Robocode Player Part

Robocode Battle:

Robocode is a standalone programming game where robots are developed to play against each other in the battle field. The entire robocode is developed using JAVA and the battle is displayed in Java Swing.

Workflow to start a battle in robocode:

The general approach is to specify the inputs for the battle before the execution of the robocode. The inputs needed for the battle are

1.The robots to engage in battle.

2. Number of rounds.

3. Battle width and Battle height. - The battle width and height are specified inside the code.

4. Boolean to set the battle visible or not. - By default, the battle visibility is set to True.

The above input parameters are specified to a class in the robocode called BattleSpecification.

```
BattleSpecification battleSpec = new
BattleSpecification(numberOfRounds, battlefield, selectedRobots);
```

<u>The problems that are encountered while porting robocode into web application:</u>

1. Java security exception: access denied: While packaging everything to a Jar file and running it using a web browser the JVM doesn’t allowed to show the applet in the browser due to security policies. (Since Eclipse creates its own java security policy while running the applet, it was able to show the battle in the applet)

Changes that are done in Java security policy are:

```
Grant{

Permission java.security.AllPermission;

Permission java.lang.RuntimePermission “create classloader”;
```

Permission java.util.PropertyPermission “NLS_LANG”, “read”;

Permission java.lang.RuntimePermission “createSecurityManager”;

}

2. Java.util.FilePermission : access denied: The robocode is designed in such a way that it loads the images for the battle and robots from the directory (i.e : /resource/images and /robots/sample/ etc). Since the whole robocode is packaged into a single jar, the applet are restricted to access local filesystem, executable files, system clipboard, and printers.

Changes made to overcome this:

ImageUtil.java – It is a java file that reads all the images needed for the battle field and robots.

- Converted the local file system path to an URL path.

  Code changes:

  ```
  public static Image getImage(String filename) {

  url = new URL(get_file_path + filename);

  }
  ```

- Changed the sandbox applet to privileged applet to allow it read local file system

3. ClassNotfound exception: The BattleManager will pass the robots to RobotClassManager to create an instance for each robots to play in the battle field. Before creating the instance, the robot class loader function is called to load the class from the robot class path. Since robot class path is not set since all the resources are packaged into a single jar therefore the compiler thrown the class not found exception.

Modifications done in the robocode:

- Created a new function inside the robocode class loader that loads the robots specified as input to the program.

```
	public synchronized Class<?> loadRobotClass(String name, boolean toplevel)
throws ClassNotFoundException {
		if (cachedClasses.containsKey(name)) {
			return cachedClasses.get(name);
		}
		Class<?> c = null;
		if (toplevel) {
			uid1 = 0;
			uid2 = 0;
		}
		robotClassManager.addResolvedClass(name);
		if (name.equals(robotClassManager.getFullClassName())) {
			if (robotClassManager.getRootPackage() == null) {
				rootPackageDirectory = null;
				classDirectory = null;
			} else {
				//rootPackageDirectory = new File(classPath +
File.separator + robotClassManager.getRootPackage() +
File.separator).getCanonicalPath();
				//classDirectory = new File(classPath + File.separator +
robotClassManager.getClassNameManager().getFullPackage().replace('.',
File.separatorChar) + File.separator).getCanonicalPath();
			}
			//rootDirectory = new File(classPath).getCanonicalPath();
		}
		if (toplevel) {
			robotClassManager.loadUnresolvedClasses();
			robotClassManager.setUid(uid1 + "" + uid2);
		}

		cachedClasses.put(name, c);
		return Class.forName(name);
```

-
  ```
  	}
  ```
- Wherever the file operations are performed, those functions are removed from the robocode. The constructor of the robotspecification is changed to accept only the robot name and robot.

  ```
  public RobotSpecification() {   }
  ```
- The robot name is set to a member variable called "name" in RobotSpecification. For that, a getter and setter functions are created.

```
•	public String get_robot_name() {
•		return name;
•	}
•
•	public void set_robot_name(String name) {
•		this.name = name;
•	}
•
```

