

CPEN 502

Assignment 2 Report

Table of Contents

Question 2a.....	1
Question 2b.....	2
Question 2c	3
Question 3a.....	4
Appendix for source code	6

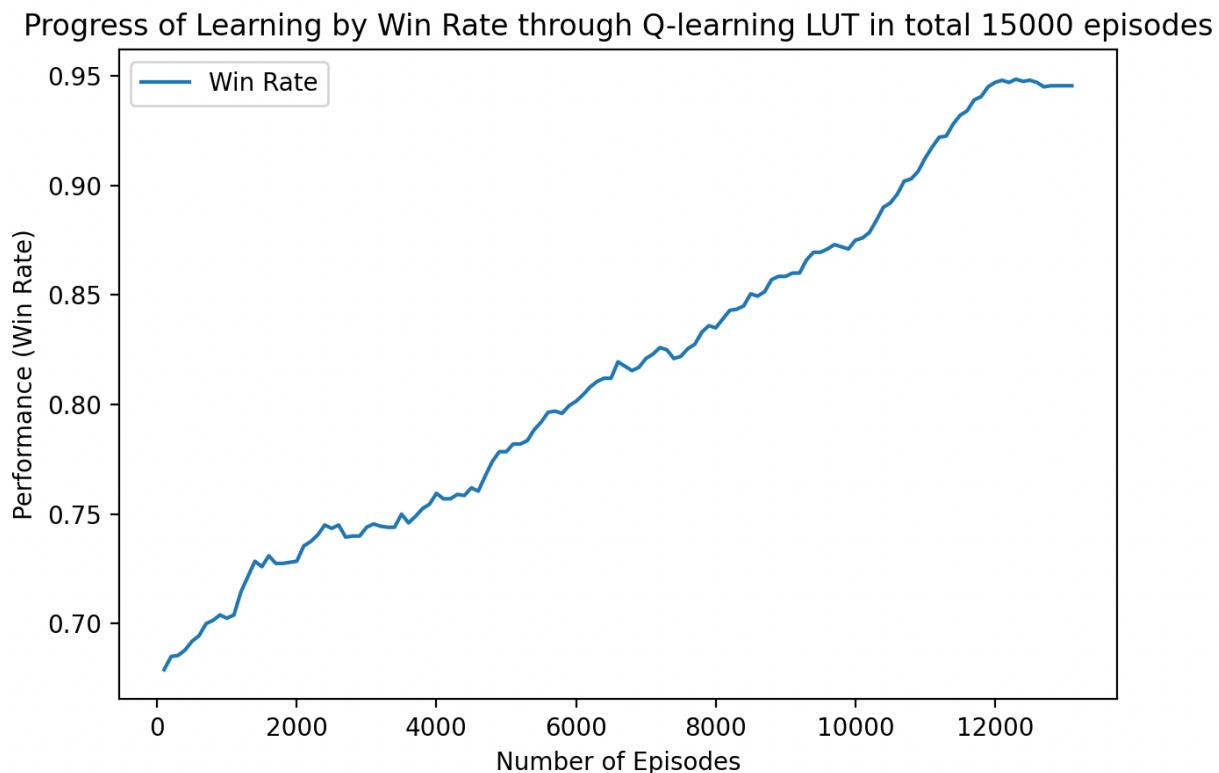
Question 2a

(2) Once you have your robot working, measure its learning performance as follows:

a) Draw a graph of a parameter that reflects a measure of progress of learning and comment on the convergence of learning of your robot.

I chose win rate (number of winning rounds in every 100 episodes) to indicate the progress of learning. As the graph below shows, the win rate increases as the number of episodes increases from 0 to 15000. At around 12000 episodes, the win rate performance seems to converge to around 0.95.

Note: In order to better show the trend, moving average is used to smooth out fluctuations in data, which is likely the reason why the x slightly shifted.

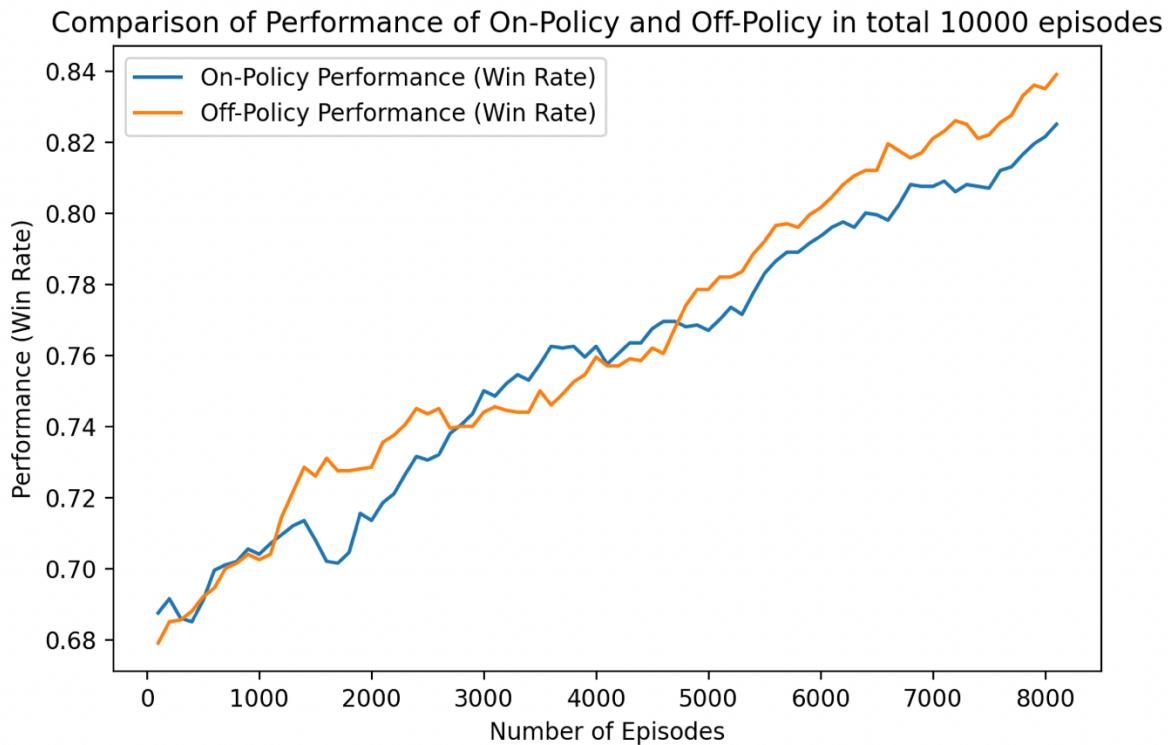


Question 2b

b) Using your robot, show a graph comparing the performance of your robot using on-policy learning vs off-policy learning.

I compared the performance (using win rate in every 100 episodes) using on-policy learning vs off-policy learning in 10000 episodes. As the graph below shows, the off-policy learning seems ultimately better than on-policy training when the number of episodes is relatively large.

Note: In order to better show the trend, moving average is used to smooth out fluctuations in data, which is likely the reason why the x slightly shifted.

