Semantic Map Design/Spec (V1.0)

Authors:

- Adarsh Pyarelal (adarsh@arizona.edu)
- Jeff Rye (rye@sift.net)
- Roger Carff (rcarff@ihmc.us)
- Dana Hughes
- Curt Wu
- Brett Israelsen
- Guy Hoffman

Purpose: This document serves to collect and refine discussions on the format of a machine-readable, high-level representation of the mission environment, which we term a *semantic map.* Eventually, the document will also contain the specification of the format and notes on the rationale behind the design of the components.

Table of Contents

Motivation

Semantic maps are motivated by a number of reasons. At a high level, given that (i) ASIST is a program motivated by developing agents with machine theory of mind, i.e. the ability to develop models of human teammates, and that (ii) humans tend to think, talk about, and plan about mission environments not in terms of coordinates, but higher-level, semantically meaningful regions with unique labels, we assert that it makes sense to imbue AI agents with some knowledge of these regions and labels.

A large fraction of the performers in ASIST already use some version of a machine-readable semantic map. The majority of these use the JSON files that serve as configuration files for IHMC's LocationMonitor agent. The goal of this document is to capture requirements for the semantic map format that are common to as many performers as possible, and convert those into a format specification that enables easier data exchange and minimizes the re-invention of the wheel.

Design Principles

The environment of the ASIST task has several levels of static and dynamic information. At the base is the pre-mission "blueprint" of the task environment. This is perturbed by the pre-mission "ground truth", including victims and changes to the blueprint (collapses, fires, new openings). This is further perturbed by in-mission changes to the ground truth. Finally, there are dynamic agents moving in the space and affecting it.

The structure of the specification should therefore allow layers of information starting from the fundamental structure of the game environment, such as the static walls in the building and then adding layers of information on top of that. This layering will allow:

- building a simple representation now and adding to it as we get more and more information that is needed and can derive more and more useful map information
- Not every team will find value in every bit of information. By layering the information, each group can decide at what layer is the fundamentation information they need and then layer on top of that their custom information that maybe only they need.

This layering approach does require that higher layers are able to reference elements in lower layers in order to create linkages. For example, given a room layer and a victim layer, the ability to link together the room that the victim is in.

Caveats: It is inevitable that any format specification will not be able to perfectly address all the desires of all the teams. In particular, things that are unlikely to be addressed at this time (simply because they are decided not 'low-hanging fruit') include specifying regions that have curvilinear boundaries, Additionally, while it might be fruitful to incorporate polygons that are non-convex or

not quadrilaterals in the future, we will begin with regions that are simply axis-aligned bounding boxes (AABBs) in either two dimensions (rectangles) or three dimensions (cuboids)

Information Retrieval Queries

The functional use of the data storage format can be assessed with respect to probable queries that it should support. Below is a non-exhaustive illustrative list of queries that can guide the specifics of the data format. This API is not meant to be prescriptive, but to serve as a basis of discussion.

- *getLocationName(x, y, z)* — returns the location name for a point
- *getEntitiesInLocation(name/id)* — returns a list of entities inside a location
- *getTypedEntitiesInLocation(ent_type, loc_name/id)* — returns a list of entities of a specific type inside a location
- *isInLocation(ent_name/id, loc_name/id)* — tests whether an entity is in a location
- *canPass(loc1_name/id, loc2_name/id)* — tests whether there is a passable connection between the two locations

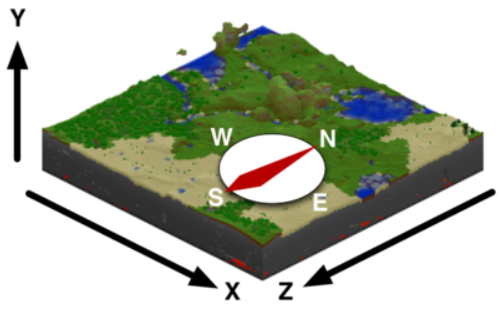
Preliminaries

Glossary

- Block — Cube with integral Cartesian coordinates on the map, equivalent in size to one Minecraft block.
- Entity — Agent that can move in space (player, AI agent, 'mobs'). Its position vector can have real-valued components.

