Design

Preliminary Design

Need:

- Data Flows
- Large modules/packages
- Architectural Design
  - The modules, and
  - Their interactions

Architecture

We will be utilizing the Model-View-Controller architecture. Since this architecture splits the project into 3 parts, it will allow us to easily collaborate on code. Each module will have it's own API, or public interface, allowing someone working on one part of the project to easily interact with another part without worrying about the underlying implementation. Further, the person working on the one module will be able to make changes without fear of breaking it for the other modules, since they know how the other modules are interacting with it.

TODO: Diagram

Modules

The architecture for this project will be broken into 3 parts, as per the Model-View-Controller architecture:

1. Model

1. View

1. Controller

Data Flows

Detailed Design

Need:

- Data Formats/Table Layouts
- Code interfaces (i.e. public class interfaces)
  - Method names
  - Post-, pre-, and error conditions
- Optional pseudocode for complex operations

Code Interfaces

Robot Object
int teamNumber
int robotNumber
int pointsLeft
int maxMoves
int movesLeft
int power
int health
int range
bool hasFired
void: move(coords)
void: fire(coords)
Robot: scan(coords)
Stats: getStats()

Match Controller Object
Robot [ ] robots
Cell [ ] cells
int turnNumber
int maxTurns
int currentTeam
int executionSpeed
void: stepForward()
void: stepBack()
Robot: getContentsOfCell(coords)
void: damageRobot(robot)

Views

Main Menu View
void loadWatchMatch()

Main Menu View
void loadInstantResults() *void* loadTestBench()

Team Select View
Team [ ] teams *void* loadTeam(int slot) *void* confirm()

Watch Match View
MatchController controller *void* step() *void* play() *void* stop() *void* setRate(int rate) *void* updateLog(string log) *void* (string log)

Test Bench View (inherits from Watch Match View)
void runCommand()

Match Results View
score [ ] scores *void* returnToMenu