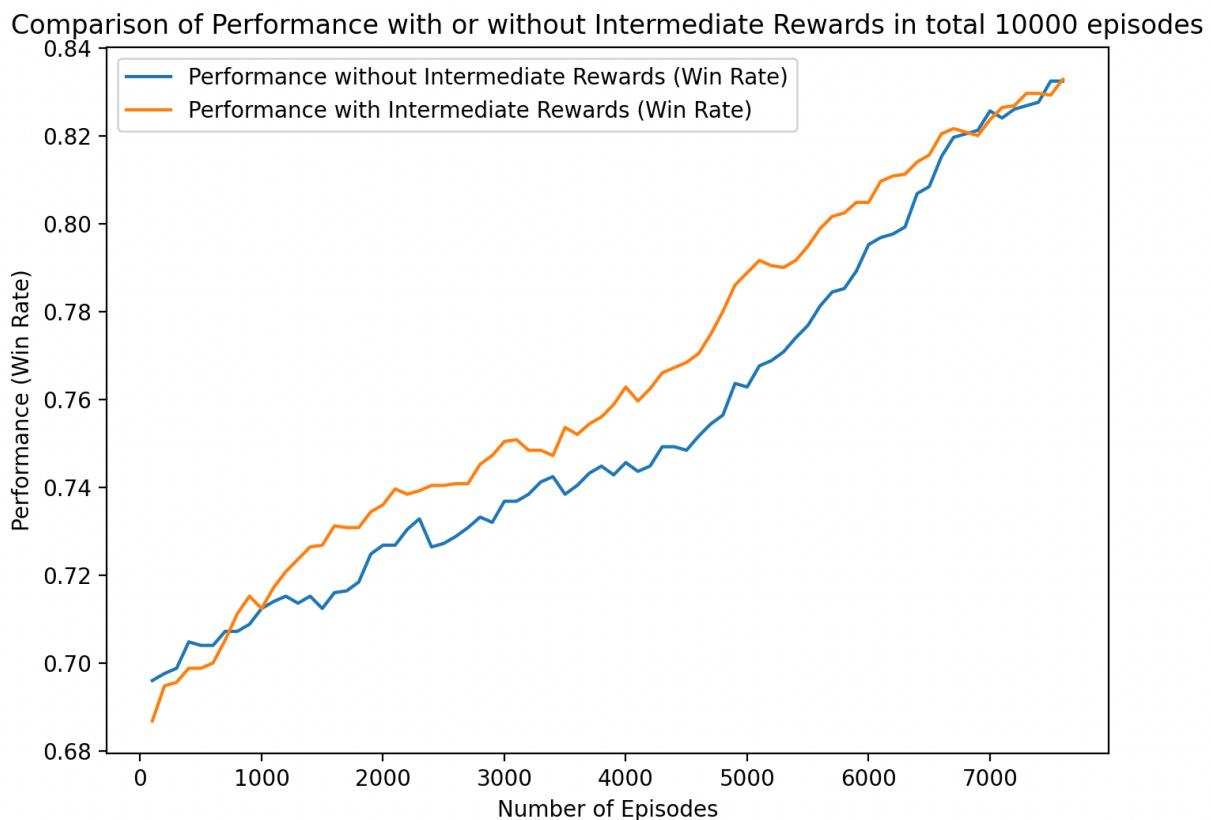


Question 2c

c) Implement a version of your robot that assumes only terminal rewards and show & compare its behaviour with one having intermediate rewards.

I compared the performance (using win rate in every 100 episodes) using only terminal rewards vs using both terminal and intermediate rewards in 10000 episodes. As the graph below shows, the one using both terminal and intermediate rewards seems to make progress faster. However they don't seem to have significant difference in performance once the number of episodes is relatively large.

Note: In order to better show the trend, moving average is used to smooth out fluctuations in data, which is likely the reason why the x axis is slightly shifted.



Question 3a

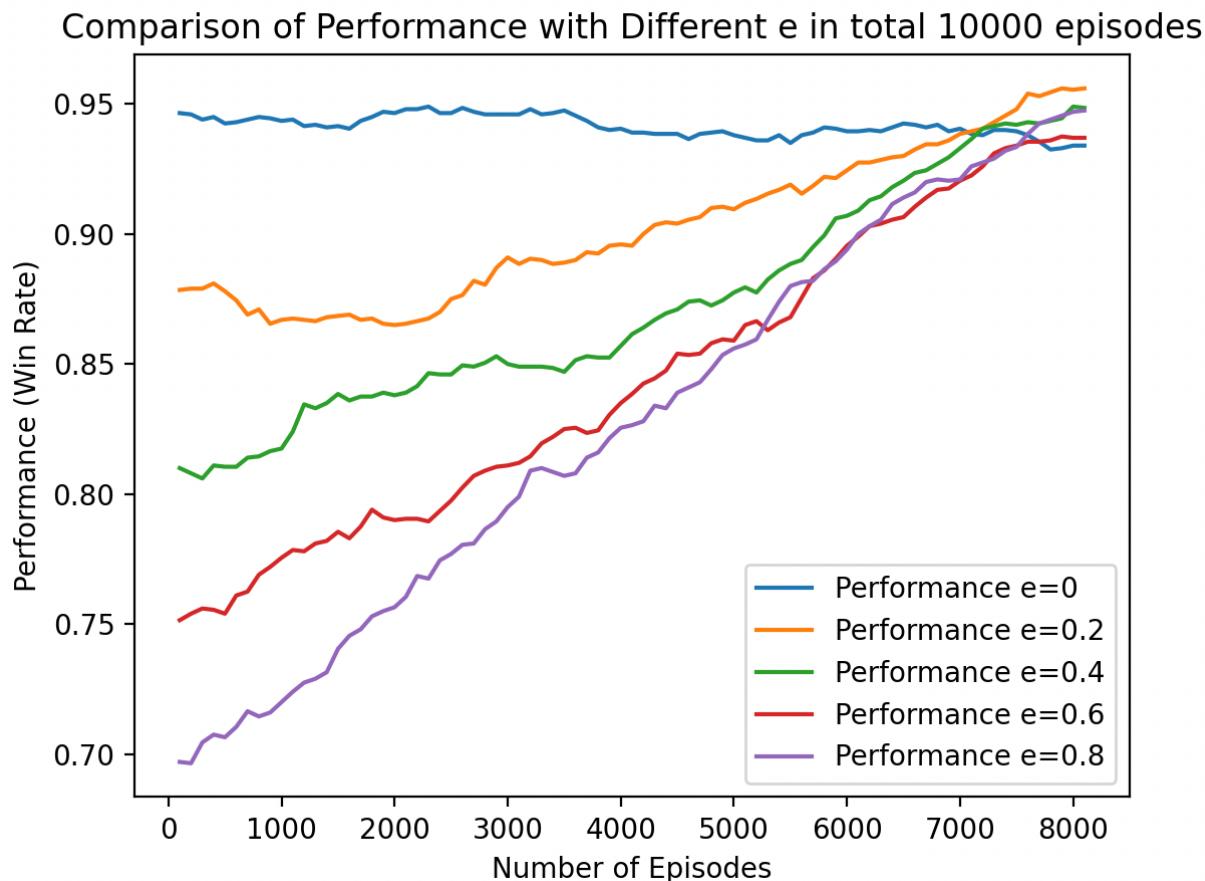
(3) This part is about exploration. While training via RL, the next move is selected randomly with probability e and greedily with probability $1-e$.

a) Compare training performance using different values of e including no exploration at all. Provide graphs of the measured performance of your tank vs e .

I compared the performance (using win rate in every 100 episodes) using different epsilon e in 10000 episodes. In every case, e is gradually decaying to 0 in the first 80% episodes, and staying at 0 in the last 20% episodes in order to be compared fairly.

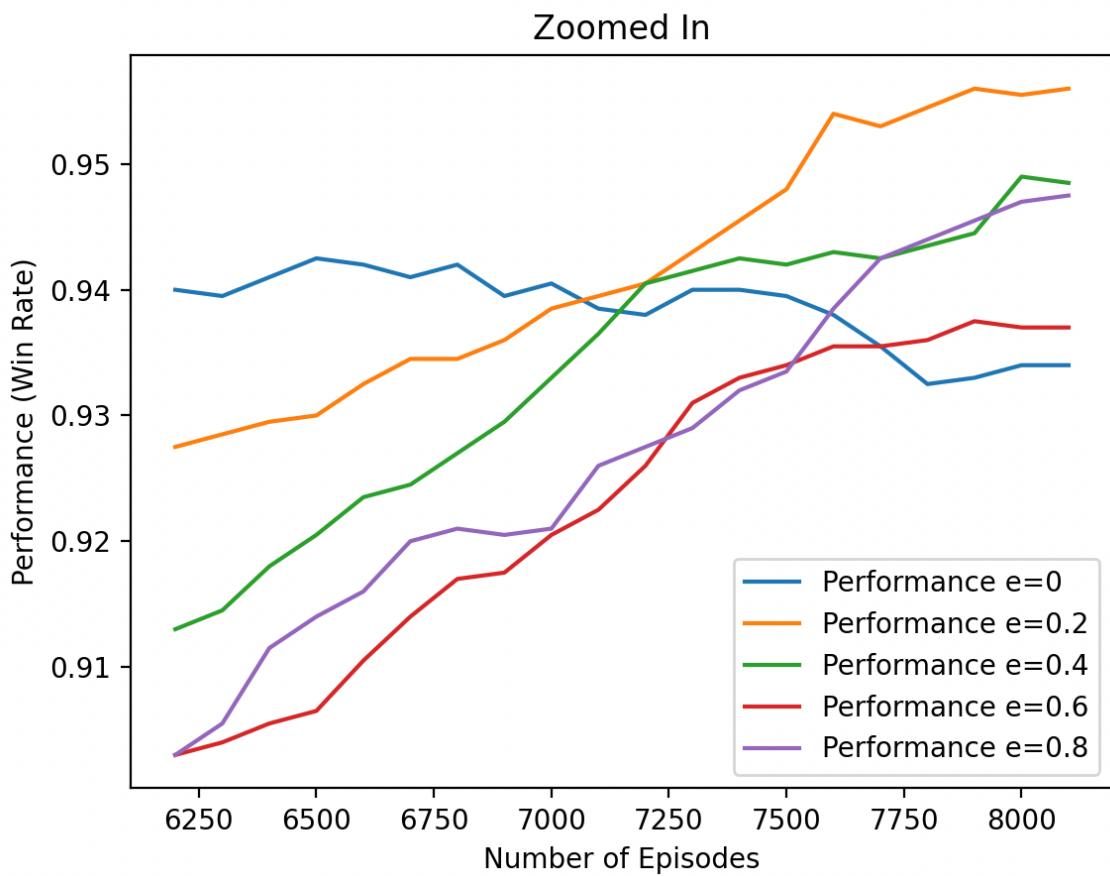
As the graph below shows, during the e decaying stage, learning processes using different e are all improving except $e=0$. During the measuring stage, we can see using $e>0$ would have better performance than $e=0$. (see a zoomed-in graph in the next page)