Minecraft coordinate system

The Minecraft coordinate system is shown in the figure below for reference. The x and z axes form a plane that is parallel to the surface, while the y axis represents the vertical direction.



The Minecraft environment is discretized. Each block has integer coordinates and occupies a fixed volume in 3D space. Entities, however, can have real-valued coordinates.

Running example

We use the map shown in Figure 1 as a running example throughout the text to demonstrate concrete examples of the semantic map elements.

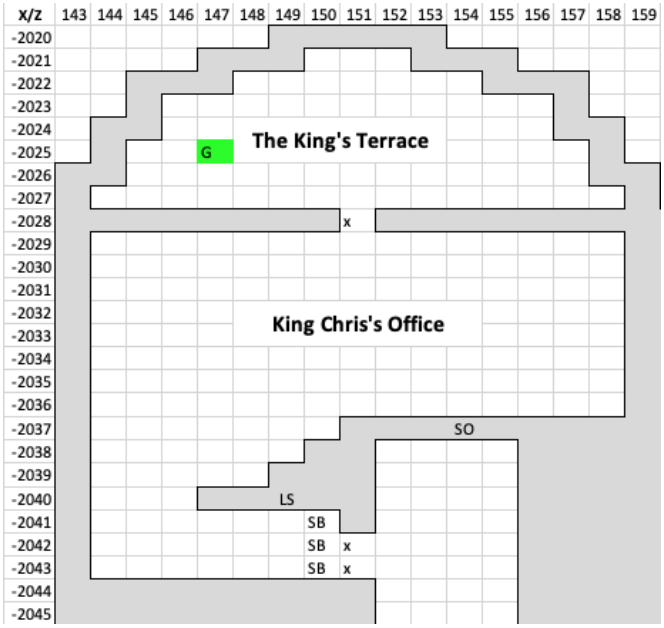


Figure 1: Minecraft Block Layout.

Legend:

- G: Green victim
- x: Door
- LS: Light switch
- SO: Static opening
- SB: Static blockage

Map building blocks

The fundamental building blocks of the semantic map are 'locations', 'connections', and 'objects'.

Guideline for knowledge engineers: aim for surjective mapping (i.e. each Cartesian coordinate in the mission area that a player can be in should map to at least one location).

Locations

The term 'location' has been chosen to be general enough to encompass the different types of named spatial locations of interest in a Minecraft environment. A 'location' could be a point in space, for example, the Cartesian coordinates of a victim block. It could also be a rectangle, for example, the projection of any of the rooms in the 'Sparky' map office building environment onto the XZ plane, or a cuboid, e.g. the Sparky rooms themselves.

Each ‘location’ object has an id, a name, a type, and either a bounds or a list of child locations. Some examples of locations are: room, hallway, elevator, balcony, yard, bathroom, closet. An example of a location is shown below.

```
{
    "id": "kcoe",
    "name": "King Chris's Office Entryway",
    "type": "room",
    "bounds": {
      "type": "rectangle",
      "coordinates": [{"x": -2043, "z": 147}, {"x": -2040, "z": 151}]
    }
  },
```

If a location needs to be split up then its bounds are determined by its child locations (that is, the parent is a disjoint union of its children of the same type) so instead of a bounds it has a list of child locations as such:

```
  {
    "id": "kco",
    "name": "King Chris's Office",
    "type": "room",
    "bounds": null,
    "child_locations": ["kcoe", "kco1", "kco2", "kco3"]
  },
```

Non-point locations can also overlap with other locations.

Bounds

The term ‘bounds’ is chosen even though it doesn’t quite fit with ‘point’ locations, for the sake of convenience. Each ‘bounds’ object will have a ‘type’ and ‘coordinates’. The type is one of ‘point’, ‘rectangle’, ‘block’, or ‘cuboid’. In the future, this set of allowed values will be extended to include ‘polygon’ (and possibly ‘polyhedron’ as well?).

