

CS_2110_HW5

ReadMe

Lan Zhang

NetID: lz376

No.3669619

1.Features

➤ **Smart routing algorithm for gnomes:**

- Because each road has a capacity, gnomes might have to queue for roads. When routing gnomes, consider both the time spent on the road and the time spent on queuing;

➤ **Thread-safe gnome on road information:**

- To make gnomes queue at the road when it reaches its capacity, the information of that queue must be thread-safe among all gnomes. Each road now has a manager, which uses a BlockingQueue. With an analogy to producer-consumer model of multithreading, each gnome now acts like a consumer when entering the road (consuming 1 capacity), and acts like a producer when leaving the road (producing 1 capacity);

➤ **Push notification:**

- To make GUI more efficient, the system uses push-notification instead of constant update. Whenever there is a new village / road/ gnome or the gnome moves, the system will notify GUI to update its map;

2. Design & Data Structure

➤ **Design:**

- The system consists of 3 basic layers: GUI, the world (GnomeWald) and entities (Village, Road, Gnome). GUI should obviously be separated from the “world”, but they should be able to communicate with each other, so GUI knows the “world” it currently represents and the “world” knows the GUI representing it. Then notice that the gnomes on the map should not alter the abstract structure of villages and roads, which is a directed graph, so the systems separates Village and Road from Gnome and wrap them in class Map. GnomeWald as the “world” handles information of the map and all the gnomes, passing the GUI commands and gnome push-notifications between them.

➤ **UserInterface:**

- Contains Main(), initiate a GUI and a GnomeWald;

➤ **GUI:**

- Contains all GUI related elements, buttons, textfields, panels;

➤ **GnomeWald:**

- Contains the map and gnomen information;

- “Go-Betweenner” of the GUI and other classes , GnomeWald tells what GUI should draw, and delivers command from GUI to map and gnomes;
- Manages the traffic system;

➤ **Map:**

- Contains all the information of the villages and roads;
- Contains all the traverse strategy like depth-first-search, topological sort and minimum spanning tree;

➤ **Village:**

- Contains a set of incoming roads information and a set of outgoing roads information;
- Contains other self information: ID, position, capacity, current gnome residents, required stamped villages and other village require me;

➤ **Road:**

- Contains information of its start village and end village;
- Contains other self information: cost, time, capacity;

➤ **RoadManager:**

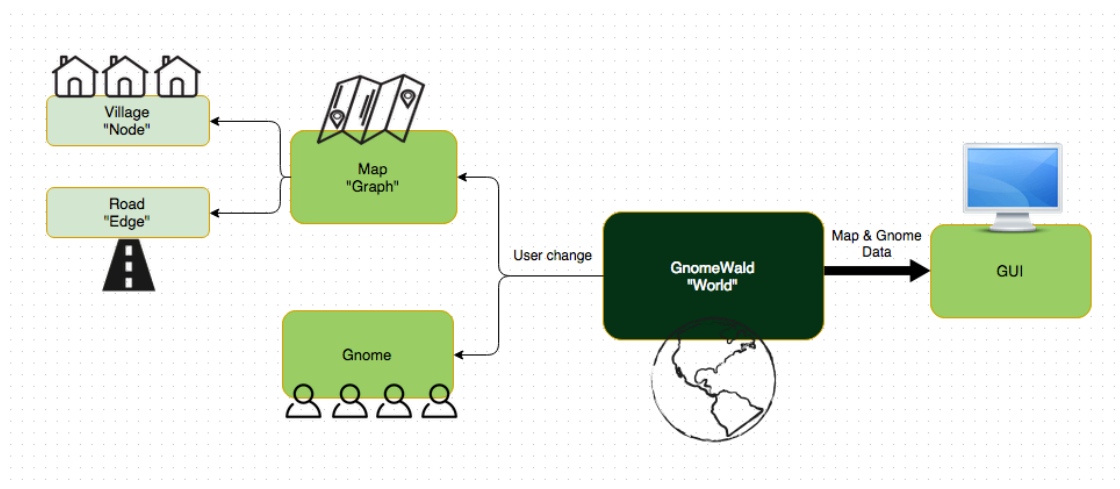
- Helps GnomeWald to control the map's traffic system;
- Each RoadManager is a blockingqueue, when a road is full, the roadmanager will block other gnomes who want to go through this road;

➤ **Path:** helper class

- A path is a sequence of roads;

➤ **Gnome:**

- Contains gnome self information: ID, position, status(stay, random walk or have destination), urgency level, current village, destination village, stamps village;



Data Flow Chart

- **Advantages of this data structure:**

- GnomeWald is the “Go-Between” of the GUI and stored information, prevent user directly change the data of the map and gnome;
- GnomeWald is also the “Go-Between” of Gnome and Map, gnomes “tells” GnomeWald where it wants to go, and GnomeWald “asks” the map for the possible paths and let gnome select the best one according to its urgency level;