Note: In order to better show the trend, moving average is used to smooth out fluctuations in data, which is likely the reason why the x slightly shifted.



This graph below is a zoomed-in portion of the graph above, to provide a close look at the measuring stage in the last 20% episodes. In which we can clearly see that using $\epsilon > 0$ would have better performance than $\epsilon = 0$. In this specific case, $\epsilon = 0.2$ has the best performance, followed with $\epsilon = 0.4$, $\epsilon = 0.8$, $\epsilon = 0.6$, and $\epsilon = 0$ has the worst.

Note: In order to better show the trend, moving average is used to smooth out fluctuations in data, which is likely the reason why the x slightly shifted.



Appendix for source code

MyLUTRobot.java

```
1 package Robot.My502Robot;
2
3 import LUT.LUT;
4 import Robot.Action;
5 import Robot.State;
6 import robocode.*;
7
8 import java.awt.*;
9 import java.io.IOException;
10 import java.io.PrintStream;
11 import java.util.Arrays;
12 import java.util.Date;
13
14 import static robocode.util.Utils.normalRelativeAngleDegrees;
15
16
17 public class MyLUTRobot extends AdvancedRobot {
18     private enumOperationalMode operationalMode = enumOperationalMode.scan;
19 //     private String weightsFileName = getClass().getSimpleName() + "-weights.txt";
20     private String logFileName = getClass().getSimpleName() + "-" + "qValues" + new Date().toString() + ".dat";
21
22     private String logFileNameWinRate = getClass().getSimpleName() + "-" + "winRate" + new Date().toString() + ".dat";
23
24     private String logFileNameEpsilonList = getClass().getSimpleName() + "-" + "epsilonList" + new Date().toString() + ".dat";
25     static private LUT lut = new LUT();
26 //     private String[] outputLog;
27     private double curR;
28     private double goodTerminalReward = 1;
29     private double badTerminalReward = -1;
30     private double totalR = 0;
31
32
33
34
35     static double alpha = 0.2;
36
37     static double gamma = 1;
38     static int totalNumRounds = 0;
39     static int numRoundTo100 = 0;
40     static int numWins = 0;
41 // for question2(a), used 15000, other times used 10000
42     static int desiredTotalRounds = 300;
43     static double[] winRatePer100 = new double[desiredTotalRounds/100];
44
45 // for question3(a), try e = 0, 0.2, 0.4, 0.6 and 0.8(default)
46     static double epsilon = 0;
47     static double epsilon_init = 0;
48     static double[] epsilonList = new double[winRatePer100.length];
49
50
51     static int numOfRoundsToDecayE = (int) (desiredTotalRounds * 0.8);
52     static double decayEStepSize = epsilon_init/numOfRoundsToDecayE;
53
54     public enum enumOperationalMode {scan, performAction};
55     private double oE;
56     private double oD;
57     private double oV;
58     private double oB;
59     private double eH;
60
61     private State curS, preS;
62     private Action curA, preA;
63
64     public void run() {
65         initialize();
66         setColor();
67
68         while (true) {
69             switch (operationalMode) {
70                 case scan: {
71                     turnRadarLeft(360);
72                     curR = 0; // reset curR to 0 when scan again
73                     break;
74                 }
75                 case performAction: {
76                     if (Math.random() <= epsilon)
77                         curA = selectRandomAction();
78                     else
79                         curA = bestAction(curS);
80
81                     switch (curA) {
82                         case ATTACK: {
83                             turnGunRight(normalRelativeAngleDegrees(getHeading() - getGunHeading() + oB));
84                             fire(2);
85                             execute();
86                             break;
87                         }
88                     }
89                 }
90             }
91         }
92     }
93
94     private void initialize() {
95         operationalMode = enumOperationalMode.scan;
96         curR = 0;
97         curS = null;
98         curA = null;
99
100        oE = 0;
101        oD = 0;
102        oV = 0;
103        oB = 0;
104        eH = 0;
105    }
106
107    private void setColor() {
108        setBodyColor(100, 100, 100);
109        setGunColor(100, 100, 100);
110        setTurretColor(100, 100, 100);
111    }
112
113    private void turnRadarLeft(int degrees) {
114        turnRadarLeft(degrees);
115    }
116
117    private void turnGunRight(double degrees) {
118        turnGunRight(degrees);
119    }
120
121    private void fire(int count) {
122        fire(count);
123    }
124
125    private void execute() {
126        execute();
127    }
128
129    private void turnGunLeft(int degrees) {
130        turnGunLeft(degrees);
131    }
132
133    private void turnTurretLeft(int degrees) {
134        turnTurretLeft(degrees);
135    }
136
137    private void turnTurretRight(int degrees) {
138        turnTurretRight(degrees);
139    }
140
141    private void turnRadarRight(int degrees) {
142        turnRadarRight(degrees);
143    }
144
145    private void turnGunRight(int degrees) {
146        turnGunRight(degrees);
147    }
148
149    private void turnGunLeft(int degrees) {
150        turnGunLeft(degrees);
151    }
152
153    private void fire() {
154        fire();
155    }
156
157    private void execute() {
158        execute();
159    }
160
161    private void turnRadarLeft() {
162        turnRadarLeft();
163    }
164
165    private void turnTurretLeft() {
166        turnTurretLeft();
167    }
168
169    private void turnTurretRight() {
170        turnTurretRight();
171    }
172
173    private void turnRadarRight() {
174        turnRadarRight();
175    }
176
177    private void turnGunRight() {
178        turnGunRight();
179    }
180
181    private void turnGunLeft() {
182        turnGunLeft();
183    }
184
185    private void fire(int count) {
186        fire(count);
187    }
188
189    private void execute() {
190        execute();
191    }
192
193    private void turnGunLeft(int degrees) {
194        turnGunLeft(degrees);
195    }
196
197    private void turnTurretLeft(int degrees) {
198        turnTurretLeft(degrees);
199    }
200
201    private void turnTurretRight(int degrees) {
202        turnTurretRight(degrees);
203    }
204
205    private void turnRadarRight(int degrees) {
206        turnRadarRight(degrees);
207    }
208
209    private void turnGunRight(int degrees) {
210        turnGunRight(degrees);
211    }
212
213    private void turnGunLeft(int degrees) {
214        turnGunLeft(degrees);
215    }
216
217    private void fire() {
218        fire();
219    }
220
221    private void execute() {
222        execute();
223    }
224
225    private void turnRadarLeft() {
226        turnRadarLeft();
227    }
228
229    private void turnTurretLeft() {
230        turnTurretLeft();
231    }
232
233    private void turnTurretRight() {
234        turnTurretRight();
235    }
236
237    private void turnRadarRight() {
238        turnRadarRight();
239    }
240
241    private void turnGunRight() {
242        turnGunRight();
243    }
244
245    private void turnGunLeft() {
246        turnGunLeft();
247    }
248
249    private void fire(int count) {
250        fire(count);
251    }
252
253    private void execute() {
254        execute();
255    }
256
257    private void turnRadarLeft(int degrees) {
258        turnRadarLeft(degrees);
259    }
260
261    private void turnTurretLeft(int degrees) {
262        turnTurretLeft(degrees);
263    }
264
265    private void turnTurretRight(int degrees) {
266        turnTurretRight(degrees);
267    }
268
269    private void turnRadarRight(int degrees) {
270        turnRadarRight(degrees);
271    }
272
273    private void turnGunRight(int degrees) {
274        turnGunRight(degrees);
275    }
276
277    private void turnGunLeft(int degrees) {
278        turnGunLeft(degrees);
279    }
280
281    private void fire() {
282        fire();
283    }
284
285    private void execute() {
286        execute();
287    }
288
289    private void turnRadarLeft() {
290        turnRadarLeft();
291    }
292
293    private void turnTurretLeft() {
294        turnTurretLeft();
295    }
296
297    private void turnTurretRight() {
298        turnTurretRight();
299    }
300
301    private void turnRadarRight() {
302        turnRadarRight();
303    }
304
305    private void turnGunRight() {
306        turnGunRight();
307    }
308
309    private void turnGunLeft() {
310        turnGunLeft();
311    }
312
313    private void fire(int count) {
314        fire(count);
315    }
316
317    private void execute() {
318        execute();
319    }
320
321    private void turnRadarLeft(int degrees) {
322        turnRadarLeft(degrees);
323    }
324
325    private void turnTurretLeft(int degrees) {
326        turnTurretLeft(degrees);
327    }
328
329    private void turnTurretRight(int degrees) {
330        turnTurretRight(degrees);
331    }
332
333    private void turnRadarRight(int degrees) {
334        turnRadarRight(degrees);
335    }
336
337    private void turnGunRight(int degrees) {
338        turnGunRight(degrees);
339    }
340
341    private void turnGunLeft(int degrees) {
342        turnGunLeft(degrees);
343    }
344
345    private void fire() {
346        fire();
347    }
348
349    private void execute() {
350        execute();
351    }
352
353    private void turnRadarLeft() {
354        turnRadarLeft();
355    }
356
357    private void turnTurretLeft() {
358        turnTurretLeft();
359    }
360
361    private void turnTurretRight() {
362        turnTurretRight();
363    }
364
365    private void turnRadarRight() {
366        turnRadarRight();
367    }
368
369    private void turnGunRight() {
370        turnGunRight();
371    }
372
373    private void turnGunLeft() {
374        turnGunLeft();
375    }
376
377    private void fire(int count) {
378        fire(count);
379    }
380
381    private void execute() {
382        execute();
383    }
384
385    private void turnRadarLeft(int degrees) {
386        turnRadarLeft(degrees);
387    }
388
389    private void turnTurretLeft(int degrees) {
390        turnTurretLeft(degrees);
391    }
392
393    private void turnTurretRight(int degrees) {
394        turnTurretRight(degrees);
395    }
396
397    private void turnRadarRight(int degrees) {
398        turnRadarRight(degrees);
399    }
400
401    private void turnGunRight(int degrees) {
402        turnGunRight(degrees);
403    }
404
405    private void turnGunLeft(int degrees) {
406        turnGunLeft(degrees);
407    }
408
409    private void fire() {
410        fire();
411    }
412
413    private void execute() {
414        execute();
415    }
416
417    private void turnRadarLeft() {
418        turnRadarLeft();
419    }
420
421    private void turnTurretLeft() {
422        turnTurretLeft();
423    }
424
425    private void turnTurretRight() {
426        turnTurretRight();
427    }
428
429    private void turnRadarRight() {
430        turnRadarRight();
431    }
432
433    private void turnGunRight() {
434        turnGunRight();
435    }
436
437    private void turnGunLeft() {
438        turnGunLeft();
439    }
440
441    private void fire(int count) {
442        fire(count);
443    }
444
445    private void execute() {
446        execute();
447    }
448
449    private void turnRadarLeft(int degrees) {
450        turnRadarLeft(degrees);
451    }
452
453    private void turnTurretLeft(int degrees) {
454        turnTurretLeft(degrees);
455    }
456
457    private void turnTurretRight(int degrees) {
458        turnTurretRight(degrees);
459    }
460
461    private void turnRadarRight(int degrees) {
462        turnRadarRight(degrees);
463    }
464
