Alpha Double Prime Documentation

Kevin Justice- UP:10/27/16

Global Conventions:

- All variables camel case. 2 or more words can be abbreviated with lowercase letters, if flow makes sense.
- All parent object constructors should be defined like ObjectConstructor1 = function(vars){};.
- When defining a parent object base class method (In Sheet), refer to the parent object with "this".
- When a function that is intended to be a sub object for a parent object defined elsewhere, the first line in the definition should be "var scope = this;". When referring to the parent object within this function always use scope rather than this.
- When a function that is intended to be a method for a parent object defined elsewhere, the first line should be "var scope = this". Use scope to refer to the parent object.
- Json object childnames go in single quotes.
- Event names go in single quotes.
- When defining strings that will go into an HTML element, use double quotes.
- All variables declared in any scope other than global shall be defined with the var directive.
- Module files should be named like "CodeFileNumberOne.js"
- Data files should be named with camel case filenames
- Folders should be all lowercase letters
- i should be used as the index variable name for loops ☺
- Comparisons for equality should be done like, obj1 == obj2 (note the 2 eq signs !3).

==========

Server Side:

Directory:

- **/game**
  - **/data**
    - **/usr**
  - **/map**
    - **Map.js**
  - **/rooms**
    - **RoomObject.js**
    - **RoomObjectDefinitions.js**
- **/socketinterface**
  - **SocketInterface.js**
  - **SocketInterfaceFunctionSheet.js**
- **GameServer.js**

GameServer.js:

Main game server which connects to users via sockets. Sockets are interfaced and inputs and outputs are sent and received and handled with this code. All requires are initialized in this code. This code serves as the

Global scope for the server. Rooms are created and initialized here, and sockets are assigned to SocketInterfaces with the 'onconnect' listener.

Map.js:

Constructor for the server Map object.

- new Map(mapBuild): constructor
  - mapBuild (object):
    - mapBuild.x (int): # of tiles on the x axis
    - mapBuild.z (int): # of tiles on the z axis
    - mapBuild.tileArray (n x m array): Contains an array of tiles that has n and m dimensions corresponding to every tile on the world map. Inaccessible tiles should be listed in this array as null. 0 indicates the tile in (-z)(-x) corner. (-z) is considered NORTH.

- Map.getTileByID(id): returns tile object corresponding to tile with id 'id'.
  - id (int): id of tile in tileArray to be returned.

- Map.get[North, South, West, East, NorthWest, SouthWest, SouthEast, NorthEast]Of(tile): returns the tile located in the direction specified from the tile argument.
  - tile (object): tile to find direction relative to
    - tile.x (number): xposition of tile
    - tile.y (number): yposition of tile
    - tile.z (number): z position of tile
    - tile.id (int): integer tile id

RoomObject.js:

Constructor for new game room objects to be used by the server.

- new RoomObject(room, constructors): constructor
  - room (function): specific room function to attach to the new room. A new object will be created for this function and will be attached to the RoomObject as .room
  - constructors (function): function object of specific constructors that the room function uses to organize itself. A new object will be created for this function and it will be attached to the RoomObject as .constructors

- RoomObject.openRoom(): method for opening the room. Creates the room objet and attaches it to itself for call with .room. Note that rooms aren't ever closed.

RoomObjectDefinitions.js:

Function definitions for game room objects.

Naming Convention:

- rXXXXXXXY: definition for a room
- rcXXXXXXXY: definition for a rooms constructors

SocketInterface.js:

SocketInterface object constructor. Used for interfacing an io socket with rooms and handling its event listeners.

- new SocketInterface(socket, initializeFunction, destroyFunction, username): constructor
    - socket (socket): socket associated with the SocketInterface.
    - initializeFunction (function): function to run on socket and username after construction. Hosts the construction of socket listeners.
    - destroyFunction (function): function called when the SocketInterface is replaced by another initializeFunction and destroyFunction. Should remove current socket listeners.
    - username (string): username associated with the SocketInterface.

- SocketInterface.changeInterface(initializeFunctionNew, destroyFunctionNew): method for destroying and updating the SocketInterface when a room change or other even causes the socket listeners and attributes to change.
    - initializeFunctionNew (function): function to replace initializeFunction with.
    - destroyFunctionNew (function): function to replace destroyFunction with.

SocketInterfaceFunctionSheet.js:

Function sheet for specific SocketInterface init and destroy functions. Init functions add listeners and change socket rooms, while destroy functions remove listeners and do other actions on the current SocketInterface.