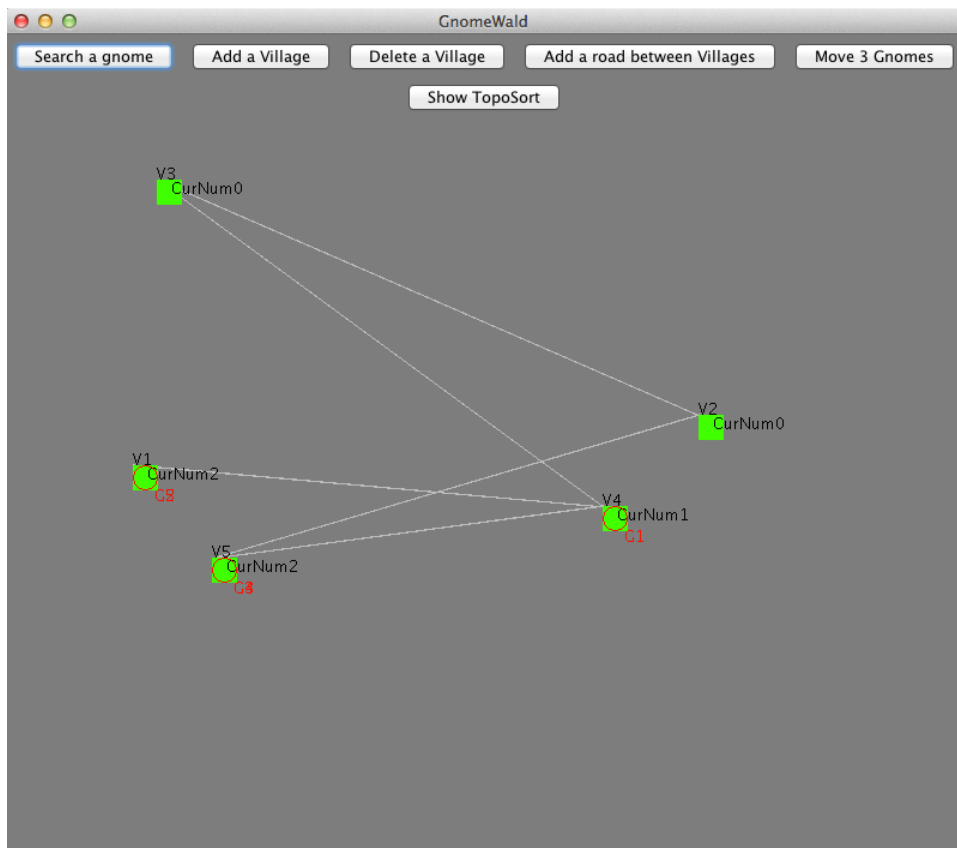
3.Algorithm:


- **Depth-First-Search:** when a gnome wants to go from its current village to a assigned village, it will tell the GnomeWald, and the GnomeWald will tell the map, then map will use DFS to find all the possible paths from gnome’s current village to the assigned village;
- **TopoSort:** when the GnomeWald been first initialized, the map will return a topological sort result;
- **Mimimum Spanning Tree:** when the GnomeWald been first initialized, the roads are connected to villages with using of Minimum spanning tree strategy;


4.GUI:


- **Delete a Village:** when delete a village, there are two options: delete all its connected road or delete the roads then reconnect the villages which previously are connected by the delete village; when delete a village, the gnome in this village will be “killed”;
- **Add a Village:** when add a village, its position is randomly generated and will also randomly generate some new gnomes, the new add village will connect to its closest village which could maintain the minimum spanning tree structure;
- **Add Road:** when add a road, there are two options: add one way road or two-way road;

- **Search Gnome:** input a gnome's ID and return its current position;
- **Move 3 Gnomes:** when move a gnome, there are two options: move randomly or to a user assigned village, the purpose of letting 3 gnomes move at the same time is to test the traffic system; when move randomly, gnome will random choose an adjacent village to go, however, when he gets there, and find that village has no capacity, he then will move random again, until he finds some village to stay, same approach to the gnome who have a destination, so that's why sometimes even only hit the "move random" and "go to assigned village" button once, the gnome will go from one village to another, and then go from that village to other one, that's because the previous villages have no capacity;
- **Show TopoSort Result:** return the initial GnomeWald map toposort result;



 Village: CurrNum is the number of current gnomes in this village

 Gnome: G3 is the ID of each gnome

 Road: Gnome will move along the road

5. Traffic System implementation:

When considering how to design the traffic system of this project, there are some problems raised during the design process:

- **When road has no capacity, gnomes queue on road:** each road has a random generated capacity, when there is no capacity on this road, other gnomes who want to go through this road have to wait. Each road has a RoadManager, who manages this queueing information of the gnomes on its road. RoadManager holds the information in a BlockingQueue, with size limit equals to the road capacity. When the number of gnome is equal to the BlockingQueue's size limit, other gnomes will wait in a queue. Only after the first gnome "leaves" the road and returns 1 capacity to the road, the first queueing gnome could enter the road;
- **Reroute:** each gnome has a random generated urgency level, when a gnome queueing in a waiting list of a road, it will recompare the cost of the current path and other paths' cost from current position to the destination village, the heuristic function is (Road total cost = urgency * road time + (1-urgency) * road money cost);
- **When village has no capacity:** when a gnome arrives at its destination village, it finds this village has no capacity for him, it will random pick an adjacent village and go;
- **Can't go to a certain village:** because this map (graph) is directed, so there is possible that a gnome cannot go from its current village to a user assigned village, when this situation happens (when the Map returns a null path between these two villages), the gnome will stay at its current village;