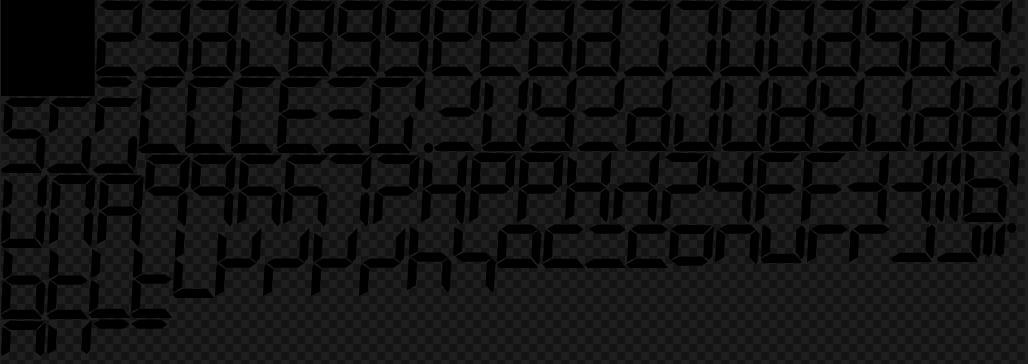










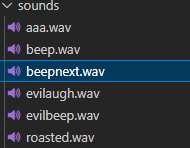


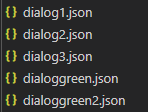


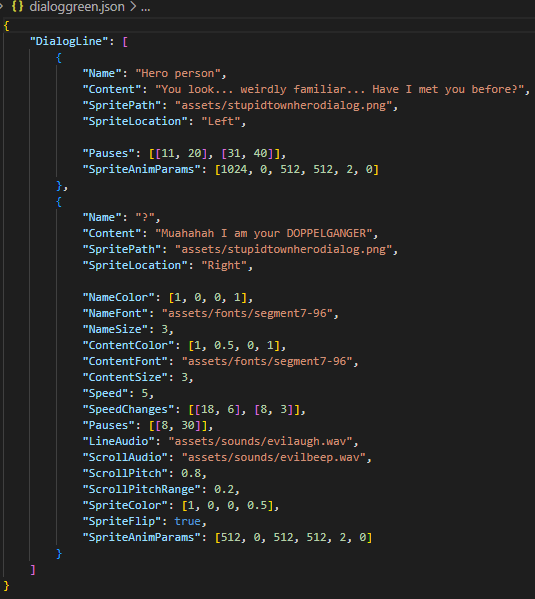








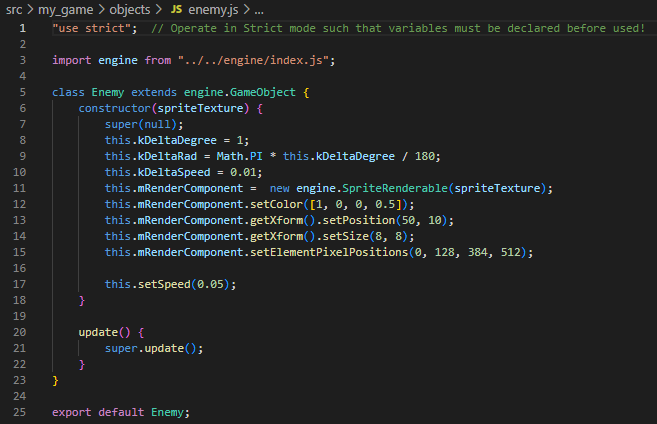




1. **Create Walking\_Enemy Object class**

****

1. **Create Enemy Object class**

****

1. **Create Hero Object class**

**"use strict"; // Operate in Strict mode such that variables must be declared before used!**

**import engine from "../../engine/index.js";**

**class Hero extends engine.GameObject {**

**constructor(spriteTexture) {**

**super(null);**

**this.kDelta = 0.3;**

**this.direction = 4;**

**this.lastDirection = 4;**

**this.walking = false;**

**this.lastWalking = false;**

**this.mRenderComponent = new engine.SpriteAnimateRenderable(spriteTexture);**

**this.mRenderComponent.setColor([1, 1, 1, 0]);**

**this.mRenderComponent.getXform().setPosition(35, 35);**

**this.mRenderComponent.getXform().setSize(8, 8);**

**this.mRenderComponent.setSpriteSequence(512, 0,**

**128, 128,**

**1,**

**0);**

**this.mRenderComponent.setAnimationType(engine.eAnimationType.eRight);**

**this.mRenderComponent.setAnimationSpeed(10);**

**}**

**update() {**

**// control by WASD**

**let xform = this.getXform();**

**let xDelta = 0;**

**let