Introduction

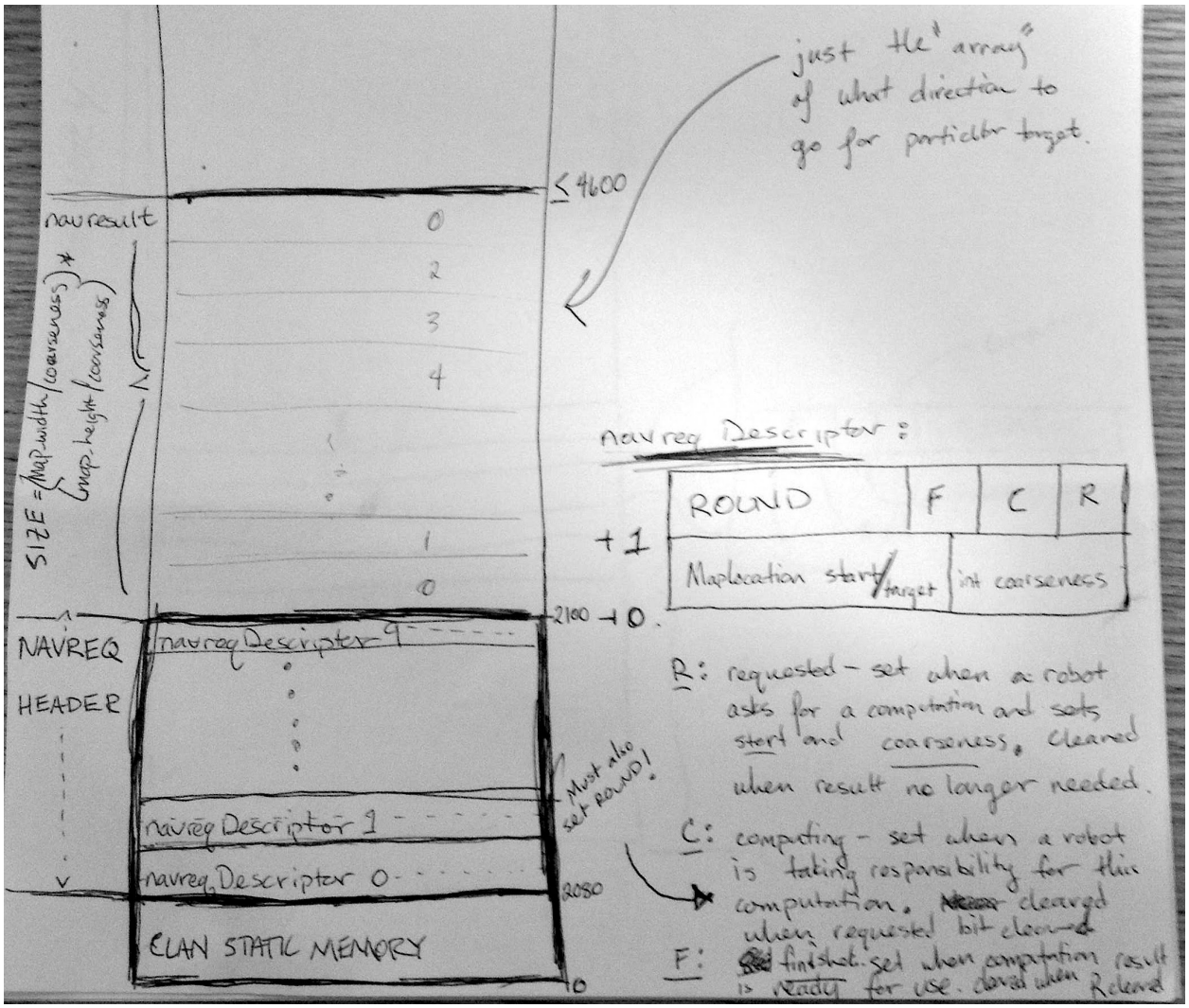
We explain the most important parts of Caddar. The radio has aspects inspired by standard memory segmentation and the data structure used in navigation is fascinating.

Distributed Computing

In Caddar, robots can post a computation “job” to the radio. Structures with spare bytecodes periodically scan the radio for newly posted jobs. If they can spare they bytecodes, they compute the solution and post it back to the radio for all robots to use.