465    private void turnGunRight(int degrees) {
466        turnGunRight(degrees);
467    }
468
469    private void turnGunLeft(int degrees) {
470        turnGunLeft(degrees);
471    }
472
473    private void fire() {
474        fire();
475    }
476
477    private void execute() {
478        execute();
479    }
480
481    private void turnRadarLeft() {
482        turnRadarLeft();
483    }
484
485    private void turnTurretLeft() {
486        turnTurretLeft();
487    }
488
489    private void turnTurretRight() {
490        turnTurretRight();
491    }
492
493    private void turnRadarRight() {
494        turnRadarRight();
495    }
496
497    private void turnGunRight() {
498        turnGunRight();
499    }
500
501    private void turnGunLeft() {
502        turnGunLeft();
503    }
504
505    private void fire(int count) {
506        fire(count);
507    }
508
509    private void execute() {
510        execute();
511    }
512
513    private void turnRadarLeft(int degrees) {
514        turnRadarLeft(degrees);
515    }
516
517    private void turnTurretLeft(int degrees) {
518        turnTurretLeft(degrees);
519    }
520
521    private void turnTurretRight(int degrees) {
522        turnTurretRight(degrees);
523    }
524
525    private void turnRadarRight(int degrees) {
526        turnRadarRight(degrees);
527    }
528
529    private void turnGunRight(int degrees) {
530        turnGunRight(degrees);
531    }
532
533    private void turnGunLeft(int degrees) {
534        turnGunLeft(degrees);
535    }
536
537    private void fire() {
538        fire();
539    }
540
541    private void execute() {
542        execute();
543    }
544
545    private void turnRadarLeft() {
546        turnRadarLeft();
547    }
548
549    private void turnTurretLeft() {
550        turnTurretLeft();
551    }
552
553    private void turnTurretRight() {
554        turnTurretRight();
555    }
556
557    private void turnRadarRight() {
558        turnRadarRight();
559    }
560
561    private void turnGunRight() {
562        turnGunRight();
563    }
564
565    private void turnGunLeft() {
566        turnGunLeft();
567    }
568
569    private void fire(int count) {
570        fire(count);
571    }
572
573    private void execute() {
574        execute();
575    }
576
577    private void turnRadarLeft(int degrees) {
578        turnRadarLeft(degrees);
579    }
580
581    private void turnTurretLeft(int degrees) {
582        turnTurretLeft(degrees);
583    }
584
585    private void turnTurretRight(int degrees) {
586        turnTurretRight(degrees);
587    }
588
589    private void turnRadarRight(int degrees) {
590        turnRadarRight(degrees);
591    }
592
593    private void turnGunRight(int degrees) {
594        turnGunRight(degrees);
595    }
596
597    private void turnGunLeft(int degrees) {
598        turnGunLeft(degrees);
599    }
600
601    private void fire() {
602        fire();
603    }
604
605    private void execute() {
606        execute();
607    }
608
609    private void turnRadarLeft() {
610        turnRadarLeft();
611    }
612
613    private void turnTurretLeft() {
614        turnTurretLeft();
615    }
616
617    private void turnTurretRight() {
618        turnTurretRight();
619    }
620
621    private void turnRadarRight() {
622        turnRadarRight();
623    }
624
625    private void turnGunRight() {
626        turnGunRight();
627    }
628
629    private void turnGunLeft() {
630        turnGunLeft();
631    }
632
633    private void fire(int count) {
634        fire(count);
635    }
636
637    private void execute() {
638        execute();
639    }
640
641    private void turnRadarLeft(int degrees) {
642        turnRadarLeft(degrees);
643    }
644
645    private void turnTurretLeft(int degrees) {
646        turnTurretLeft(degrees);
647    }
648
649    private void turnTurretRight(int degrees) {
650        turnTurretRight(degrees);
651    }
652
653    private void turnRadarRight(int degrees) {
654        turnRadarRight(degrees);
655    }
656
657    private void turnGunRight(int degrees) {
658        turnGunRight(degrees);
659    }
660
661    private void turnGunLeft(int degrees) {
662        turnGunLeft(degrees);
663    }
664
665    private void fire() {
666        fire();
667    }
668
669    private void execute() {
670        execute();
671    }
672
673    private void turnRadarLeft() {
674        turnRadarLeft();
675    }
676
677    private void turnTurretLeft() {
678        turnTurretLeft();
679    }
680
681    private void turnTurretRight() {
682        turnTurretRight();
683    }
684
685    private void turnRadarRight() {
686        turnRadarRight();
687    }
688
689    private void turnGunRight() {
690        turnGunRight();
691    }
692
693    private void turnGunLeft() {
694        turnGunLeft();
695    }
696
697    private void fire(int count) {
698        fire(count);
699    }
700
701    private void execute() {
702        execute();
703    }
704
705    private void turnRadarLeft(int degrees) {
706        turnRadarLeft(degrees);
707    }
708
709    private void turnTurretLeft(int degrees) {
710        turnTurretLeft(degrees);
711    }
712
713    private void turnTurretRight(int degrees) {
714        turnTurretRight(degrees);
715    }
716
717    private void turnRadarRight(int degrees) {
718        turnRadarRight(degrees);
719    }
720
721    private void turnGunRight(int degrees) {
722        turnGunRight(degrees);
723    }
724
725    private void turnGunLeft(int degrees) {
726        turnGunLeft(degrees);
727    }
728
729    private void fire() {
730        fire();
731    }
732
733    private void execute() {
734        execute();
735    }
736
737    private void turnRadarLeft() {
738        turnRadarLeft();
739    }
740
741    private void turnTurretLeft() {
742        turnTurretLeft();
743    }
744
745    private void turnTurretRight() {
746        turnTurretRight();
747    }
748
749    private void turnRadarRight() {
750        turnRadarRight();
751    }
752
753    private void turnGunRight() {
754        turnGunRight();
755    }
756
757    private void turnGunLeft() {
758        turnGunLeft();
759    }
760
761    private void fire(int count) {
762        fire(count);
763    }
764
765    private void execute() {
766        execute();
767    }
768
769    private void turnRadarLeft(int degrees) {
770        turnRadarLeft(degrees);
771    }
772
773    private void turnTurretLeft(int degrees) {
774        turnTurretLeft(degrees);
775    }
776
777    private void turnTurretRight(int degrees) {
778        turnTurretRight(degrees);
779    }
780
781    private void turnRadarRight(int degrees) {
782        turnRadarRight(degrees);
783    }
784
785    private void turnGunRight(int degrees) {
786        turnGunRight(degrees);
787    }
788
789    private void turnGunLeft(int degrees) {
790        turnGunLeft(degrees);
791    }
792
793    private void fire() {
794        fire();
795    }
796
797    private void execute() {
798        execute();
799    }
800
801    private void turnRadarLeft() {
802        turnRadarLeft();
803    }
804
805    private void turnTurretLeft() {
806        turnTurretLeft();
807    }
808
809    private void turnTurretRight() {
810        turnTurretRight();
811    }
812
813    private void turnRadarRight() {
814        turnRadarRight();
815    }
816
817    private void turnGunRight() {
818        turnGunRight();
819    }
820
821    private void turnGunLeft() {
822        turnGunLeft();
823    }
824
825    private void fire(int count) {
826        fire(count);
827    }
828
829    private void execute() {
830        execute();
831    }
832
833    private void turnRadarLeft(int degrees) {
834        turnRadarLeft(degrees);
835    }
836
837    private void turnTurretLeft(int degrees) {
838        turnTurretLeft(degrees);
839    }
840
841    private void turnTurretRight(int degrees) {
842        turnTurretRight(degrees);
843    }
844
845    private void turnRadarRight(int degrees) {
846        turnRadarRight(degrees);
847    }
848
849    private void turnGunRight(int degrees) {
850        turnGunRight(degrees);
851    }
852
853    private void turnGunLeft(int degrees) {
854        turnGunLeft(degrees);
855    }
856
857    private void fire() {
858        fire();
859    }
860
861    private void execute() {
862        execute();
863    }
864
865    private void turnRadarLeft() {
866        turnRadarLeft();
867    }
868
869    private void turnTurretLeft() {
870        turnTurretLeft();
871    }
872
873    private void turnTurretRight() {
874        turnTurretRight();
875    }
876
877    private void turnRadarRight() {
878        turnRadarRight();
879    }
880
881    private void turnGunRight() {
882        turnGunRight();
883    }
884
885    private void turnGunLeft() {
886        turnGunLeft();
887    }
888
889    private void fire(int count) {
890        fire(count);
891    }
892
893    private void execute() {
894        execute();
895    }
896
897    private void turnRadarLeft(int degrees) {
898        turnRadarLeft(degrees);
899    }
900
901    private void turnTurretLeft(int degrees) {
902        turnTurretLeft(degrees);
903    }
904
905    private void turnTurretRight(int degrees) {
906        turnTurretRight(degrees);
907    }
908
909    private void turnRadarRight(int degrees) {
910        turnRadarRight(degrees);
911    }
912
913    private void turnGunRight(int degrees) {
914        turnGunRight(degrees);
915    }
916
917    private void turnGunLeft(int degrees) {
918        turnGunLeft(degrees);
919    }
920
921    private void fire() {
922        fire();
923    }
924
925    private void execute() {
926        execute();
927    }
928
929    private void turnRadarLeft() {
930        turnRadarLeft();
931    }
932
933    private void turnTurretLeft() {
934        turnTurretLeft();
935    }
936
937    private void turnTurretRight() {
938        turnTurretRight();
939    }
940
941    private void turnRadarRight() {
942        turnRadarRight();
943    }
944
945    private void turnGunRight() {
946        turnGunRight();
947    }
948
949    private void turnGunLeft() {
950        turnGunLeft();
951    }
952
953    private void fire(int count) {
954        fire(count);
955    }
956
957    private void execute() {
958        execute();
959    }
960
961    private void turnRadarLeft(int degrees) {
962        turnRadarLeft(degrees);
963    }
964
965    private void turnTurretLeft(int degrees) {
966        turnTurretLeft(degrees);
967    }
968
969    private void turnTurretRight(int degrees) {
970        turnTurretRight(degrees);
971    }
972
973    private void turnRadarRight(int degrees) {
974        turnRadarRight(degrees);
975    }
976
977    private void turnGunRight(int degrees) {
978        turnGunRight(degrees);
979    }
980
981    private void turnGunLeft(int degrees) {
982        turnGunLeft(degrees);
983    }
984
985    private void fire() {
986        fire();
987    }
988
989    private void execute() {
990        execute();
991    }
992
993    private void turnRadarLeft() {
994        turnRadarLeft();
995    }
996
997    private void turnTurretLeft() {
998        turnTurretLeft();
999    }
1000
1001    private void turnTurretRight() {
1002        turnTurretRight();
1003    }
1004
1005    private void turnRadarRight() {
1006        turnRadarRight();
1007    }
1008
1009    private void turnGunRight() {
1010        turnGunRight();
1011    }
1012
1013    private void turnGunLeft() {
1014        turnGunLeft();
1015    }
1016
1017    private void fire(int count) {
1018        fire(count);
1019    }
1020
1021    private void execute() {
1022        execute();
1023    }
1024
1025    private void turnRadarLeft(int degrees) {
1026        turnRadarLeft(degrees);
1027    }
1028
1029    private void turnTurretLeft(int degrees) {