Naming Convention:

- siXXXXXXXY: SocketInterface initialize functions
- sdXXXXXXXY: SocketInterface destroy functions

==========

Client Side:

Directory:

- **/js**
    - **/engine**
        - **/cameracontrols**
            - **OrbitingFollowControls.js**
        - **/gameinterface**
            - **GameInterface.js**
            - **GameInterfaceFunctionSheet.js**
        - **/helpers**
            - **ElementHelpers.js**
            - **PlayerMovementHelper.js**
        - **/hermes**
            - **/hermesinterface**
                - **HermesInterface.js**
                - **HermesInterfaceFunctionSheet.js**
            - **Hermes.js**
        - **/includes**
        - **/init**
            - **Init.js**
        - **/map**
            - **Map.js**
        - **/models**
            - **/obj**
            - **/json**
- **/pictures**
    - **/icons**
- **/style**
    - **/fonts**
    - **style.css**

/cameracontrols

Folder for holding threejs camera controllers

OrbitingFollowControls.js:

A set of threejs camera controls that allow the camera to orbit a moving object.

- `new OrbitingFollowControls(cameraObject, domElement, orbitObject, orbitRadius): constructor`
    - `cameraObject (THREE.PerspectiveCamera): camera object to apply the controls to.`
    - `domElement (HTMLdom): dom element where camera scene is drawn.`
    - `orbitObject(THREE.Mesh): Mesh for the controls to orbit.`
    - `orbitRadius (number): radius for the camera to orbit the object.`

- `OrbitingFollowControls.update(): update the position and tilt of the camera based on the elapsed time since the method was last called.`

- `OrbitingFollowControls.dispose(): removes listeners for the controls.`

/gameInterface

Files related to GameInterface objects

GameInterface.js:

GameInterface object constructor. This object allows for different game rooms to be displayed on the client side through hermes. Its animate function is called continuously with requestAnimationFrame from the Init.js's animate loop. Its init function is called when the object is constructed. Its destroy function is called when the interface is changed or closed. Each custom init function has the potential to render a new scene and is programmed accordingly. The GameInterface is hot swapped, and often is, to render new rooms while removing traces of the old init call (room).

- `new GameInterface(initializeFunction, destroyFunction, animateFunction): constructor`

- `NOTE: this object typically references global objects and variables!`
    - `initializeFunction (function): init creator function that is called to set up a room for the interface. Gets overwritten when the interface changes. Contains room object variables like controls, helpers and arrays.`
    - `destroyFunction (function): method that needs to be called when the interface is changed or removed. Removes controls, room variables and other specific room objects.`
    - `animateFunction (function): animation loop for the current interface room. Usually calculates (or looks up) dT first for reference in animations.`

GameInterfaceFunctionSheet.js:

Specific functions for init, destroy and animate methods for GameInterfaceObjects. These functions make up programmable client side rooms, doing things like drawing meshes, moving meshes and using helpers to communicate to the server through hermes.

`Naming Convention:`

- `giXXXXXXXY: GameInterface initialize functions`
- `gaXXXXXXXY: GameInterface animate functions`
- `gdXXXXXXXY: GameInterface destroy functions`

/helpers

ElementHelpers.js:

A set of helpers that draw HTML to get user input. These perform functions like using text boxes to get strings or creating context menus. Used in conjunction with threejs to create graphics for the game.

Constructors:

- **`ContextMenu()`**

    - Creates a drop down context menu by right clicking particular html elements
- **LoginBox()**
    - Box for handling login info
- **MessageBox()**
    - Unfinshed
- **OptionsMenu()**
    - Draws the main options menu
- **InteractionsMenu()**
    - Creates the interactive object menu

- ContextMenu.addFormulas(formulaObject): add context menu construction formulas to the object.
    - formulaObject (object): the object for applying menus to specific right-clickable divs.
        - formulaObject[divId].numOptions (int): The number of drop downs the div with id=divId should have.
        - formulaObject[divId][dropDownIndex].gameAction (string): The text that gets placed in the drop down.
        - formulaObject[divId][dropDownIndex].onClick (string): stringified javascript code to run when the dropdown is clicked.
    - Ex: {numOptions:1, 0:{gameAction:"Option 1", onClick:"console.log('Clicked Option 1');"}

- ContextMenu.removeFormulas(formulaObject): remove context menu construction formulas from the object.
    - formulaObject (object): the object for applying menus to specific right-clickable divs.
        - formulaObject[divId].numOptions (int): The number of drop downs the div with id=divId should have.
        - formulaObject[divId][dropDownIndex].gameAction (string): The text that gets placed in the drop down.
        - formulaObject[divId][dropDownIndex].onClick (string): stringified javascript code to run when the dropdown is clicked.
    - Ex: {numOptions:1, 0:{gameAction:"Option 1", onClick:"console.log('Clicked Option 1');"}

- ContextMenu.load(event, element): draw the context menu on the screen based on the right click domElement and the element in question.
    - event (event): right click dom event firing
    - element (HTMLdom): HTML element that called the context menu draw.

