PERIOD TWO – CODE REVIEW

INTRODUCTION

The code we are reviewing will be for the LEGO bot that will be used to sweep the people and the poms around as well as carrying the supplies using a lift mechanism around carefully. The code is made by Satya and Shahryar. It is then reviewed by Hagar on 01/18/2019.

BEST PRACTICES CHECKLIST

Checklist	**Done**
Code uses functions to organize code	Yes
Code includes comments documenting each function's purpose	Yes
Code includes comments documenting each function's arguments and return values.	No
Variable names in the reviewed code are descriptive and convey their use in the program.	Yes
Code generally avoids the use of unnamed numeric constants other than 0, 1, or 2.	No
Code is appropriately formatted to show flow of control.	Yes
Use of comments do not contain blocks of old code that is no longer in use.	Yes

At the moment, we currently use functions the same way as we use subroutines with no return values. We also did not use arguments for the functions but will do later on once we combine all the functions because we are still testing each function separately and thus we declared our variables and constants within the function itself. Our motor power values are unnamed because they we feel that it is easier to leave it as an unnamed constant rather than making a variable for it as it would make the code less messy as we have to make different constants for different turns. To make up for the lack of unnamed constants, we decided to add comments next to the different unnamed constants, explaining why they are the way they are.

GENERAL CODE ANALYSIS

RELIABILITY:

At this current stage, we are using data from the sensor and specific functions on what to do depending on the condition for example, if the proximity sensor detects something for the first time, the bot will have a rough idea of where it currently is on the game board. We found this to be reliable to some degree but we are not fully confident about the program and this is because we rely on the top hat sensor and we found that the values for different shades of white/gray/black differs a lot depending on the environment (mainly the shadows) and hence we are improving on that by taking more readings and placing the sensor in different parts of the robot.

For the program to be more reliable and rely less on the sensors, we plan to make a function that would determine the distance travelled by the bot and to do this, we will use the number of ticks it takes to do

a full rotation of the wheels and from there we would find the distance travelled in one full rotation. This will allow the bot to get a better idea of where it's currently at on the gameboard.

MAINTAINABILITY:

We constantly review our code and peer review each other's code in case any of us were to make a typo or forgot to comment their code. Most of our codes is broken down into functions which makes it really easy to copy/paste the code and use it elsewhere. After one of us made a function, we perform extensive testing on the program to see how it holds up and from there we could improve our code. Our code is filled with effective concise comments which are descriptive, so when the code is given to another teammate, he/she will know what the code is supposed to be performing which makes modifications by other members a lot easier. Before we leave a session, we backup all the codes that were made on that day to the cloud (for example: google drive) and have it shared among us so that any of us could access the codes or program when they have internet.

The maintainability of our code could be improved by having the author of the code clearly defined at the top or in the wallaby so that it would be easier to find the coder and ask him/her to explain a section of the code more clearly or to modify it. Furthermore, we will start making a short update log on a code if any of us were to make any modifications so that it will be easy to identify if that code is the latest version.

EFFECTIVENESS:

The code performs the task at a satisfactory level as our coder rushed and made a lot of unnecessary variables that were not used during the run.

SCREENSHOTS OF OUR CODE:

```
#include <kipr/botball.h>
int main()
{
    int leftmotor = 0;
    int rightmotor =1;
    int tophatright = 1; //tophat that detects white will be connected to analog port 1
    int tophatleft = 0;
    int proximity = 2;
    int white = 3400; //max white
    int black = 3900; //min black
    int greylower = 3500 ; //lower bound of grey
    int greyupper = 3850; //upper bound of grey
    while (analog(proximity) > 200 )
    {
while ((tophatright >greyupper) && (tophatleft < greyupper) && (tophatleft > greylower))
{motor_power(leftmotor, 100); //moving
motor_power(rightmotor, 95);
}
//going right
  while ((analog(tophatright) < black)  && (analog(tophatleft) > greyupper))
    {motor_power(leftmotor, 500);
     motor_power(rightmotor, 100);
        }
  //going left
   while ((analog(tophatright) < black) && (analog(tophatleft) < greylower))
    {
       motor_power(leftmotor, 100);
      motor_power(rightmotor, 50);
        }
    }
```

To improve the code, we gave the coder 1 hour as one day wasn't enough for him. The one hour he spent on doing the code was supervised by one of the general helpers. The coder has a severe case of procrastination. The result is that he used two top hat sensors to follow a black line even though only one is needed. Another coder later on modified his code so that only one top hat is required for the code to work. The originally coder also left most comments out and added misleading comments

making it hard to fathom by other members.

```c
#include <kipr/botball.h>
/* Following_black_line_onetophat by Satya 16/01/2019
This function will allow the bot to follow the black lines on the middle of the game board using only one tophat sensor.
This assumes the bot is following the black line from the right side of the board to the left*/
int main()
{
    int leftmotor = 0; //The port to control the left motor
    int rightmotor =1; //The port to control the right motor
    int tophat = 0; //tophat that detects white will be connected to analog port 1
    int proximity = 1; //port for proximity sensor
    int greylower = 3500 ; //Lowest grey value
    int greyupper = 3850; //Highest grey value

    while (analog(proximity) > 200) //will stop if it detects something infront of the bot
    {
        //while tophat detects black, the bot will keep moving
        while (analog(tophat) > greyupper)
        {
            motor_power(leftmotor, 100) ;
            motor_power(rightmotor, 95); //The right motor is slightly more powerful than the left hence the forward value is lower than the left.
        }

       //If tophatright is not detecting black and detects grey instead, then it's slightly off course to the left.
       while ((analog(tophat) > greylower) && (analog(tophat) < greyupper))
        {
            motor_power(leftmotor, 100);
            motor_power(rightmotor, 50); //lowering the power for the right motor so that the bot would turn right.
         } //Continues to move right until tophat no longer detects grey

       //If tophatright is not detecting black and detects white instead, then it's slightly off course to the right.
       while (analog(tophat) < greylower) //anything below the lowest grey will be considered white
        {
            motor_power(leftmotor, 50); //lowering the power for the left motor so that the bot would turn left.
            motor_power(rightmotor, 100);
        } //continues to move left until tophat no longer detects white
    }
    ao();
```