1030        turnTurretLeft(degrees);
1031    }
1032
1033    private void turnTurretRight(int degrees) {
1034        turnTurretRight(degrees);
1035    }
1036
1037    private void turnRadarRight(int degrees) {
1038        turnRadarRight(degrees);
1039    }
1040
1041    private void turnGunRight(int degrees) {
1042        turnGunRight(degrees);
1043    }
1044
1045    private void turnGunLeft(int degrees) {
1046        turnGunLeft(degrees);
1047    }
1048
1049    private void fire() {
1050        fire();
1051    }
1052
1053    private void execute() {
1054        execute();
1055    }
1056
1057    private void turnRadarLeft() {
1058        turnRadarLeft();
1059    }
1060
1061    private void turnTurretLeft() {
1062        turnTurretLeft();
1063    }
1064
1065    private void turnTurretRight() {
1066        turnTurretRight();
1067    }
1068
1069    private void turnRadarRight() {
1070        turnRadarRight();
1071    }
1072
1073    private void turnGunRight() {
1074        turnGunRight();
1075    }
1076
1077    private void turnGunLeft() {
1078        turnGunLeft();
1079    }
1080
1081    private void fire(int count) {
1082        fire(count);
1083    }
1084
1085    private void execute() {
1086        execute();
1087    }
1088
1089    private void turnRadarLeft(int degrees) {
1090        turnRadarLeft(degrees);
1091    }
1092
1093    private void turnTurretLeft(int degrees) {
1094        turnTurretLeft(degrees);
1095    }
1096
1097    private void turnTurretRight(int degrees) {
1098        turnTurretRight(degrees);
1099    }
1100
1101    private void turnRadarRight(int degrees) {
1102        turnRadarRight(degrees);
1103    }
1104
1105    private void turnGunRight(int degrees) {
1106        turnGunRight(degrees);
1107    }
1108
1109    private void turnGunLeft(int degrees) {
1110        turnGunLeft(degrees);
1111    }
1112
1113    private void fire() {
1114        fire();
1115    }
1116
1117    private void execute() {
1118        execute();
1119    }
1120
1121    private void turnRadarLeft() {
1122        turnRadarLeft();
1123    }
1124
1125    private void turnTurretLeft() {
1126        turnTurretLeft();
1127    }
1128
1129    private void turnTurretRight() {
1130        turnTurretRight();
1131    }
1132
1133    private void turnRadarRight() {
1134        turnRadarRight();
1135    }
1136
1137    private void turnGunRight() {
1138        turnGunRight();
1139    }
1140
1141    private void turnGunLeft() {
1142        turnGunLeft();
1143    }
1144
1145    private void fire(int count) {
1146        fire(count);
1147    }
1148
1149    private void execute() {
1150        execute();
1151    }
1152
1153    private void turnRadarLeft(int degrees) {
1154        turnRadarLeft(degrees);
1155    }
1156
1157    private void turnTurretLeft(int degrees) {
1158        turnTurretLeft(degrees);
1159    }
1160
1161    private void turnTurretRight(int degrees) {
1162        turnTurretRight(degrees);
1163    }
1164
1165    private void turnRadarRight(int degrees) {
1166        turnRadarRight(degrees);
1167    }
1168
1169    private void turnGunRight(int degrees) {
1170        turnGunRight(degrees);
1171    }
1172
1173    private void turnGunLeft(int degrees) {
1174        turnGunLeft(degrees);
1175    }
1176
1177    private void fire() {
1178        fire();
1179    }
1180
1181    private void execute() {
1182        execute();
1183    }
1184
1185    private void turnRadarLeft() {
1186        turnRadarLeft();
1187    }
1188
1189    private void turnTurretLeft() {
1190        turnTurretLeft();
1191    }
1192
1193    private void turnTurretRight() {
1194        turnTurretRight();
1195    }
1196
1197    private void turnRadarRight() {
1198        turnRadarRight();
1199    }
1200
1201    private void turnGunRight() {
1202        turnGunRight();
1203    }
1204
1205    private void turnGunLeft() {
1206        turnGunLeft();
1207    }
1208
1209    private void fire(int count) {
1210        fire(count);
1211    }
1212
1213    private void execute() {
1214        execute();
1215    }
1216
1217    private void turnRadarLeft(int degrees) {
1218        turnRadarLeft(degrees);
1219    }
1220
1221    private void turnTurretLeft(int degrees) {
1222        turnTurretLeft(degrees);
1223    }
1224
1225    private void turnTurretRight(int degrees) {
1226        turnTurretRight(degrees);
1227    }
1228
1229    private void turnRadarRight(int degrees) {
1230        turnRadarRight(degrees);
1231    }
1232
1233    private void turnGunRight(int degrees) {
1234        turnGunRight(degrees);
1235    }
1236
1237    private void turnGunLeft(int degrees) {
1238        turnGunLeft(degrees);
1239    }
1240
1241    private void fire() {
1242        fire();
1243    }
1244
1245    private void execute() {
1246        execute();
1247    }
1248
1249    private void turnRadarLeft() {
1250        turnRadarLeft();
1251    }
1252
1253    private void turnTurretLeft() {
1254        turnTurretLeft();
1255    }
1256
1257    private void turnTurretRight() {
1258        turnTurretRight();
1259    }
1260
1261    private void turnRadarRight() {
1262        turnRadarRight();
1263    }
1264
1265    private void turnGunRight() {
1266        turnGunRight();
1267    }
1268
1269    private void turnGunLeft() {
1270        turnGunLeft();
1271    }
1272
1273    private void fire(int count) {
1274        fire(count);
1275    }
1276
1277    private void execute() {
1278        execute();
1279    }
1280
1281    private void turnRadarLeft(int degrees) {
1282        turnRadarLeft(degrees);
1283    }
1284
1285    private void turnTurretLeft(int degrees) {
1286        turnTurretLeft(degrees);
1287    }
1288
1289    private void turnTurretRight(int degrees) {
1290        turnTurretRight(degrees);
1291    }
1292
1293    private void turnRadarRight(int degrees) {
1294        turnRadarRight(degrees);
1295    }
1296
1297    private void turnGunRight(int degrees) {
1298        turnGunRight(degrees);
1299    }
1300
1301    private void turnGunLeft(int degrees) {
1302        turnGunLeft(degrees);
1303    }
1304
1305    private void fire() {
1306        fire();
1307    }
1308
1309    private void execute() {
1310        execute();
1311    }
1312
1313    private void turnRadarLeft() {
1314        turnRadarLeft();
1315    }
1316
1317    private void turnTurretLeft() {
1318        turnTurretLeft();
1319    }
1320
1321    private void turnTurretRight() {
1322        turnTurretRight();
1323    }
1324
1325    private void turnRadarRight() {
1326        turnRadarRight();
1327    }
1328
1329    private void turnGunRight() {
1330        turnGunRight();
1331    }
1332
1333    private void turnGunLeft() {
1334        turnGunLeft();
1335    }
1336
1337    private void fire(int count) {
1338        fire(count);
1339    }
1340
1341    private void execute() {
1342        execute();
1343    }
1344
1345    private void turnRadarLeft(int degrees) {
1346        turnRadarLeft(degrees);
1347    }
1348
1349    private void turnTurretLeft(int degrees) {
1350        turnTurretLeft(degrees);
1351    }
1352
1353    private void turnTurretRight(int degrees) {
1354        turnTurretRight(degrees);
1355    }
1356
1357    private void turnRadarRight(int degrees) {
1358        turnRadarRight(degrees);
1359    }
1360
1361    private void turnGunRight(int degrees) {
1362        turnGunRight(degrees);
1363    }
1364
1365    private void turnGunLeft(int degrees) {
1366        turnGunLeft(degrees);
1367    }
1368
1369    private void fire() {
1370        fire();
1371    }
1372
1373    private void execute() {
1374        execute();
1375    }
1376
1377    private void turnRadarLeft() {
1378        turnRadarLeft();
1379    }
1380
1381    private void turnTurretLeft() {
1382        turnTurretLeft();
1383    }
1384
1385    private void turnTurretRight() {
1386        turnTurretRight();
1387    }
1388
1389    private void turnRadarRight() {
1390        turnRadarRight();
1391    }
1392
1393    private void turnGunRight() {
1394        turnGunRight();
1395    }
1396
1397    private void turnGunLeft() {
1398        turnGunLeft();
1399    }
1400
1401    private void fire(int count) {
1402        fire(count);
1403    }
1404
1405    private void execute() {
1406        execute();
1407    }
1408
1409    private void turnRadarLeft(int degrees) {
1410        turnRadarLeft(degrees);
1411    }
1412
1413    private void turnTurretLeft(int degrees) {
1414        turnTurretLeft(degrees);
1415    }
1416
1417    private void turnTurretRight(int degrees) {
1418        turnTurretRight(degrees);
1419    }
1420
1421    private void turnRadarRight(int degrees) {
1422        turnRadarRight(degrees);
1423    }
1424
1425    private void turnGunRight(int degrees) {
1426        turnGunRight(degrees);
1427    }
1428
1429    private void turnGunLeft(int degrees) {
1430        turnGunLeft(degrees);
1431    }
1432
1433
```

```

88             case RUN_AWAY: {
89                 turnRight(normalRelativeAngleDegrees(90 - (getHeading() - eH)));
90                 ahead(50);
91                 execute();
92                 break;
93             }
94             // deleted this action because it does not seem to be useful
95             case CHASE: {
96                 turnRight(oB);
97                 setVelocityRate(3);
98                 ahead(50);
99                 execute();
100                break;
101            }
102        }
103    }
104
105    // update Q value for (s,a)
106    lut.train(preS.transformToX(preA), computeQ(preS, curS, curR));
107
108    // TODO: for assignment 3
109    // replayMemory.add(new Experience(pres, preA, curR, curS));
110    // replayExperience(replayMemory);
111
112    this.operationalMode = enumOperationalMode.scan;
113}
114}
115}
116}
117
118 private void setColor() {
119     setBodyColor(Color.yellow);
120     setGunColor(Color.black);
121     setRadarColor(Color.red);
122     setBulletColor(Color.white);
123     setScanColor(Color.white);
124 }
125
126 private void initialize() {
127     curS = new State(getEnergy(), 100, getX(), getY(), 100, getVelocity(), 0.2);
128     curA = Action.values()[0];
129
130     preS = curS;
131     preA = curA;
132 }
133
134 private Action bestAction(State curS) {
135     double bestQ = -Double.MAX_VALUE;
136     int bestAindex = 0;
137     double[] X = curS.transformToX();
138     double[] newX = Arrays.copyOf(X, X.length + 1);
139     for (int i = 0; i < Action.values().length; i++) {
140         newX[X.length] = i;
141         double q = lut.outputFor(newX);
142         if (q > bestQ) {
143             // bestQ = q;
144             bestAindex = i;
145         }
146     }
147     Action bestA = Action.values()[bestAindex];
148     return bestA;
149 }
150
151 private double bestActionQ(State curS) {
152     double bestQ = 0;
153     int bestAindex = 0;
154     double[] X = curS.transformToX();
155     double[] newX = Arrays.copyOf(X, X.length + 1);
156     for (int i = 0; i < Action.values().length; i++) {
157         newX[X.length] = i;
158         double q = lut.outputFor(newX);
159         if (q > bestQ) {
160             bestQ = q;
161             bestAindex = i;
162         }
163     }
164     Action bestA = Action.values()[bestAindex];
165     return bestQ;
166 }
167
168
169 private double computeQ(State pres, State curS, double r) {
170     // off-policy, q learning
171     // take action, observe r, s' (find the best a' and update Q(s,a))
172     //  $Q(s,a) = Q(s, a) + \alpha(r + \gamma * \max(Q(s', a')) - Q(s,a))$ 
173     double oldQ = lut.outputFor(pres.transformToX(preA));
174     double maxNextQ = bestActionQ(curS);
175     return oldQ + alpha * (r + gamma * maxNextQ - oldQ);
176
177     // on-policy
178     // double oldQ = lut.outputFor(pres.transformToX(preA));

