|  |  |
| --- | --- |
|  | NavPal Floorplan Creator |
|  |  |
| 5/4/2013 | Authoring Tool Summary |
|  | A brief overview of how the Authoring Tool was structured, what features are currently available within the tool, and instruction on how the tool can be extended to add more features. |

NavPal Floorplan Creator

Authoring Tool Summary

# **Overview**

Once the Authoring Tool is initialized and ready to accept user input (see the Front End Summary for more details leading up to this point), the file StateManager.js takes over. The State Design Pattern is used to operate between the different “states” (draw, select/move, zoom in, etc.) in the tool. Check out <http://en.wikipedia.org/wiki/State_pattern> to learn more about the State Design Pattern, and check out <http://robdodson.me/blog/2012/06/02/take-control-of-your-app-with-the-javascript-state-patten/> to see some of the inspiration behind a lot of the code in StateManager.js.

# **State Manager**

There exists a lone StateManager object in the tool. It is the only global variable.

# **Globals Container**

Each building that the user uploads can contain multiple floors. For each floor, a Floor object (defined in draw/floor.js) is created before the Authoring Tool is entered, and the State Manager contains an array of Floor objects. To keep track of the current state of each floor to enable switching between them in the Authoring Tool, two methods are employed:

1. The Floor object contains class variables that keep track of certain higher-level features like spaces and obstacles.
2. The Floor object contains a Globals Container. This is described in further detail below.

A Globals Container (defined in lib/GlobalsContainer.js) maintains the state of each Point (defined in draw/Point.js) and Line (defined in draw/line.js) in the floor plan. It also contains methods for adding and removing such objects, and a slew of other utilities for similar operations such as the functions that draw all the Points and Lalls on the floor. Most importantly, it contains two key class variables:

1. points
   1. This array contains all of the Point objects that currently exist on the given floor. It is accessed and modified by nearly every state in the Authoring Tool.
2. walls
   1. This array contains all of the Line objects (walls and doors) that currently exist on the given floor. Its use is also ubiquitous in the Authoring Tool.

# **View**

# This section gives a brief description of each feature in the authoring tool. The tools appear on the left hand side of the authoring tool.

# Select:

# The select state allows the user to modify points and lines. There is the ability to click and drag points or lines. This state can also be used to delete points or lines. It is possible to select multiple points or lines at once, by dragging to create a selection area.

# Pan:

# The pan state allows the user to click and drag to pan the floor.

# Zoom In:

# The zoom in state allows the user to click to zoom in.

# Zoom Out:

# The zoom in state allows the user to click to zoom out.

# Draw:

# The draw tool allows the user to add new lines (walls) to the floor plan. The user clicks to add a point, and line are automatically created from the previous point. New points 'snap' to existing points and lines to make creating closed off areas easier. Any overlapping lines will automatically split existing lines into two lines and will add a point to the location of intersection.

# Door:

# The door tool is used to mark that a wall represents a door, which a person can pass through to get to a different room. Every wall can be toggeled between a normal wall and a door by clicking on a wall in this state. If a user hovers over a wall, it will change color to indicate which wall will change upon clicking.

# Exit:

# The exit tool is similar to the door tool, but it used to indicate exits or entrances on the floor that lead to outside the building.

# Classification:

# The classification tool is used to label rooms, hallways, or obstacles. The authoring tool automatically detects the smallest closed off areas, which it considers be to be rooms, hallways, or obstacles. While hovering in this state, a blue polygon appears to indicate the closed off area the user will select. Upon clicking, a box pops up for the user to enter the name of the room and a description. Any closed off areas that have been labeled become red.

# Landmark:

# The landmark state can be used to indicate landmarks on the floor. If the user clicks in this state, a box pops up to allow the user to enter the name and description of a landmark. If the use clicks near an existing landmark, they can edit the name or location of that landmark, or simply delete it.