- Similarly battle manager is also changed to create the instance of robotspecification.
- `RobotSpecification robotSpec = new RobotSpecification();`
- `robotSpec.set_robot_name(bot);`

The code that creates an instance to play the game as a “Team” are removed to simplify the logic in creating a new battle in BattleManager.java

Code that are removed from the robocode that throw unnecessary errors and access denied are:

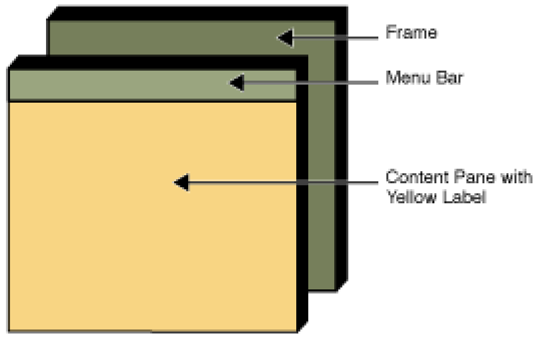
- RobotRepositoryManager: getRobotCache() : Removed all the code to return only the robotdirectory.
- RobocodeManager: getProperties(): All the file operations are removed.
- ImageManager: GetGroundTileImage(): Through classloader.class.getResource(filename) the images are loaded in runtime from ImageUtil. This is changed by directly calling the ImageUtil.getImage function.
- BattleManager: StartNewBattle: code related to playing the robocode as “Team” are removed.
- BattleManager: StartNewBattle(): All the security exceptions are removed since it causes access denied error while running applet in the browser.

After the first round in a battle between two or multiple robots, the battle gets hanged due to some unknown wait time imposed on the robots. Those wait times are removed from the function called: Wakeuprobots(). There was two for-loop that takes creates wait time of 99999 for each robot during each round. To avoid such bigger waiting time for each robot, the wait time is reduced to cpuconstanttime%100000.

Code change:

```
if (!r.isSleeping()) {

      try {

            long waitTime = manager.getCpuManager().getCpuConstant();


            int millisWait = (int) (waitTime / 1000000);

            if (!r.isSleeping()) {
                r.wait(0, (int) (waitTime % 1000000));
            }
        } catch (InterruptedException e) {
            // ?
            log("Wait for " + r + " interrupted.");
        }
    }
```

<u>Swing in Robocode:</u>



A swing window is created by extending a class called JFrame. A Frame is a top-level window with a title and a border. JFrame is an extended version of Swing which supports Swing component architecture. JFrame consists of menu bar and content pane. The content pane contains all the non-menu components displayed by the JFrame.

In Robocode, the RobocodeFrame is the main class which controls all the swing activities in the playing the battle between the robots.

One of its member function of RobocodeFrame is:

```
    private JPanel getRobocodeContentPane() {
        if (robocodeContentPane == null) {
            robocodeContentPane = new JPanel();
            robocodeContentPane.setLayout(new BorderLayout());
```

```
                robocodeContentPane.add(getToolBar(), "South");

                robocodeContentPane.add(getMainPanel(), "Center");

           }

           return robocodeContentPane;

     }
```

This getRobocodeContentPane provides the entire battle window of the Robocode.

Java Applets:

It is a special kind of Java program that a browser enabled with a java technology can download from the internet and run. If an applet wants use any of the Swing's features then it should be a subclass of Japplet: javax.swing.JApplet.

The project is to create an applet for Robocode to display the battle in the browser window. By creating an applet class that extends the JApplet, we can bring the swing-based graphical user interface of the robocode into java applet.

The lifecycle of an applet are :

Init, start, stop and destroy. These methods are invoked automatically by the browser.

Init method is like the constructor of the Java applet. This will instantiate the swing classes of the robocode and will set applet's content pane to robocode's content pane.

This is achieved by making few modifications in the robocode to get its content pane.