Examples:

Point

```
   "bounds": {
      "type": "point",
      "coordinates": [{"x": -2025, "z": 147}]
    }
```

Rectangle

```
"bounds": {
   "type": "rectangle",
   "coordinates": [{"x": -2043, "z": 151}, {"x": -2041, "z": 152}]
 }
```

Block

```
"bounds": {
   "type": "block",
   "coordinates": [{"x": -2025, "z": 147}]
 }
```

Cuboid

```
"bounds": {
    "type": "cuboid",
    "coordinates": [
        {"x": 1, "y": 0, "z": 1},
        {"x": 10, "y": 10, "z": 10}
    ]
 }
```

In the case of the rectangle and cuboid, the bounds represent a pair of points at the ends of a diagonal. Blocks are considered to equate to the size of one Minecraft block.

Other values for 'type' can be: *'line'* with 2 coordinates (maybe more in the future), and in the future *'polygon'* with at least 3 coordinates.

Objects

There are a number of important labeled objects that need to be represented in the semantic map. In the USAR mission, this includes victims and light switches. For example, the object below represents the location of the green victim in Figure 1.

Example: Green victim object

```
{
  "id": "vg1",
  "type": "green_victim",
  "bounds": {
    "type": "block",
    "coordinates": [{"x": -2025, "z": 147}]
  }
```

```
}
```

We disallow multiple labels for the same 'point'-type location.
However, note that a given label can map to multiple 'rectangle' or 'cuboid' regions.

If the point object represents a block, the components of the coordinates must be integers. If the location represents an entity, the components can be real-valued.

An object differs from a point location since it allows us to represent objects (blocks/block collections/entities) that might change their locations, but we want to keep track of which object has moved - for example, if in the future we have a scenario with a victim that is moved from one place to another in the process of rescuing them.

Connections

Connections define how to go from one location to another. They have an id, a type, a bounds, and a list of the IDs of the locations they connect. They can also have a name. Some types of connections are: door, double_door, elevator_door, extension, opening, debris, and fire.

```
{
  "id": "c11",
  "name": "Door from to the Terrace from King Chris's Office",
  "type": "door",
  "bounds": {
    "type": "rectangle",
    "coordinates": [{"x": -2028, "z": 151}, {"x": -2027, "z": 152}]
  },
  "connected_locations": ["kco2", "tkt"]
}
```

Not all connections are passable. Debris and fire have a 'passable' property that is set to 'false' unless they are cleared.

```
{
  "id": "c16",
  "name": "Debris blocking entry into King Chris's Office",
  "type": "debris",
  "passable": false,
  "bounds": {
    "type": "rectangle",
    "coordinates": [{"x": -2043, "z": 150}, {"x": -2040, "z": 151}]
  },
  "connected_locations": ["koce1", "kcoe2"]
},
```

Complete Example

Figure 2 represents a graphical view of the portion of the map depicted in Figure 1, suppressing the actual walls and blocks, and surfacing what is actually conveyed by the semantic map to the machine that is reading it. The figure shows views of the region before and after the collapse of the office building.

