Robosport 370

Design Document

Group C3

Corey Hickson - crh208

Daniel Sanche - drs801

Landon Tetreault - lzt639

Levi Paradis - ldp343

Matthew Wind - mdw466

October 30th, 2015

I. Introduction

Within this document, we have described in detail how we will implement the RoboSport370 project. First off we discussed the different types of architectures that could choose to design our program around. Following the discussion of the architecture we used a matrix diagram to decide which architectures were the most applicable to the project .

II. Requirement Revisions

A. explain why we didn't make changes and why it was right to begin with JSON & JSON Library addition (standards meeting).

B.

III. Architecture Details

A. Decision Matrix:

Our group used a decision matrix to determine which architectures would apply to our project. For the decision matrix we chose a number of attributes as seen below. The primary attributes included applicability and collaborability; these attributes were chosen because of the nature of the project (ie. a group project to develop a game with a GUI). We also looked at modularity, ease of implementation, adaptability, usability, and testability as they were all important attributes for our architectures to properly apply good design principles. Lastly, security was considered, although lowly, due to the non-sensitive nature of the project.

From the decision matrix, it follows the pipes and filters architecture, interpreter architecture and model-view-controller architecture came forth as appropriate possible architectures. During brainstorming, we decided to apply a model-view-controller architecture primarily and interpreter architecture as a portion for the robots.

				Rankings					
	Applicability	**Modularity**	**Ease of Implementation**	**Adaptability**	**Collaborability**	**Useability**	**Security**	**Testability**	**Total**
Weight (1->5)	4	3	3	3	4	3	2	3	
Architectures (1->10)									
Monolithic	2	2	5	2	2	5	3	4	76
Pipes and Filters	7	4	6	6	5	5	4	7	140
Objects & Components	2	7	5	5	5	6	3	5	118
Events	3	4	5	4	4	4	2	5	98
Repositories	2	2	3	4	4	7	7	5	101
Layered	5	2	3	5	3	6	4	5	103
Interpreters	7	4	6	4	4	7	4	6	133
Model-View-Controller	8	7	6	7	7	7	4	7	170

B. Architectural Diagrams

Architectures

Monolithic

pipes/filters

objects/components -

events

repositories -

layered

interpreters - forth interpreter

MVC - system

IV. Classes

A. Class Descriptions

1. Robot:

The class Robot will contain all of the aspects of one robot. This class is meant to contain the health, strength, identifying information, which team it belongs to, and the actions that the robot can perform. The Robot objects will be created at the beginning of the simulation,

and the classes fields will be updated whenever the particular instance of robot is interacted with.

2. Team:

   This is the class in our program that will contain all of the robots that are on one team, and the information about that team. Some of the information included will be the number of wins and losses, the team name, the number of currently alive robots, and a field that determines which robot goes next.

3. GameController:

   Game controller is the class that controls all of the game play. Once the game is started by SetupController this controller takes over and runs all of the simulations. It tells the teams when it is their turn to play a robot and then runs the specified action. This Controller also pauses, resumes, and ends the game, however the game will only be ended when the simulation is complete, and there is a winner. MapView will be used to display the controller's data.

4. SetupController:

   The setup controller is the class that will be used to set up the simulation, this class will set the map size, the teams, and then begin the match. This class also decides which type of match will be run, a GUI match, a headless match, or a debug match. The controller displays all of its data using SetupView.

5. EndController:

   This controller takes over after the simulation has ended. EndController takes data from the LoggerController and calculates what needs to be displayed, which team won the game as well as stats about each team and robot such as the number of shots fired. The data calculated is then displayed using the EndView.

6. DebugGameController:

   Our DebugGameController is what will allow us to move through the robot simulation in either direction. This will allow us to determine what actions were taken and when to help us find errors. We will also be able to change various attributes of the simulation such as robot health, and robot strength. This controller will provided the data that DebugMapView will display.

7. LoggerController:

   This class is used to store all of the data from events during the simulation. It will be used to run debugging view and provide information to the EndController. The logger will track events such as shots fired, deaths, and movement.

8. JSONInterpreter:

   The JSON Interpreter is what we will be using to translate between Java and JSON. We will use this interpreter to read data into

each robot object in the simulation. The data will be read in as JSON and then translated into Java. We will also be able to take the robot data and translate it back into JSON.

9. Fourth Interpreter:
   This interpreter will be used to set up each robot object using the data the we are provided in the fourth language. The data provided will contain robot attributes such as how it moves. FourthInterpreter will also execute turns for each robot by
10. RobotView:
    This view is used to display each robot on the map, depending on what the robot is doing a different image will be shown.
11. EndView:
    The end view will be used at the end of a simulation to show the final results that were calculated by the end controller.
12. DebugMapView:
    This view is very similar to MapView but will also display buttons to allow reversing through the game and changing the various robots attributes.
13. MapView:
    Map View is what will provide the graphics of the simulation, it will animate all of the movements of the robots and the actions that they perform. This view will be represented using 2D graphics.

B. UML Diagrams
C. Libraries (and descriptions)

Java, open source GUI (game dev) library, JSON

V. Conclusion

Robot()

- preformTurn()
- destroy()
- health
- strength
- name
- serialNumber
- team
- deaths
- hexPosition

Team()
- numberAlive
- nextTurnIdx
- teamName
- robots[]
- wins
- losses

GameController()
- gameEnd()
- startWithTeams(Team[])
- pause()
- resume()
- stepBackwards(numTurns)
- stepForward(numTurns);
- nextTurn()
- nextTeamIdx
- Team[] competitors;
- Logger
- mapView
- hexSize

SetupController()
- -pickRobotFile();
- -setMapSize(newSize);
- -startGUIMatch()
- -startHeadlessMatch()

- Team[] competitors;
- SetupView

EndController()
- -displayStats(Logger);
- -endView

MapView()
SetupView()
EndView()

JsonInterpreter()

- robotFromJson(Json) : Robot
- JsonFromRobot(Robot) : JSON

ForthInterpreter()

-executeTurn(forth, robot);

Logger()

-shotListener()
-deathListener()
-moveListener()

<u>Architectures</u>

Monolithic
pipes/filters
objects/components -
events
repositories -
layered
interpreters -forth interpreter
MVC -system