# SDP Group 8: Final Individual Report

Blake Hawkins

21 April, 2014

Mentor: Katharina Heil, Guest Mentor: Tom Spink
Members: Blake Hawkins, Lubomir Vikev, Borislav Ikonomov, Yordan Stoyanov, James Linehan, Lukas Dirzys, Iain Brown, Emanuel Martinov, Robaidh Mackinnon, Aneesh Ghosh

## 1 Introduction

My contributions to our project spanned many topics, each of which will have a section in this report, where I describe them and offer commentary. For brevity, and to meet the two-page limit, many contributions will be left out, and most that were mentioned in previous milestones will be cut short.

## 2 Software Design Contributions

My recent works in software design include a major refactoring to our monolithic **Strategy** class, in which I split it into two more cohesive classes: **Strategy**, which runs our GUI components and vision/world state; and **StrategyThread**, which is an interruptible state machine that keeps both robots updated with the latest commands. This refactoring also opened our strategy system to unit tests by completely eliminating *static* references. Additionally, I wrote a total of 19 unit tests[1] related to ball prediction, our **minorHull()** algorithm, and various vector mathematics methods.

I conducted many smaller refactorings, wrote the majority of our project's JavaDocs, and made the initial transition from Milestone 3 to our **Strategy/Robot** class pair, thereby successfully providing abstract commands to both the attacker and defender. Further, I fixed many regressions left over by large refactorings written by others.

In relation to code re-usability, I believe these contributions are the most significant, because having feature-full code is less useful for SDP 2015 than having well-maintained, documented, and easily extendible code. I am pleased with the time and effort I spent on JavaDoc and refactorings, which I believe to be reasonable and efficient.

## 3 Physics Simulation Contributions

I wrote a large ball prediction system which supported the ball reflecting[2] off multiple boundary walls, provided abstract values for coefficients of restitution and friction, and distinguished goal lines from normal walls.

During milestone 1 I also wrote an odometry module which allowed us to estimate our robot location after an arbitrary distance traveled, and any given number of turns. However, due to imprecision in our robot, the system was less accurate than a much simpler solution involving timers.

The ball prediction system is probably my single biggest contribution to the codebase, which I believe to be full-featured, well-made, and useful. It works by calculating an estimated total distance travelled by a ball using geometric series, and then iterating pixel by pixel looking for walls and re-calculating. The time complexity of the system is therefore linear in distance (assuming no walls struck) and could be improved by using a line-intersection algorithm (constant time complexity in distance, assuming no walls struck). However, due in part to the simplicity of the system and the nature of the coefficient of restitution feature, we never experienced performance issues with it.

## 4 Project Management Contributions

I acted as a project manager in cases where having one was academically necessary, but our group mostly worked cohesively, with everyone administering their own tasks and availability. I provided some additional support by partially managing our issue tracker and writing tutorials[3] on our team wiki.

I also used paired programming on some occasions as a way of staying on track with everyone's activities, and found the process surprisingly useful and efficient. In addition, I worked on other tasks like presentation design and speech planning.

Whether having no official group leader would become a problem was a gamble I took based on our performance in the first few weeks of development. We had so many meaningful contributions and active members that I believed changing anything would only cause problems. Even now, I doubt having myself as a final "shot-caller" would make a significant difference in our team's performance. I also appreciated splitting administrative tasks with other members as it allowed me to contribute more in software.

## 5 Strategy Contributions

I spent a large fraction of my time developing 20+ abstract methods for our robot pair to execute strategy, for example **kickBallToPoint()**, which provided all the functionality to go to and grab a ball, face a point and shoot; and **goToFast()**[4], which makes a robot go to a point as fast as possible, driving forwards or backwards to minimise delay. I also spent a lot of time developing a method for projecting ball prediction data onto a robot's facing axis, in order to efficiently find an intersection point for the robot to move when blocking balls, passing, and estimating data from opposing robots facing angles.

I also introduced our team's first non-blocking, state-based strategy design during milestone 3, which was used all the way until our final match.

I believe most of the work I did on strategy code could have been just as well or even better by others, but I feel that my direct and immediate contributions helped ignite the creativity of other members to contribute - having an existing, testable strategy codebase is much less intimidating than having the task of making it yourself.