The primary benefit is that caddar can use different strategies for navigation. We start with bug navigation, but we are capable of switching to Dijkstra when appropriate. Because certain structures (like PASTRs and the HQ) have more spare bytecodes each round than others, they typically compute paths using Dijkstra (described in the next sections) for the Cowboys.

The figure below illustrates the primary features of our distributed computing design.



Navigation

We initially implemented Bug, but that was tricked too easily. We then implemented A*, but that took too long. So we settled on an alternative. Compute Dijkstra and while computing, use Bug. Once Dijkstra was done, it could pick up wherever we were and get to the target. This still took longer than we wanted, so we coarsened the map and bugged between the coarse squares once Dijkstra finished. Still unsatisfied, we made a data structure to speed things up.

In the method costs list, we noticed something hidden at the bottom.

java/lang/StringBuilder/delete	20	false
java/lang/StringBuilder/indexOf	20	false
java/lang/StringBuilder/insert	20	false
java/lang/StringBuilder/lastIndexOf	20	false

indexOf performs an O(n) operation in O(1) cost. So we created “String Heap”[1]

Performance:

Type	Extract Min	Decrease Key	Dijkstra	Num Rounds[2]
List	O(n)	O(1)	$O(n^2)$	300
List + Heap	O(log n)	O(log n)	O(n log n)	55
String Heap	O(1)	O(1)	O(n)	30

[1] “String Heap” described later

[2] number of rounds to compute on castles

Micro

5 engagement behaviors.

- UNENGAGED
    - Perform macro behavior
- CHASE
    - Run at the enemy
    - Don’t be the first to step in firing range
- FIGHT
    - Attack weakest enemy
- RETREAT
    - Run away from the nearest enemy
- KAMIKAZEE
    - Charge towards where you can do the most damage
    - Self-destruct

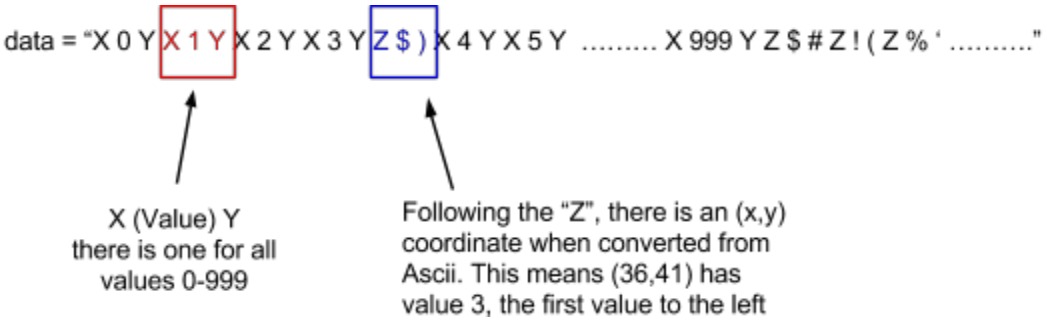
Macro

4 types of clans, BUILDER, DEFENDER, RAIDER, and IDLE. We decided that the best clan size was 5 since that performed well in micro, but didn’t take long to build.

String Heap

Data Structure:

Use a StringBuilder called data



Supported Operations:

Init:

- New StringBuilder with all coordinates at 999
- Hardcode this in < 10 bytecodes

GetValue (x,y):

Find 'Z' + char(x) + char(y)	(StringBuilder.indexOf)
Find the first X before that	(StringBuilder.lastIndexOf)
Read the value after that X	(StringBuilder.charAt)

Extract Min:

Find the first 'Z' which tags a coordinate	(StringBuilder.indexOf)
Read (x,y) coordinates immediately following	(StringBuilder.charAt)
Delete 'Z' and the coordinates	(StringBuilder.delete)

UpdateKey (x, y, newVal):

Find 'Z' + char(x) + char(y)	(StringBuilder.indexOf)
Delete	(StringBuilder.delete)
Find 'X' + newVal + 'Y'	(StringBuilder.indexOf)
Insert 'Z' + char(x) + char(y) immediately after	(StringBuilder.insert)