```

```

179 //      double curQ = lut.outputFor(curS.transformToX(curA));
180 //      return oldQ + alpha * (r + gamma * curQ - oldQ);
181 }
182
183     private Action selectRandomAction() {
184         int numOfChoice = Action.values().length;
185         return Action.values()[(int) (Math.random() * numOfChoice)];
186     }
187
188 //    public void replayExperience(ReplayMemory rm){
189 //        int ms = rm.sizeOf();
190 //        int requestedSs = (ms < MAX)
191 //    }
192
193     public void onScannedRobot(ScannedRobotEvent e) {
194         // update preS, preA; update curS
195         preS = curS;
196         preA = curA;
197         curS = new State(getEnergy(), e.getEnergy(), getX(), getY(), getDistance(), getVelocity(), e.getVelocity());
198         oB = e.getBearing();
199         eH = e.getHeading();
200
201         this.operationalMode = enumOperationalMode.performAction;
202     }
203
204
205     public void onWin(WinEvent e) {
206         System.out.println("I win!!!!!!!!!!!!");
207         numWins++;
208
209         curR = goodTerminalReward;
210         totalR += curR;
211
212         lut.train(preS.transformToX(preA), computeQ(preS, curS, curR));
213         //TODO: can add stat
214     }
215
216     public void onDeath(DeathEvent e) {
217         System.out.println("I lose.");
218         curR = badTerminalReward;
219         totalR += curR;
220
221         lut.train(preS.transformToX(preA), computeQ(preS, curS, curR));
222         //TODO: can add stat
223     }
224     ////////////////////intermediate rewards start/////////////////
225     public void onBulletHit(BulletHitEvent e) {
226         curR = +0.4;
227         totalR += curR;
228
229         //lut.train(preS.transformToX(), computeQ(preS, curS, curR));
230     }
231
232     public void onBulletMissed(BulletMissedEvent e) {
233         curR = -0.01;
234         totalR += curR;
235     }
236
237     public void onHitByBullet(HitByBulletEvent event) {
238         curR = -0.2;
239         totalR += curR;
240     }
241
242     public void onHitWall(HitWallEvent event) {
243         curR = -0.01;
244         totalR += curR;
245     }
246     ////////////////////intermediate rewards end/////////////////
247     public void onRoundEnded(RoundEndedEvent event) {
248
249         if(totalNumRounds < numOfRoundsToDecayE){
250             if(epsilon > 0 & epsilon > decayEStepSize){
251                 epsilon -= decayEStepSize; // so e decaying to 0 in the first 80% round
252             }
253         }else{
254             epsilon=0;
255         }
256
257         totalNumRounds++;
258         if(totalNumRounds % 100 == 0){
259             int index = totalNumRounds / 100 - 1;
260             winRatePer100[index] = numWins;
261             epsilonList[index] = epsilon;
262
263             out.println("The round has ended and the winRatePer100[] updated");
264             out.println("totalNumRounds"+ totalNumRounds);
265             out.println("winRatePer100" + winRatePer100[index]);
266             out.println("numWins" + numWins);
267             numWins = 0; // reset
268             out.println("numWins set to 0 again");
269         }