The program flow from Robocode engine to Robocode's content pane:

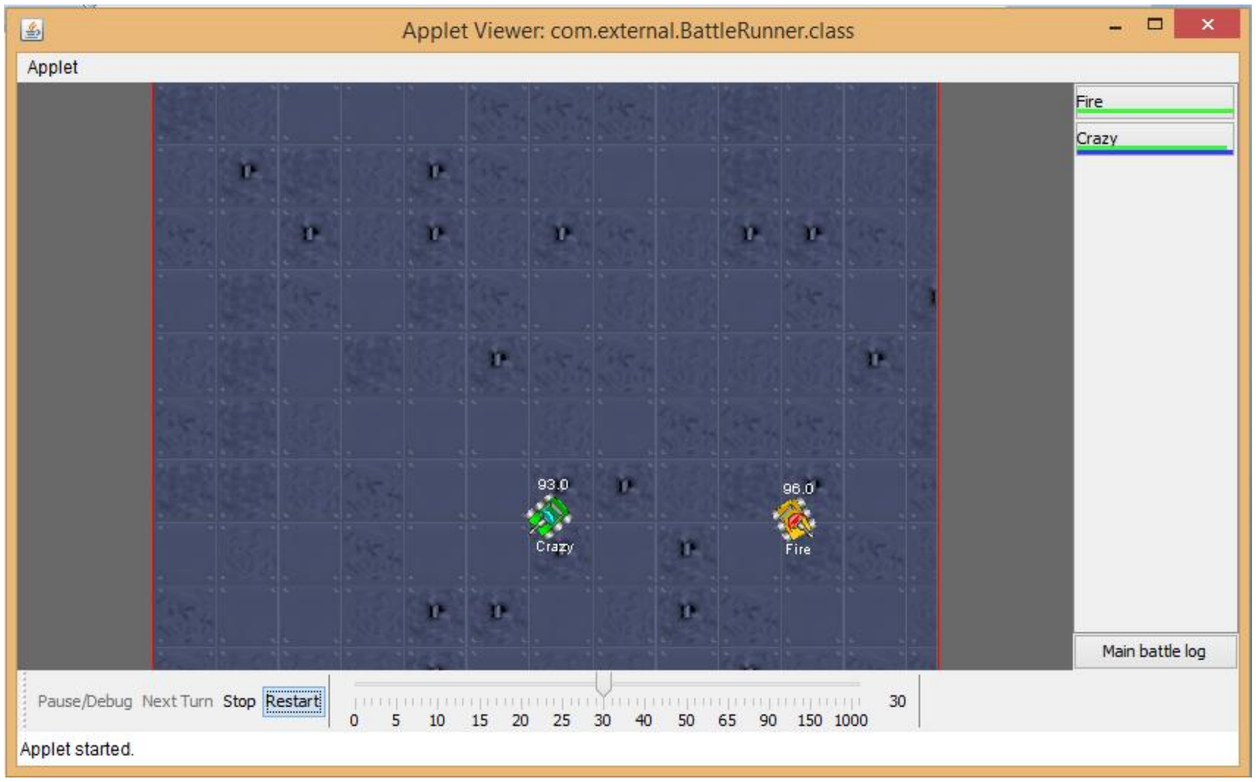
RobocodeManager -> Set Visible(true) -> Window Manger's setVisibleForRobotEngine -> ShowRobocodeFrame -> GetRobocodeFrame.

The RobocodeManager calls the function Setvisible(True) to set the battle window to true to display it to the user. This in-turn creates the window for the battle using WindowManager class and calls the function SetVisibleForRobotEngine, ShowRobocodeFrame and GetRobocodeFrame.

Modification in the above Robocode work flow:

A member variable is created in RobocodeFrame to store the content pane of the battle window and it is passed to the RobocodeManager. Further, it is returned to the applet class to set it to its content pane.

Applet window displaying the battle in the appletviewer:



Applet started.

Deploying an Applet:

To develop an Java Applet:

1. Compile your Code.

2. Package it as JAR file.

3. Sign the JAR file.

Packaging it as JAR File:

A text file that contains the JAR file manifest attributes is needed to create a Jar file.