- ContextMenu.close(): close the last/only context menu currently drawn.

==========

- LoginBox.open(): open the initial login box.

- LoginBox.close(): close the initial login box.

==========

- MessageBox.open(): open message box.
- (UNDER CONSTRUCTION)

==========

- OptionsMenu.open(): draws the main options menu on the screen.

- OptionsMenu.select(menu): opens the selected options menu and closes the rest.

    - menu (int): specifies which menu to toggle to.

- OptionsMenu.close(): closes the main options menu.

==========

- InteractionsMenu.update(interactions): updates the InteractionsMenu by creating and storing the innerHTML that fills the InteractionMenu wrapper. If the menu is currently drawn, it places this HTML fill in the wrapper. If not it displays nothing.

    - Interactions (object): an object for queing the innerHTML of the InteractionsMenu wrapper
        - interactions[objectName].objectName (string): name of the interactive object.
        - Interactions[objectName].imgName (string): name of the icon image file of the interactve object. DOES NOT INCLUDE FILE EXTENSION.
        - Interactions[objectName].formula (object): context menu formula object for interactive object click.
            - formula (object): the object for applying context menus to specific right-clickable divs.
                - formula.numOptions (int): The number of drop downs the div should have.
                - formula.gameAction (string): The text that gets placed in the drop down.
                - formulaObject.onClick (string): stringified javascript code to run when the dropdown is clicked.
            - Ex: {numOptions:1, 0:{gameAction:"Option 1", onClick:"console.log('Clicked Option 1');"}
    - EX: {"Cube":{objectName: "Cube", imgName: "testcube", formula: {numOptions:1, 0:{gameAction:"Option 1", onClick:"console.log('Clicked Option 1');"}

- InteractionsMenu.open(): opens the most recently updated InteractionsMenu.

- InteractionMenu.close(): closes the InteractionsMenu.

PlayerMovementHelper.js:

This constructor allows the attachment of the rotating compass to a selected player actor. It can be rotated with the A and D keys, while the W key may allow the user to broadcast a forward movement in that direction, depending on some factors.

- new PlayerMovementHelper(compass, pawn): constructor
    - compass (THREE.Mesh): the compass object that is to rotate around the selected actor.
    - pawn (THREE.Mesh): the mesh that visually inherits the controls (the rotating compass and movement).

- PlayerMovementHelper.update(): updates the rotation and position of the compass object depending on the pawns location and active button presses.

- `PlayerMovementHelper.currentDirection(): returns an integer that represents the compass's active direction.`

- `PlayerMovementHelper.dispose(): removes the helpers listeners.`

HermesInterface.js:

An object for handling hermes' socket io states and listeners, depending on the current server gameroom the client's socket is in. also contains user defined methods that can be called to communicate to the server through hermes.

- `new HermesInterface(initializeFunction, destroyFunction): constructor`
    - `initializeFunction (function): the function that is to execute on the construction of the object which contains listeners, methods and or interface helpers.`
    - `destroyFunction (function): the function that is to be called when the HermesInterface changes or is destroyed. This function removes socket event listeners.`

HermesInterfaceFunctionSheet.js:

A sheet of user-defined functions for use as HermesInterface object initialize and destroy functions. Functions contain user callable methods for access in the engine through hermes.

`Naming Convention:`

- `hiXXXXXXXY: HermesInterface init function.`
- `hdXXXXXXXY: HermesInterface destroy function.`

Hermes.js

An object used to communicate with a server using socketiojs. All server emits are sent through hermes, and hermes handles serever event listeners with HermesInterfaces, which are abstract socket state machines. HermesInterface changes are triggered by hermes, usually when the current GameInterface is also changed by the server.

Init.js

The initial code for the engine which establishes a camera, scene, hermes, renderer, current game interface, and other variables for use by the current game interface on an html canvas.

Map.js

Constructor for the client Map object.

- new Map(mapBuild): constructor
    - mapBuild (object):
        - mapBuild.x (int): # of tiles on the x axis
        - mapBuild.z (int): # of tiles on the z axis
        - mapBuild.tileArray (n x m array): Contains an array of tiles that has n and m dimensions corresponding to every tile on the world map. Inaccessible tiles should be listed in this array as null. 0 indicates the tile in (-z)(-x) corner. (-z) is considered NORTH.

- Map.getTileByID(id): returns tile object corresponding to tile with id ‘id’.
    - id (int): id of tile in tileArray to be returned.

- Map.get[North, South, West, East, NorthWest, SouthWest, SouthEast, NorthEast]Of(tile): returns the tile located in the direction specified from the tile argument.
    - tile (object): tile to find direction relative to
        - tile.x (number): xposition of tile
        - tile.y (number): yposition of tile
        - tile.z (number): z position of tile
        - tile.id (int): integer tile id