```

```

270     }
271     System.out.println("round ended");
272 }
273
274
275 public void onBattleEnded(BattleEndedEvent e)
276 {
277     finalWriteQ();
278     finalWriteWins();
279     finalWriteEpsilonList();
280 }
281
282 private void finalWriteEpsilonList() {
283     PrintStream w = null;
284     try {
285         w = new PrintStream(new RobocodeFileOutputStream(getDataFile(logFileNameEpsilonList)));
286
287         for(double e: epsilonList){
288             w.println(e);
289         }
290
291         if (w.checkError()) {
292             out.println("I could not write the finalWriteEpsilonList!");
293         }
294     } catch (IOException e) {
295         out.println("IOException trying to write: ");
296         e.printStackTrace(out);
297     } finally {
298         if (w != null) {
299             w.close();
300         }
301     }
302 }
303
304 private void finalWriteWins() {
305     PrintStream w = null;
306     try {
307         w = new PrintStream(new RobocodeFileOutputStream(getDataFile(logFileNameWinRate)));
308
309         for(double winR: winRatePer100){
310             w.println(winR);
311         }
312
313         if (w.checkError()) {
314             out.println("I could not write the winRatePer100!");
315         }
316     } catch (IOException e) {
317         out.println("IOException trying to write: ");
318         e.printStackTrace(out);
319     } finally {
320         if (w != null) {
321             w.close();
322         }
323     }
324 }
325
326
327
328 private void finalWriteQ(){
329     PrintStream w = null;
330     try {
331         w = new PrintStream(new RobocodeFileOutputStream(getDataFile(logFileName)));
332
333         double[] qs = lut.getQValues();
334         for(double q: qs){
335             w.println(q);
336         }
337         // PrintStreams don't throw IOExceptions during prints, they simply set a flag.... so check it here.
338         if (w.checkError()) {
339             out.println("I could not finalWriteQ!");
340         }
341     } catch (IOException e) {
342         out.println("IOException trying to write: ");
343         e.printStackTrace(out);
344     } finally {
345         if (w != null) {
346             w.close();
347         }
348     }
349 }
350
351 }
352 }
```