Permissions: all-permissions

Codebase: *

Application-Name: Applet Name

Command to create the Jar file:

```
Jar –cvfm robocode.jar mymanifest.txt robots robocode
```

The folders resources, robots and robocode contains the class files of the robocode project, images of the battle and robot class files.

Sign the JAR File:

The command to create Jar file is:

```
> keystore –genkey –alias robocode

> keystore -selfcert

> jarsigner robocode.jar robocode
```

If the jar created is not signed then it will be treated as unsecured jar file and will not be showed in the browser window. Either the jar file link must be added to the Java's control panel security list to exempt it from blocking from browser or it should be signed.

Further changes in the Robocode:

The input parameters are passed to the applet using the HTML tags called : <Param>

```
     <param name=Robots value="sample.Fire,sample.Corners">

     <param name=numberofrounds value=5>
```

The input specified in the param tags are received in the implementation part using the built-in function of JApplet called getParameter()

code:

```
battleProperties.setNumRounds(Integer.parseInt(getParameter("numberofrounds")));
battleProperties.setSelectedRobots(getParameter("Robots"));
```

Results of the battle:

The battle results are fetched from the applet using a javascript function getResult(). A javascript function is invoked by clicking the "submit" button in the HTML file which in-turn calls the getResult() function returned in the robocode.java. At the end of the battle results will be stored in the class called BattleResultsTableModel. The values are collected and stored in a member variable "results_data" in the battle manager class.

```
	public void printResultsData(Battle battle) {
		BattleResultsTableModel resultsTable = new
		 BattleResultsTableModel(battle);
		ByteArrayOutputStream os = new ByteArrayOutputStream();
		PrintStream ps = new PrintStream(os);
		resultsTable.print(ps);
		try {
			this.results_data = os.toString("UTF8");
		} catch (UnsupportedEncodingException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

	 }
```

getResult function in the Robocode.java file:

```
  public String getResult()
  {
    System.out.println("get frist name");
    this.manager.getBattleManager();
    System.out.println("manager.getBattleManager().results_data" +
BattleManager.results_data);
    this.manager.getBattleManager();
    String[] splited = BattleManager.results_data.split("\\s+");
    System.out.println("The splitted string is" + Arrays.toString(splited));
    int i = 0;
    String each_robot_result = "";
    for (i = 0; i < splited.length; i++) {
      if (splited[i].contains("."))
      {
        each_robot_result = each_robot_result + splited[i] + ":" + splited[(i + 1)] +
" ";
        System.out.println("The each robot result is" + each_robot_result);
      }
    }
    return each_robot_result;
  }
```