yDelta = 0;**

**if (engine.input.isKeyPressed(engine.input.keys.W)) {**

**yDelta += this.kDelta;**

**}**

**if (engine.input.isKeyPressed(engine.input.keys.S)) {**

**yDelta -= this.kDelta;**

**}**

**if (engine.input.isKeyPressed(engine.input.keys.A)) {**

**xDelta -= this.kDelta;**

**}**

**if (engine.input.isKeyPressed(engine.input.keys.D)) {**

**xDelta += this.kDelta;**

**}**

**if(xDelta > 0) {**

**this.direction = 3;**

**this.\_walk();**

**} else if(xDelta < 0) {**

**this.direction = 1;**

**this.\_walk();**

**} else if(yDelta > 0) {**

**this.direction = 2;**

**this.\_walk();**

**} else if(yDelta < 0) {**

**this.direction = 4;**

**this.\_walk();**

**} else {**

**this.\_stop();**

**}**

**xform.incYPosBy(yDelta);**

**xform.incXPosBy(xDelta);**

**this.lastDirection = this.direction;**

**this.lastWalking = this.walking;**

**this.mRenderComponent.updateAnimation();**

**}**

**\_walk() {**

**this.walking = true;**

**if(this.direction != this.lastDirection || this.walking && !this.lastWalking) {**

**this.mRenderComponent.setSpriteSequence(this.direction \* 128, 128,**

**128, 128,**

**2,**

**0);**

**}**

**}**

**\_stop() {**

**this.walking = false;**

**if(!this.walking && this.lastWalking) {**

**this.mRenderComponent.setSpriteSequence(this.direction \* 128, 0,**

**128, 128,**

**1,**

**0);**

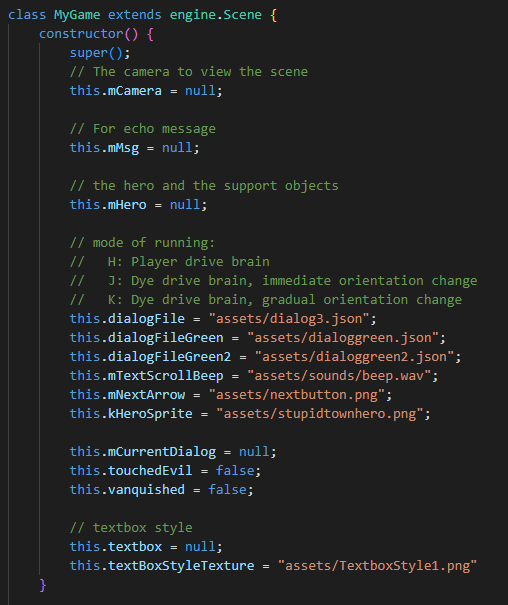