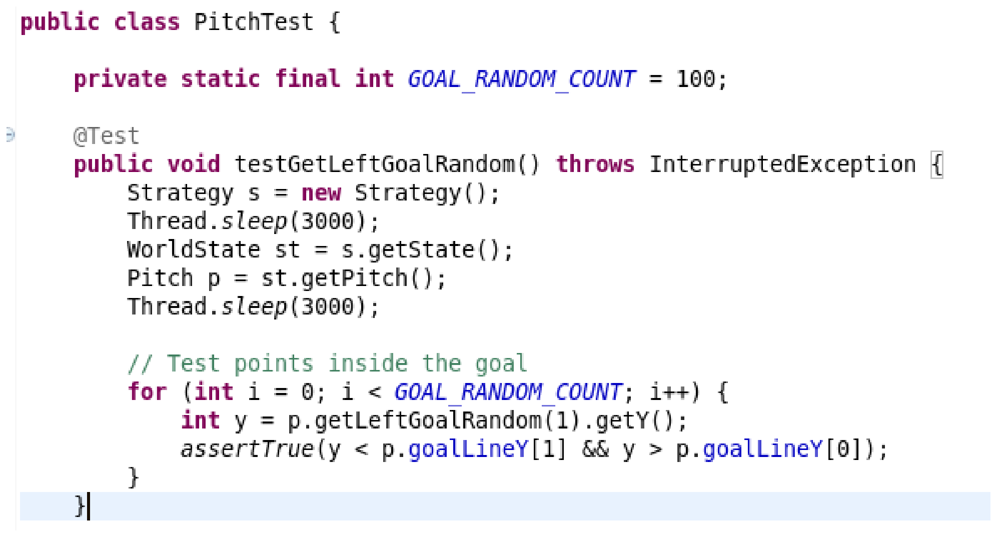
## 6 Conclusion

I am pleased with my contributions, which I believe to be the right mixture of power and usefulness, easy to understand and maintain for other members and future developers, and efficient in terms of my personal strengths. I am also pleased that we performed well in matches, and believe under other conditions we could have easily improved our final performance to second or even first place.

## 7 Appendix

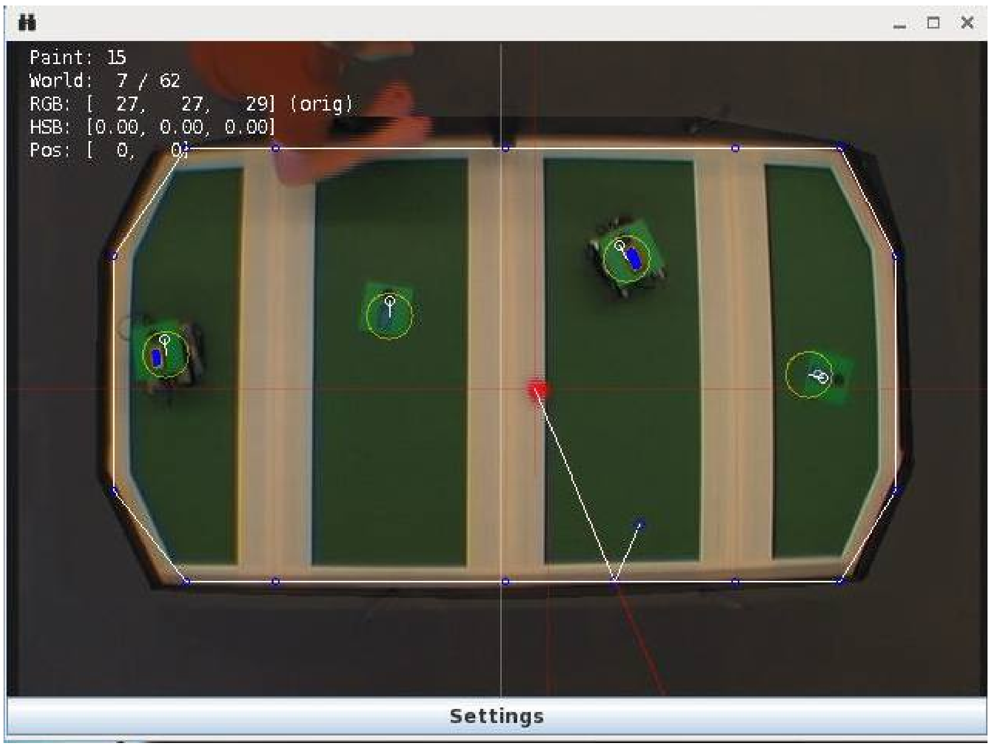
This section contains screen captures related to my works.