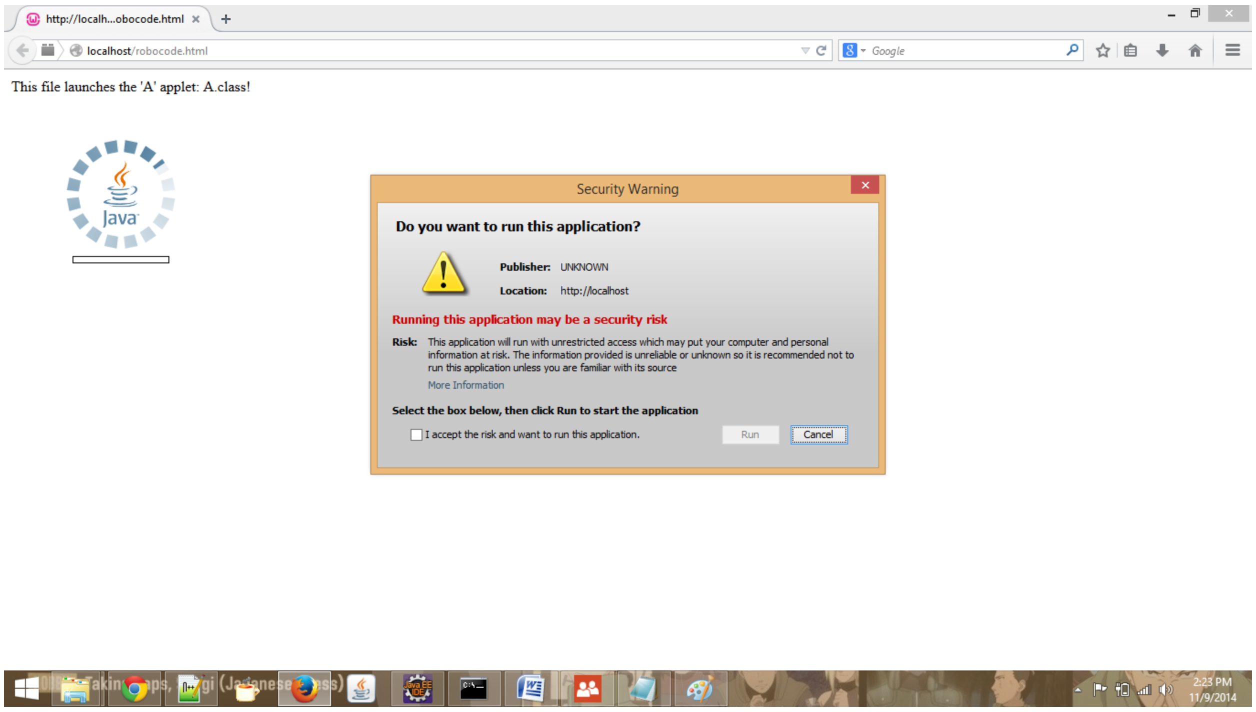
The complete information about the robots played in the battle is collected and stored while the applet is destroyed.

```
public void stop()
  {
    System.out.println("In stopping of the rbocode applet");
```

```
this.manager.getBattleManager();System.out.println("manager.getBattleManager().result
s_data" + BattleManager.results_data);
    this.manager.getBattleManager();String[] splited =
BattleManager.results_data.split("\\s+");
    System.out.println("The splitted string is" + Arrays.toString(splited));
    String results = Arrays.toString(splited);
    String[] result_column_names = { "Rank", "Robot_Name", "Total_Score", "Survival",
"Surv_Bonus", "Bullet_Dmg", "Bullet_Bonus", "Ram_Dmg_*_2", "Ram_Bonus", "1sts",
"2nds", "3rds" };
    int i = 0;
    boolean robots_flag = false;
    String final_robot_result = "";
    String each_robot_result = "";
    for (i = 0; i < splited.length; i++)
    {
      if (splited[i].contains(":"))
      {
        each_robot_result = "";

        each_robot_result = each_robot_result + result_column_names[0] + ":" +
splited[i].split(":")[0] + " ";
      }
      if (splited[i].contains("."))
      {
        int j = 1;
        each_robot_result = each_robot_result + result_column_names[j] + ":" +
splited[i] + " ";


        i++;
        j++;
        while (i < splited.length)
        {
          if ((splited[i].contains(".")) || (splited[i].contains(":"))) {
            break;
          }
          each_robot_result = each_robot_result + result_column_names[j] + ":" +
splited[i] + " ";
          if (i + 1 >= splited.length) {
            break;
          }
          if (splited[(i + 1)].contains(":")) {
            break;
          }
          i++;








          j++;
        }
        final_robot_result = final_robot_result + each_robot_result;
      }
    }
  }
```

Screenshots:

Initial Page:



http://localh...obocode.html

localhost/robocode.html

This file launches the 'A' applet: A.class!

Security Warning

Do you want to run this application?