**}**

**}**

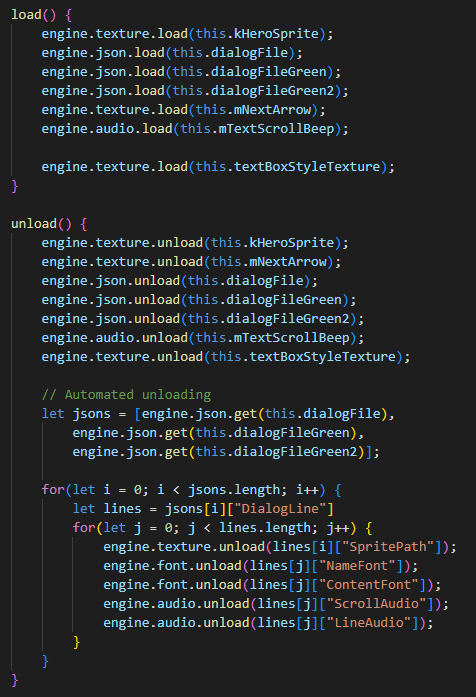
**}**

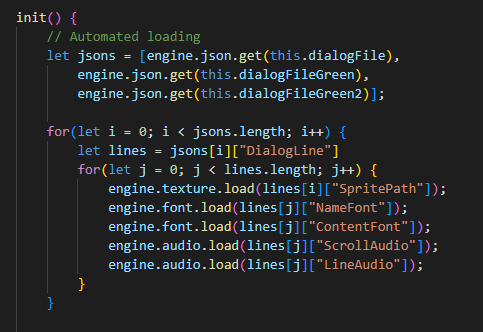
**export default Hero;**

1. **Init the Constructor**

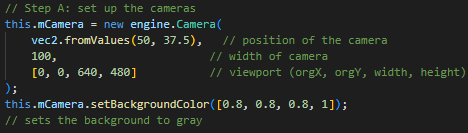


1. **Load and unLoad Asset**





1. **Initialize the My Game**
2. Set up Camera



1. Create NPC



1. Create Hero



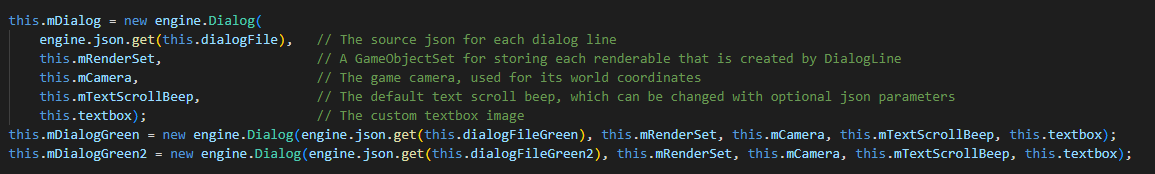
1. Create Game Object Set



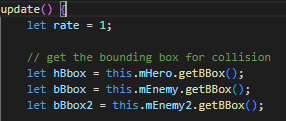
1. Create the Textbox with your selected style



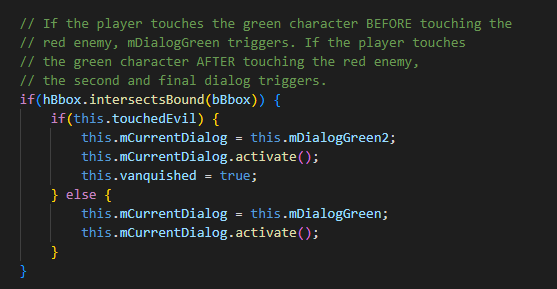
1. Create Dialog Object (Scene)



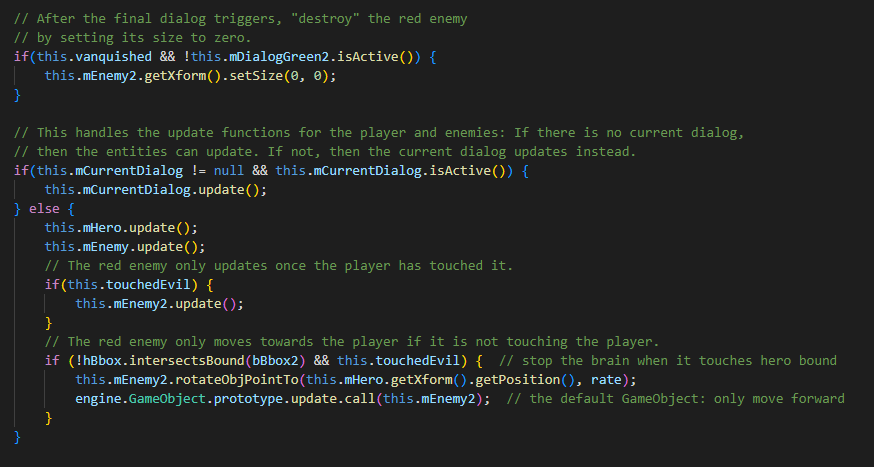
1. **Update My Game**
2. Get bounding box for NPCs + Hero



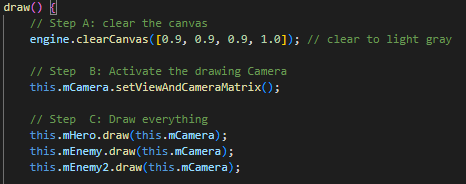
1. Create Interaction + Activate the Dialog



1. Update the Dialog when need to go to next line



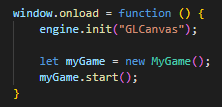
1. **Draw the Game**



1. Draw Dialog



1. **Start Your Demo Game**



**You have now successfully created your Dialog Game Demo!!!**

## Instruction

*TO RUN THE LOCAL DEMO:*

* *Navigate to html file in VS Code*
* *Click Go Live*

Let’s take a look at our more advanced online hosted demo [[Link](https://nausniboar.github.io/nausniboar/)]