### 7.1 Figure 1: A typical Unit Test



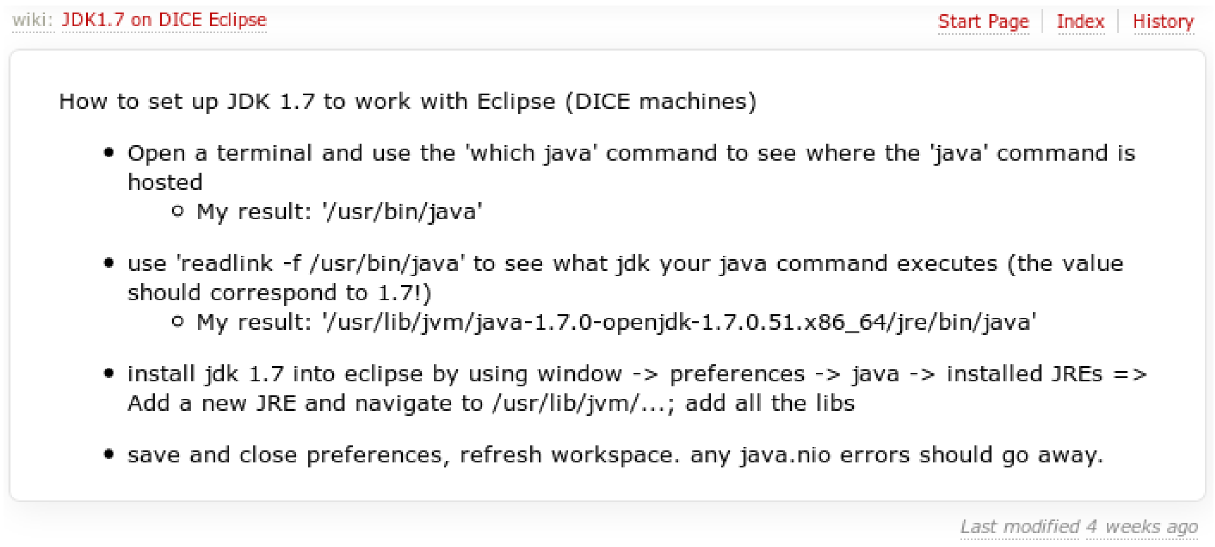
```
public class PitchTest {

    private static final int GOAL_RANDOM_COUNT = 100;

    @Test
    public void testGetLeftGoalRandom() throws InterruptedException {
        Strategy s = new Strategy();
        Thread.sleep(3000);
        WorldState st = s.getState();
        Pitch p = st.getPitch();
        Thread.sleep(3000);

        // Test points inside the goal
        for (int i = 0; i < GOAL_RANDOM_COUNT; i++) {
            int y = p.getLeftGoalRandom(1).getY();
            assertTrue(y < p.goalLineY[1] && y > p.goalLineY[0]);
        }
    }
```

### 7.2 Figure 2: Ball Prediction working, showing reflection



```
Paint: 15
World:  7 / 62
RGB: [  27,   27,   29] (orig)
HSB: [0.00, 0.00, 0.00]
Pos: [   0,    0]
```

## 7.3 Figure 3: Example of a tutorial on our wiki



How to set up JDK 1.7 to work with Eclipse (DICE machines)

- Open a terminal and use the 'which java' command to see where the 'java' command is hosted
    - My result: '/usr/bin/java'
- use 'readlink -f /usr/bin/java' to see what jdk your java command executes (the value should correspond to 1.7!)
    - My result: '/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.51.x86_64/jre/bin/java'
- install jdk 1.7 into eclipse by using window -> preferences -> java -> installed JREs => Add a new JRE and navigate to /usr/lib/jvm/...; add all the libs
- save and close preferences, refresh workspace. any java.nio errors should go away.

*Last modified 4 weeks ago*

## 7.4 Figure 4: Abstract Robot method: goToFast()

```java
/**
 * A method which signals the robot to go to a point the fastest way
 * possible. Checks the facing angle of the robot and decides to go forwards
 * or backwards to get there.
 *
 * @param to
 * @param epsilon
 * @return
 * @throws Exception
 */
public boolean goToFast(Point2 to, double epsilon) throws Exception {

    // Get the necessary angles
    double angleToPoint = normalizeToUnitDegrees(state.getRobotPosition(
            myTeam, myIdentifier).angleTo(to));
    double facing = normalizeToUnitDegrees(state.getRobotFacing(myTeam,
            myIdentifier));

    // Compare them
    double diff = normalizeToUnitDegrees(facing - angleToPoint);
    if (Math.abs(diff) > 90.0) {
        return goToReverse(to, epsilon);
    }
    return goTo(to, epsilon);
}
```