Publisher: UNKNOWN

Location: http://localhost

Running this application may be a security risk

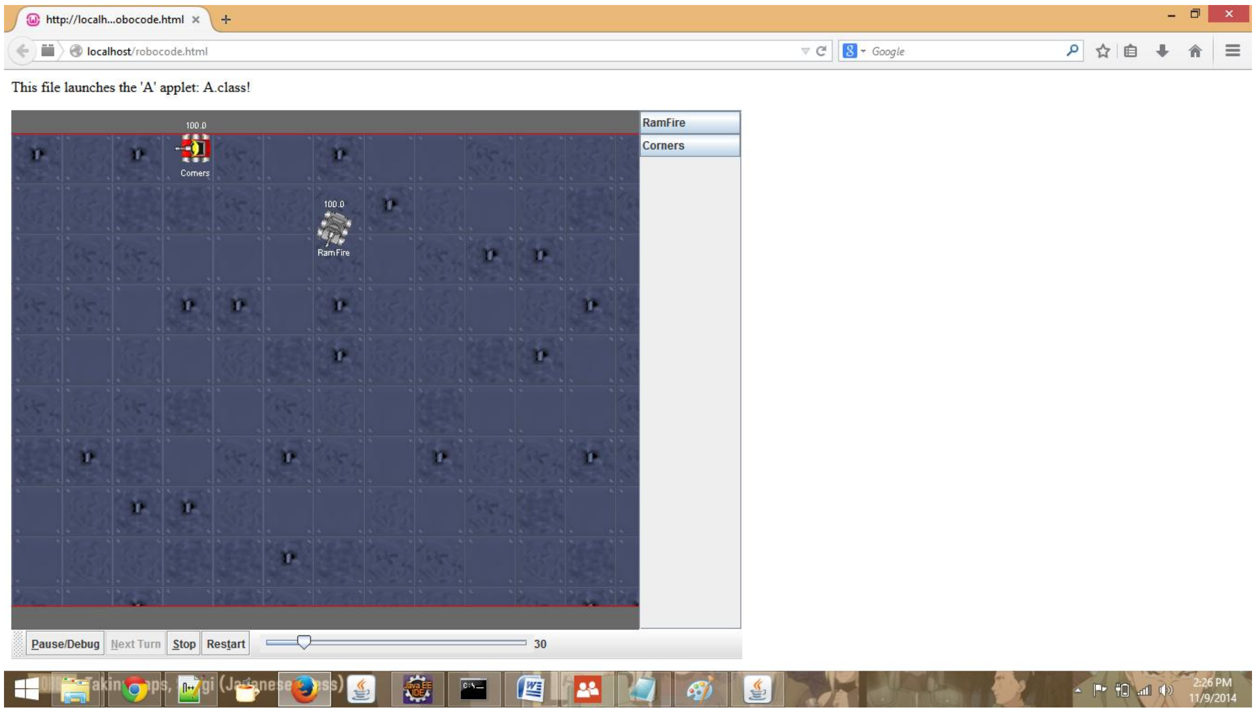
Risk: This application will run with unrestricted access which may put your computer and personal information at risk. The information provided is unreliable or unknown so it is recommended not to run this application unless you are familiar with its source

More Information

Select the box below, then click Run to start the application

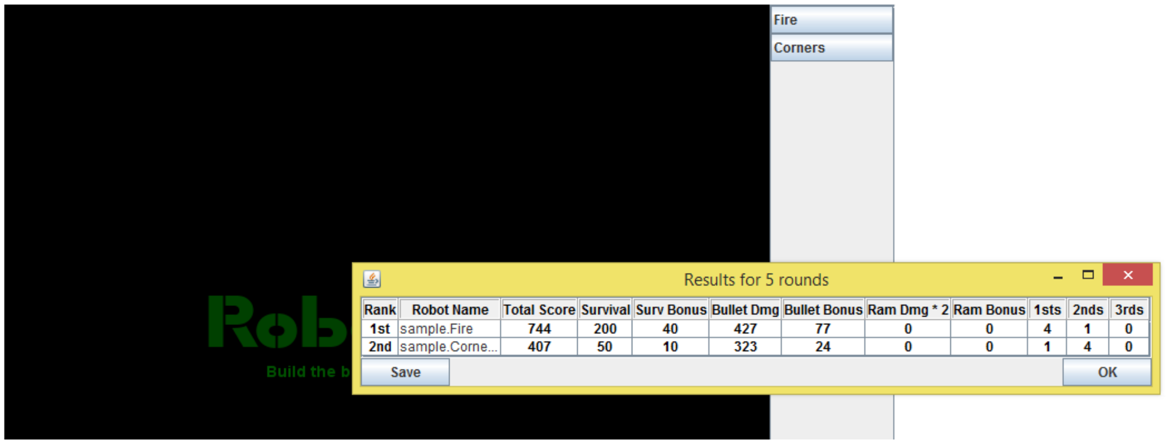
☐ I accept the risk and want to run this application.