**Control**

* **W A S D → Movement**
* **N → Skip & next line**

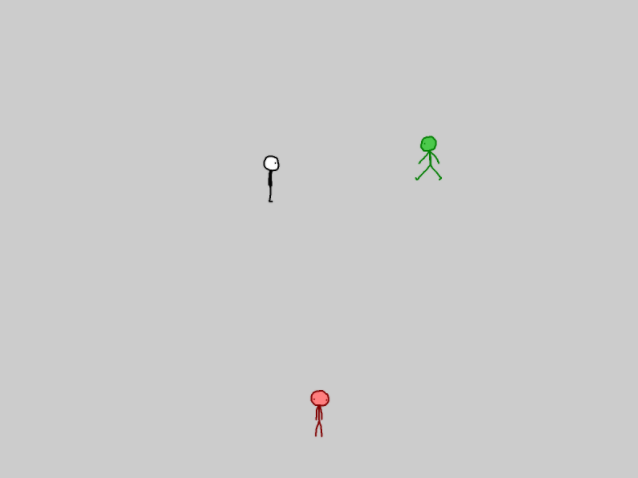
**Interaction**

* **Activate Through Bounding Box Intersect**
  + **Nearby the NPC**

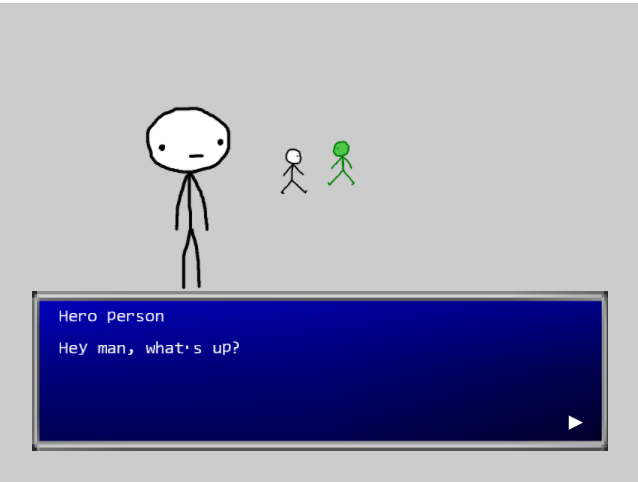
**Start Demo**

1. **Interact with the Green Man**

* **Walk toward Green Man**

****

1. **Dialog With Green Man Starts**

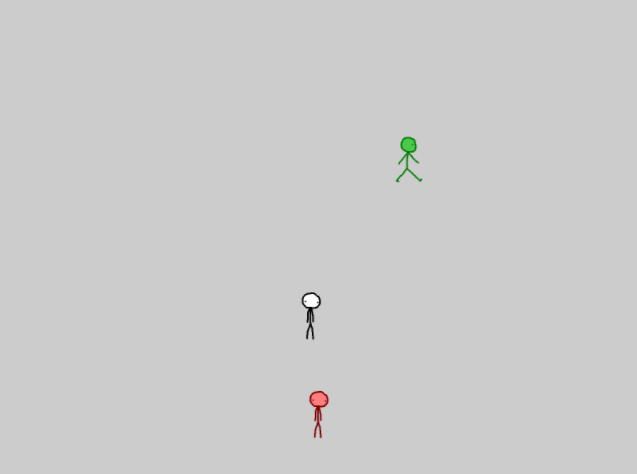
****

* **Use the “N” key to skip the line or go to the next line.**

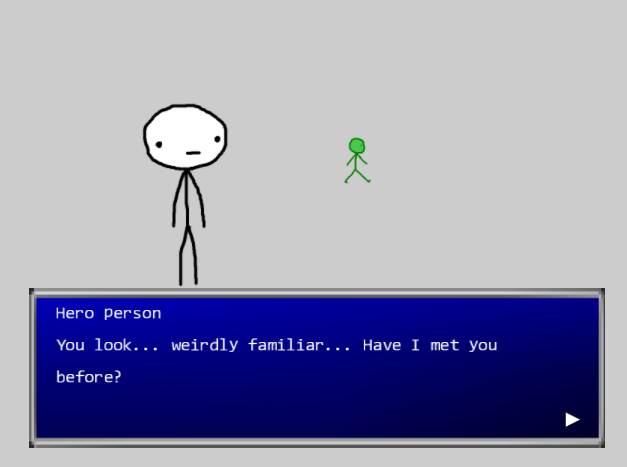
****

1. **Interact with Red Man**

