# Task 11 - Spike: Game Graphs from Data

**CORE SPIKE**

**Context:** Selecting appropriate data structures and representations for game information is a critical performance and development issue for game programmers. Graphs are a general data structure with many applications. Developers should be able to take advantage of graph data structures in their game implementations.

**Knowledge/Skill Gap:** The developer is not familiar with the use of graph data structures for the representation of game world composed of locations and connections.

**Goals/Deliverables:**
[CODE] + [SPIKE REPORT]

Soon we will want to extend the Zorkish program you created in earlier tasks to include the loading of an Adventure text file. This file must contain the details of location and connection information for the game world as a graph data structure. Refer to the game specification document on the unit website for details.

In this spike we are going to implement a game world graph, with locations and connection, loaded from file. The user will be able to navigate to different locations using the "go" command (hard-coded). This is NOT yet part of the Zorkish code base. It should NOT contain a full command processor or stages (states) of earlier spikes.

This spike is a small stand-alone demonstration of a game world graph loaded from a text file only.

Create a program that demonstrates the following:

1. **Design**: Specify a design as a sketch or diagram (REQUIRED) for the text-file format that represents a Zorkish game "Adventure" details. Include this in your spike report details.
    - This can be vector based, or a design on paper, and included as images, in your design document.
    - Specifically, your design will need to include (at this stage) world locations, location details (name, description etc.) and connections to other locations.
    - You can include other details in your design, but we only need locations and connections.
2. **Args**: Take the game world "adventure" filename at run time using a command line argument.
3. **Graph**: Process the locations and connections as a graph data structure in your program.
4. "**Go**": Implement ONLY the basic "go" and "quit" commands. Print location and direction options (N, UP, NE etc) to the user so they know where they are and what directions they can go in.

NOTE: Do NOT implement a full command processor/pattern! Not needed yet.
NOTE: You must implement more than North, East, etc. This is NOT a grid world – any direction is possible!

**Recommendations:**

- Create a stand-alone world-graph demo. Do NOT integrate into Zorkish (yet). Seriously.
- Create a simple text file format. Write code to load the file (maybe just print the details back to screen to start with) and then create a graph using the loaded details.
- Don't worry about implementing other entities (rocks, boxes, swords etc) in the world for this spike. Just focus on locations and connections and the basic hard-coded "go" command.
- Make a list of the type of details that need to be stored at each location, then convert your list into a node design (class?) and a graph design. Identify what graph-based functions you will need to move a player around the world.
- Research STL "maps" ("dictionaries") if you haven't already. These are collections that allow you to access their contents using keys such as strings or ints – very handy.
- To move the player will you alter the graph? Does the graph contain a reference to the player, or does the Player contain a reference to the graph? What potential advantages/disadvantages are offered by each approach? (Answer this in your spike report – if you have a good answer.)
- Test early, test often. Frequently commit to your local repo with good comments! At the end – push.