Battle Page:



This file launches the 'A' applet: A.class!

Result page:



Results for 5 rounds

Rank	Robot Name	Total Score	Survival	Surv Bonus	Bullet Dmg	Bullet Bonus	Ram Dmg * 2	Ram Bonus	1sts	2nds	3rds
1st	sample.Fire	744	200	40	427	77	0	0	4	1	0
2nd	sample.Corne...	407	50	10	323	24	0	0	1	4	0

Save OK

<u>Notes:</u>

The signed jar of robocode applet is created only once and placed fused with the project. The robocode applet runs for all the robots that users creates. The robocode applet is not needed to be compiled and signed everytime for a new battle. Moreover, the project doesnot contain any Java source file of robocode since all the class files of the robocode are bundled into a jar file.

The robots already created and user created new robots will be placed in the same directory of the robocode applet. The robots will be placed in a folder that has the same name as their domain name.

The images needed to load for battle background and robots images are stored in a folder called "resources" and it should be placed in the same directory of robocode.jar

The restart button is disabled from the applet since user may violate the rule by clicking the restart button while the battle is on.

How to Enable the Player Part to play the battle:

*The player part requires an java-enabled browser to play the battle between the robots.

*An applet will be loaded into the browser to play the battle between the robots.

*Sometimes security warnings will be popped up in the browser window and it is advised to run/unblock the application for the execution of the battle.

Steps to enable the applet in the browser:

*The cloud link should be added to the exception list of the java configure window.

Start->programs->Java Control Panel->Security->Edit Site List->Add the cloud link and click apply.

Then go to the advanced tab to check "show console" to debug the applet if it fails to load in the browser.

Robots Ranking:

Algorithm to calculate the battle points between multiple robots:

The project uses an algorithm called Elo's rating to calculate the points obtained by the robots after playing the game.

Elo's Rating:

- The Elo rating system is a method used for calculating the relative skill levels of players in competitor-versus-competitor games.
- A player's Elo rating is represented by a number which increases or decreases based upon the outcome of games between rated players.

- After every game, the winning player takes points from the losing one.
- The difference between the ratings of the winner and loser determines the total number of points gained or lost after a game.
- In a series of games between a high-rated player and a low-rated player, the high-rated player is expected to score more wins.
- If the high-rated player wins, then only a few rating points will be taken from the low-rated player.
- However, if the lower rated player scores an upset win, many rating points will be transferred.
- The lower rated player will also gain a few points from the higher rated player in the event of a draw. {Source: WIKIPEDIA}

<u>Calculation on each robots final rating based on Elo's algorithm:</u>

*Let r1 and r2 be the aggregation of overall ratings of the robots S1 and S2. Initially, the rx = 100.

The First step is to compute the transformed rating for each robot based on the current rating rx.

R(x) = 10r(x)/400

Where R(x) = Transformed rating of the robot x

r(x) - current rating of the robot x

*Now calculate the expected score of each robot.

The expected score fo a robot is conputed as : E(X) = R(X) / (R(1) + ... + R(X))

*If a robot A wins against robot B then S(A) = 1 and S(B) = 0

*Now compute the final rating of a robot as: R'(X) = R(x) + K * (S(X) - E(X))

here K is a constant factor and it is initialized to 32.

Example:

Let robot A and robot B compete each other.

The current rating of robot A is 2400 and current rating of robot B is 2000 .Now the expected score of robot A is 0.91 and expected score of robot B is 0.9.

If robot A wins the game then S(A) = 1 and S(B) = 0

The final rating of robot A and B are : R'(A) = 2403 and R'(B) = 1997

Similarly If Robot B wins the game

then S(B) = 1 and S(A) = 0

The final rating would be R'(A) = 2371 and R'(B) = 2029