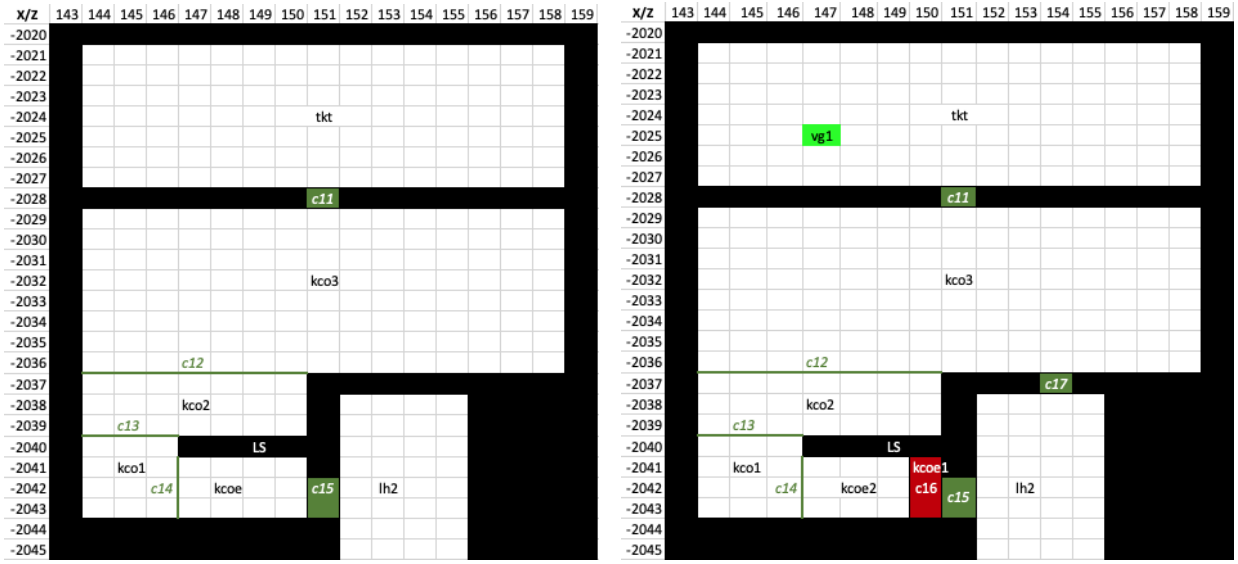


Figure 2: Visualization of semantic map pre (left panel) and post (right panel) collapse. After the building collapses, a green victim (vg1), a static blockage (c16), and a new connection (c17) appear.

Pre-collapse

We represent the pre-collapse map in the left panel of Figure 2 with the following JSON object:

```
{
  "locations": [
    {
      "id": "lh2",
      "name": "Left Hallway",
      "type": "hallway",
      "bounds": {
        "type": "rectangle",
        "coordinates": [{"x": -2089, "z": 152}, {"x": -2037, "z": 156}]
      }
    },
    {
```

```
  "id": "kco",
  "name": "King Chris's Office",
  "type": "room",
  "bounds": null,
  "child_locations": ["kcoe", "kco1", "kco2", "kco3"]
},
{
  "id": "kcoe",
  "name": "King Chris's Office Entryway",
  "type": "room",
  "bounds": {
    "type": "rectangle",
    "coordinates": [{"x": -2043, "z": 147}, {"x": -2040, "z": 151}]
  }
},
{
  "id": "kco1",
  "name": "King Chris's Office corner",
  "type": "room",
  "bounds": {
    "type": "rectangle",
    "coordinates": [{"x": -2043, "z": 144}, {"x": -2039, "z": 147}]
  }
},
{
  "id": "kco2",
  "name": "King Chris's Office desk area",
  "type": "room",
  "bounds": {
    "type": "rectangle",
    "coordinates": [{"x": -2039, "z": 144}, {"x": -2036, "z": 151}]
  }
},
{
  "id": "kco3",
  "name": "King Chris's Office main area",
  "type": "room",
  "bounds": {
    "type": "rectangle",
    "coordinates": [{"x": -2036, "z": 144}, {"x": -2028, "z": 159}]
  }
},
{
  "id": "tkt",
  "name": "The King's Terrace",
  "type": "balcony",
  "bounds": {
    "type": "rectangle",
    "coordinates": [{"x": -2027, "z": 144}, {"x": -2020, "z": 159}]
  }
```

```
  }
],
"objects": [
  {
    "id": "ls1",
    "name": "Light switch in King Chris's Office",
    "type": "light_switch",
    "bounds": {
      "type": "block",
      "coordinates": [{"x": -2040, "z": 150}],
    },
    "facing": "south",
  }
]
"connections": [
  {
    "id": "c11",
    "name": "Door from to the Terrace from King Chris's Office",
    "type": "door",
    "bounds": {
      "type": "rectangle",
      "coordinates": [{"x": -2028, "z": 151}, {"x": -2027, "z": 152}]
    },
    "facing": "south",
    "connected_locations": ["kco2", "tkt"]
  },
  {
    "id": "c12",
    "type": "extension",
    "bounds": {
      "type": "line",
      "coordinates": [{"x": -2037, "z": 144}, {"x": -2036, "z": 151}]
    },
    "connected_locations": ["kco2", "kco3"]
  },
  {
    "id": "c13",
    "type": "extension",
    "bounds": {
      "type": "line",
      "coordinates": [{"x": -2039, "z": 144}, {"x": -2039, "z": 147}]
    },
    "connected_locations": ["kco1", "kco2"]
  },
  {
    "id": "c14",
    "type": "extension",
    "bounds": {
      "type": "line",
      "coordinates": [{"x": -2043, "z": 147}, {"x": -2040, "z": 147}]
```

```
      },
      "connected_locations": ["kcoe", "kco1"]
    },
    {
      "id": "c15",
      "name": "Door to King Chris's Office",
      "type": "double_door",
      "bounds": {
        "type": "rectangle",
        "coordinates": [{"x": -2043, "z": 151}, {"x": -2041, "z": 152}]
      },
      "facing": "south",
      "connected_locations": ["lh2", "kcoe"]
    }
  ]
}
```