# Stair:

# The stair tool is used to indicate an interfloor connection. The user clicks in the area where the stair case exists, and a box will pop up. The user must label the stair case, using a name. If this is the first time labeling a particular stair case, the user types in a new name. If the same stair case has already been labeled on another floor, a list of existing stairs will be populated for the user to select from.

# Elevator:

# The elevator tool is similar to the stair tool, but it indicates an elevator can be used to connect to another floor.

# **State Functionality**

Each state must implement the following methods, because State Manager calls each of them:

* enter()
  + Defines the functionality when a state is first entered (i.e. in initialization or when switching out of another state)
* exit()
  + Defines the functionality when a state is about to be exited (to switch to another state)
* draw()
  + Draws any state-specific details that aren’t covered in the State Manager’s redraw() function
    - For example, a point is drawn at the user’s cursor (and a line from the most recently-drawn point if necessary) in the Draw State’s draw() function when the drawing tool is engaged.

# **Base State**

The Base State (defined in states/BaseState.js) is the object that all other states inherit from (though it is not required). It defines all of the keyboard and mouse events that can take place in the Authoring Tool, and has them each do nothing. This way, each state only has to implement (and thus override) the events it wants to handle.

# **To Add a New State:**

1. Upload a tool thumbnail to the img directory.
2. In authoringTool.html in the “Toolbar” section, add a row in the appropriate group.
   1. The tool dock on the left-hand side of the authoring tool subdivides the tools into groups (separated by an underline). Find the appropriate group for the new tool, or make a new one by adding <div class="toolGrouping"></div>.
   2. Look for the bottommost <div class="row-fluid toolrow"></div> in the appropriate group. If it only has one member (i.e. only one <div> inside of it), add another tool inside it. If it has two members, create a new <div class="row-fluid toolrow"></div> beneath.
      1. To add a new tool, create a new <div> which follows the form <div id=$toolName class=”toolIcon tool left/right” style=”background-image: url($thumbnailUrl) “ data-toggle=”tooltip” title=$Description></div>
      2. For example, <div id="Draw" class="active toolIcon tool left" style="background-image: url(img/wallIcon.svg)" data-toggle="tooltip" title="Wall Tool: Click to select the starting point of a wall. Move the mouse to the end of the wall and click to create. Press the space bar to release the tool. (/)"></div> is the tool div for the Draw tool.
3. Create a new javascript file for the new state and place it in the states directory.
   1. Look at one of the existing states as a model (DrawState.js, for example)
   2. Name the new state $toolNameState, where $toolName is the id you used in the html (for example, “DrawState”).
   3. The new state MUST implement the following functions, even if they do nothing (read more about them in the section entitled **State Functionality**):
      1. enter()
      2. exit()
      3. draw()
   4. It is recommended that the new state inherit from BaseState (simply add the line $toolNameState.prototype = new BaseState(); after the constructor)
      1. See more on BaseState in the section entitled **BaseState.js**
   5. Add the path to this new javascript file in the section title “Authoring tool” beneath the existing states
      1. For example, the Pan State has the following form:

<script src="./js/states/PanState.js"></script>

1. In StateManager.js, add the new state in the list of available states
   1. For example, the Pan State has the following form: "Pan": new PanState(this)

# **To Add Room Label on Hover:**

First, the state must implement the mouseMove(event) function. At the top of this function (or where appropriate), add the function call stateManager.hoverRoomLabel(new Point(event.pageX, event.pageY), pointAtCursor), where pointAtCursor must be defined as

var pointAtCursor = stateManager.currentFloor.globals.view.toRealWorld(new Point(event.pageX - stateManager.currentFloor.globals.canvas.x, event.pageY - stateManager.currentFloor.globals.canvas.y))

# **To Add a New Mouse/Keyboard Event:**

1. In main2.js, add the new event handler in the function initCanvasEventHandlers
   1. Be sure to use the proper jQuery event handler.
2. In state/BaseState.js (see the section **Base State** for more information), add the function that step 1 calls.
3. Add the function in whatever state should catch this event and do something with it.