* **Walk toward Red Man**

****

1. **Dialog With Red Man Starts**

****

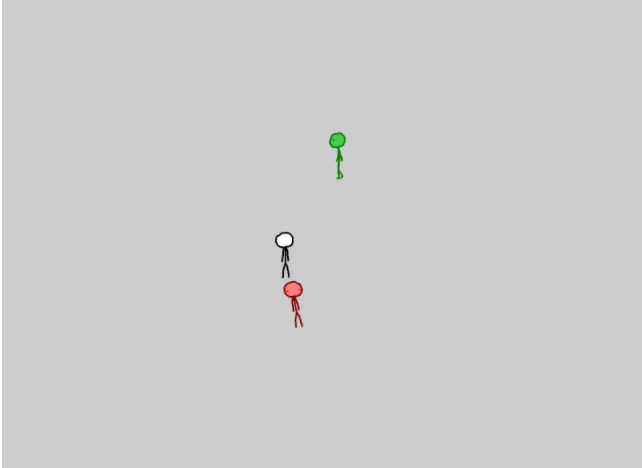
* **Use the “N” key to skip the line or go to the next line.**

****

****

1. **Seek help from Green Man**

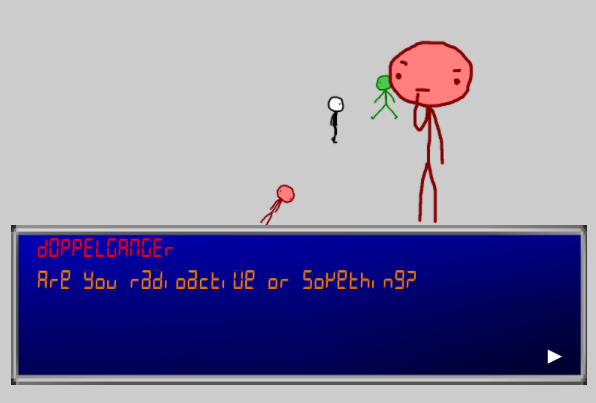
* **Walk toward Green Man**

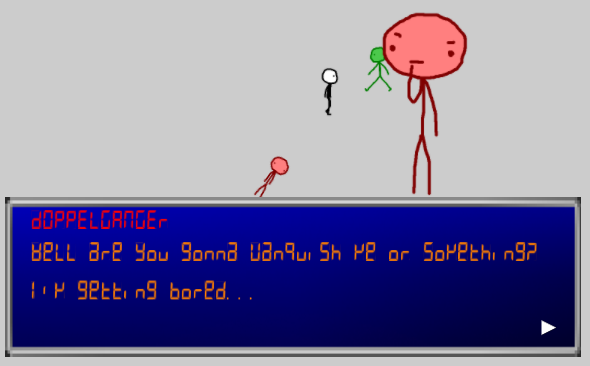
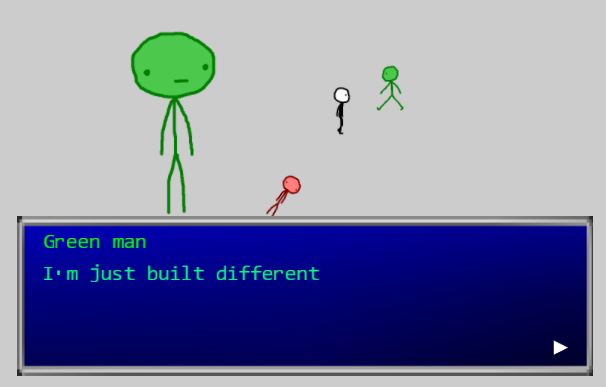
****

1. **Dialog with Green Man Starts (Help scene)**

****

****

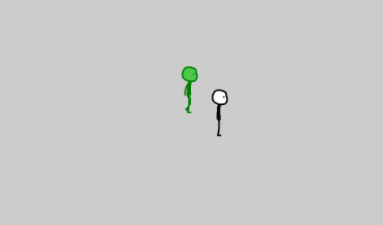
****

****

****

****

****

****

**Demo End!!!**