Post-collapse: Representing map updates

Information about a map update (either static or dynamic perturbations, or locations of victims to construct high-level percepts such as 'does the player currently see a victim on the screen') can be provided as an 'operator' (borrowing from PDDL terminology) with 'effect lists' that describe additions and deletions to the map. An operation is meant to be 'atomic', in the sense that the entire update must be processed before calculating 'global' properties of an agent's internal state (for example, a list of precomputed paths). This is because the effect lists are unordered.

The 'delete' list contains the IDs of the semantic map components that should be removed in an agent's internal representation of the mission environment.

The JSON object below represents the map update that transforms the map in the left panel of Figure 2 into the map on the right panel of Figure 2. (Note: the deletions lists are not necessary for this example, but are present so that the format can be seen.)

```
{
  "deletions": {
    "locations": [ "kcoe1", "kcoe2" ],
    "connections": [ "c16" ]
  },
  "modifications": {
    "locations": [
      {
        "id": "kcoe",
        "bounds": null,
        "child_locations": ["kcoe1", "kcoe2"]
      }
```

```
  ]
  "connections": [
    {
      "id": "c14",
      "connected_locations": ["kcoe2", "kco1"]
    },
    {
      "id": "c15",
      "connected_locations": ["lh2", "kcoe1"]
    }
  ]
},
"additions": {
  "locations": [
    {
      "id": "kcoe1",
      "type": "room",
      "bounds": {
        "type": "rectangle",
        "coordinates": [{"x": -2043, "z": 150}, {"x": -2040, "z": 151}]
      }
    },
    {
      "id": "kcoe2",
      "type": "room",
      "bounds": {
        "type": "rectangle",
        "coordinates": [{"x": -2043, "z": 144}, {"x": -2040, "z": 150}]
      }
    },

  ],
  "connections": [
    {
      "id": "c17",
      "name": "Collapsed wall between the hall and the office",
      "type": "opening",
      "bounds": {
        "type": "rectangle",
        "coordinates": [{"x": -2037, "z": 154}, {"x": -2036, "z": 155}]
      },
      "connected_locations": ["lh2", "kco3"]
    },
    {
      "id": "c16",
      "name": "Debris blocking entry into King Chris's Office",
      "type": "debris",
      "passable": false,
      "bounds": {
        "type": "rectangle",
```

```
          "coordinates": [{"x": -2043, "z": 150}, {"x": -2040, "z": 151}]
        },
        "connected_locations": ["koce1", "kcoe2"]
      }

    ],
    "objects": [
      {
        "id": "vg1",
        "type": "green_victim",
        "bounds": {
          "type": "block",
          "coordinates": [{"x": -2025, "z": 147}]
        }
      }
    ]
  }
}
```

Planned future extensions

Planned extensions for the format in the future:

- Support for polygons
- Support for defining complex shapes using unions.