LUT.java

```
1 package LUT;
2
3 import Interface.LUTInterface;
4 import Robot.Action;
5 import Robot.State;
6
7 import java.io.File;
8 import java.io.IOException;
9
10 public class LUT implements LUTInterface {
11     private double[] qValues;
12     private int numOfStates;
13     private int numOfActions;
14
15     /**
16      * Constructor. (You will need to define one in your implementation)
17      * @param argNumInputs The number of inputs in your input vector
18      * @param argVariableFloor An array specifying the lowest value of each variable in the input vector.
19      * @param argVariableCeiling An array specifying the highest value of each of the variables in the input vector.
20      * The order must match the order as referred to in argVariableFloor. *
21
22     public LUT( int argNumInputs, int [] argVariableFloor, int [] argVariableCeiling ) {
23         numOfStates = State.possibleStates;
24         numOfActions = Action.values().length;
25         int totalStates = 0;
26
27         for(int i = 0; i < argNumInputs-1; i++){
28             totalStates += (argVariableCeiling[i] - argVariableFloor[i]);
29         }
30         this.numOfStates = totalStates;
31         this.numOfActions = argVariableCeiling[argNumInputs] - argVariableFloor[argNumInputs];
32         initialiseLUT();
33     }
34
35     /**
36      * Initialise the look up table to all zeros.
37      */
38     @Override
39     public void initialiseLUT() {
40         this.qValues = new double[numOfStates * numOfActions];
41     }
42
43
44     /**
45      * A helper method that translates a vector being used to index the
46      * look up table into an ordinal that can then be used to access
47      * the associated look up table element.
48      * @param X The state action vector used to index the LUT.LUT
49      * @return The index where this vector maps to
50      */
51     @Override
52     public int indexFor(double[] X) {
53         // X = state + action
54         // form the stateVec from copying first n-1 element from X
55         // so that it can be used as a parameter to form a State object
56         // so that we can use getIndex function in State.class to index
57         double[] stateVec = new double[X.length-1];
58         for (int i = 0; i < stateVec.length; i++) {
59             stateVec[i] = X[i];
60         }
61         State state = new State(stateVec[0], stateVec[1], stateVec[2], stateVec[3], stateVec[4], stateVec[5], stateVec[6]);
62         int index = state.getIndex((int) X[X.length]);
63
64         // we can form the action from X, but it is useless
65         // Action action = Action.values()[(int)X[X.length]];
66
67         return index;
68     }
69
70     @Override
71     public int indexFor(double[] X) {
72         // X = state(size=5) + action (size=1) -> X length = 6
73         //
74         State state = new State((int)X[0], (int)X[1], (int)X[2], (int)X[3], (int)X[4]);
75         int actionIndex = (int) X[5];
76         int index = state.getIndex(actionIndex);
77
78         return index;
79     }
80
81     @Override
82     public double outputFor(double[] X) {
83         double output = 0.0;
84         try {
85             output = qValues[indexFor(X)];
86             //return output;
87         }
```

```
88     } catch (ArrayIndexOutOfBoundsException e) {
89         System.out.println("Error: " + e.getMessage());
90         for(double x: X){
91             System.out.println(x);
92         }
93     }
94     return output;
95     // return qValues[indexFor(X)];
96 }
97 public double[] getQValues(){
98     return qValues;
99 }
100
101 @Override
102 public double train(double[] X, double argValue) {
103     qValues[indexFor(X)] = argValue;
104     return 0;
105 }
106
107 @Override
108 public void save(File argFile) {
109 }
110
111
112 @Override
113 public void load(String argFileName) throws IOException {
114 }
115
116
117 }
118 }
```

State.java

```
1 package Robot;
2
3 public class State {
4     static final int numOfLevelForDistance = 5;
5     static final int disForTooCloseToWall = 100;
6     static final int numOfLevelForEnergy = 5;
7
8     public static final int possibleStates = numOfLevelForDistance * numOfLevelForEnergy * numOfLevelForEnergy * 2 * 2;
9     private int disL;
10    private int isCloseToW;
11    private int myEL;
12    private int oEL;
13    private int isFaster;
14    public State(double myE, double oE, double myX, double oD, double myV, double oV){
15        // disL: distance level between the enemy and our robot: low(1), high(numOfLevelForDistance)
16        // closeToW: if it is too close to wall: yes(1), no(-2)
17        // elMy: my energy level: low(1), high(numOfLevelForEnergy)
18        // elo: enemy's energy level: low(1), high(numOfLevelForEnergy)
19        // total possible state: numOfLevelForDistance * numOfLevelForEnergy * numOfLevelForEnergy * 2 * 2
20        this.myEL = computeEnergyLevel(myE);
21        this.oEL = computeEnergyLevel(oE);
22        this.disL = computeDistanceLevel(oD);
23        this.isCloseToW = computeTooCloseToWall(myX, myY);
24        this.isFaster = computeIsFaster(oV, myV);
25    }
26
27    public State(int myE, int oEL, int disL, int isCloseToW, int isFaster){
28        this.myEL = myE;
29        this.oEL = oEL;
30        this.disL = disL;
31        this.isCloseToW = isCloseToW;
32        this.isFaster = isFaster;
33    }
34
35    /*
36     * return the index for this state (among all possible states)
37     */
38    public int getIndex(int actionIndex){
39        int tempForisCloseToW = 0;
40        int tempForisFaster = 0;
41        if(this.isCloseToW == -1){
42            tempForisCloseToW = 1;
43        }else{
44            tempForisCloseToW = 2;
45        }
46
47        if(this.isFaster == -1){
48            tempForisFaster = 1;
49        }else{
50            tempForisFaster = 2;
51        }
52
53        //int actionIndex = a.ordinal();
54        //int numActions = Action.values().length;
55        //return this.myEL*this.oEL*this.disL*tempForisCloseToW*tempForisFaster*numActions + actionIndex;
56
57        int NUM_ACTIONS = Action.values().length;
58        return (myEL-1) * (numOfLevelForEnergy * numOfLevelForDistance * 2 * 2 * NUM_ACTIONS)
59            + (oEL-1) * (numOfLevelForDistance * 2 * 2 * NUM_ACTIONS)
60            + (disL-1) * (2 * NUM_ACTIONS)
61            + (tempForisCloseToW-1) * (2 * NUM_ACTIONS)
62            + (tempForisFaster-1) * NUM_ACTIONS
63            + actionIndex;
64    }
65
66    public double[] transformToX(){
67        double[] X = new double[5];
68        X[0] = this.myEL;
69        X[1] = this.oEL;
70        X[2] = this.disL;
71        X[3] = this.isCloseToW;
72        X[4] = this.isFaster;
73        return X;
74    }
75
76    public double[] transformToX(Action a){
77        double[] X = new double[6];
78        X[0] = this.myEL;
79        X[1] = this.oEL;
80        X[2] = this.disL;
```

```

82     X[3] = this.isCloseToW;
83     X[4] = this.isFaster;
84     X[5] = a.ordinal();
85     return X;
86 }
87
88 private int computeIsFaster(double oV, double myV) {
89     if (oV < myV){
90         return 1; // faster than the opponent
91     }else{
92         return -1;
93     }
94 }
95
96 private int computeEnergyLevel(double e) {
97     double ratio = e / 100.0;
98     int output = (int) Math.ceil(1 + ratio * (numOfLevelForEnergy-1));
99     return output >5? 5: output; //energy can actually go beyond 100
100    //return (int) Math.ceil(1 + ratio * (numOfLevelForEnergy-1));
101    //return numOfLevelForEnergy - (int) Math.round(myE / numOfLevelForEnergy);
102 }
103
104 private int computeTooCloseToWall(double myX, double myY) {
105     double YtoWall = Math.min(myY, 600-myY);
106     double XtoWall = Math.min(myX, 800-myX);
107     double toWall = Math.min(YtoWall, XtoWall);
108     if (toWall < disForTooCloseToWall){
109         return 1; // to close to wall!
110     }else{
111         return -1;
112     }
113 }
114
115 private int computeDistanceLevel(double oD) {
116     double ratio = oD / 1000.0;
117     return (int) Math.ceil(1 + ratio * (numOfLevelForDistance-1));
118     //return numOfLevelForDistance- (int) Math.round(oD/numOfLevelForDistance);
119 }
120
121 }
122 }
```

Action.java

```
1 package Robot;  
2  
3 public enum Action {  
4     //    MOVE_UP,  
5     //    MOVE_DOWN,  
6     RUN_AWAY,  
7     ATTACK;  
8     //    CHASE;  
9  
10 }  
11
```

LUTInterface.java

```
1 package Interface;
2
3 public interface LUTInterface extends CommonInterface {
4
5
6     /**
7      * Initialise the look up table to all zeros.
8      */
9     public void initialiseLUT();
10
11
12
13     /**
14      * A helper method that translates a vector being used to index the
15      * look up table into an ordinal that can then be used to access
16      * the associated look up table element.
17      * @param X The state action vector used to index the LUT.LUT
18      * @return The index where this vector maps to
19      */
20     public int indexFor(double [] X);
21
22 }
```