Red State Machine Class

```java
package org.firstinspires.ftc.teamcode;

import com.qualcomm.hardware.bosch.BNO055IMU;
import com.qualcomm.robotcore.hardware.CRServo;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorSimple;
import com.qualcomm.robotcore.hardware.Gamepad;
import com.qualcomm.robotcore.hardware.HardwareMap;
import com.qualcomm.robotcore.hardware.Servo;
import com.qualcomm.robotcore.util.ElapsedTime;

public class RedStates
{

    /*This class is made by FTC #12535 Revolutionary Robots for our robot Armstrong. The variables and
    methods you find here are for the state machines used to control the robot with help from the
    Drive Train and Armstrong Classes.*/

    //Gamepad Variables
    Gamepad gamepad1;
    Gamepad gamepad2;

    //Wheel Motors
    DcMotor leftFront;
    DcMotor rightFront;
    DcMotor leftBack;
    DcMotor rightBack;

    //Foundation Servos
    Servo leftFoundation;
    Servo rightFoundation;

    //Turret Motor
    DcMotor turret;

    //Lift Motor
    DcMotor lift;

    //Arm Motor
    DcMotor arm;
```

Red State Machine Class

```
41
42      //Claw Continuous Rotation Servo
43      CRServo leftClaw;
44      CRServo rightClaw;
45
46      //Robot Classes
47      DriveTrain dT;
48      Armstrong a;
49
50      //Timer to perform timed motor control
51      ElapsedTime runtime;
52
53      //Enumerations for the entire set of autonomous programs
54      private enum AutoStates
55      {
56
57          BEGIN, NAVIGATE_TO_STONE, GRAB_STONE,
    MOVE_TO_FOUNDATION, PLACE_STONE, MOVE_FOUNDATION,
    NAVIGATE_TO_SECOND_STONE, GRAB_SECOND_STONE,
    MOVE_TO_FOUNDATION_SSECOND, PLACE_SECOND_STONE, PARK
58
59      }
60
61      //Variable to use in the state machines
62      AutoStates cOS = AutoStates.BEGIN;
63
64      //String to display telemetry with
65      String currentState = "begin";
66
67      //Enumerations for the wheel base options
68      private enum DriveStates
69      {
70
71          BEGIN, FORWARD, BACKWARD, RIGHT, LEFT, TURN_RIGHT,
    TURN_LEFT, IDLE
72
73      }
74
75      //Variable to use in the state machines
76      DriveStates cDS = DriveStates.BEGIN;
77
78      //Enumerations for the Foundation Servos
```

Red State Machine Class

```
79      private enum FoundationGrabberStates
80      {
81
82          BEGIN, GRAB, RELEASE, IDLE
83
84      }
85
86      //Variable to use inside of the state machines
87      FoundationGrabberStates cFGS = FoundationGrabberStates.
    BEGIN;
88
89      //Enumerations of turret actions
90      private enum TurretStates
91      {
92
93          BEGIN, TURN_RIGHT, TURN_LEFT, IDLE
94
95      }
96
97      //Variables to use inside of the state machines
98      TurretStates cTS = TurretStates.BEGIN;
99
100     //Enumerations of the actions the lift can do
101     private enum LiftStates
102     {
103
104         BEGIN, RAISE, LOWER, IDLE
105
106     }
107
108     //Variable to use inside of the state machines
109     LiftStates cLS = LiftStates.BEGIN;
110     LiftStates pLS;
111
112     //Enumerations for the arm options
113     private enum ArmStates
114     {
115
116         BEGIN, RAISE, LOWER, IDLE
117
118     }
119
```

Red State Machine Class

```
120     //Variables to use in the state machines
121     ArmStates cAS = ArmStates.BEGIN;
122
123     //Enumerations for the claw actions
124     private enum ClawStates
125     {
126
127         BEGIN, GRABBING, RELEASING, GRABBED, RELEASED, IDLE
128
129     }
130
131     //Variable to use in the state machines
132     ClawStates cCS = ClawStates.BEGIN;
133
134     public RedStates(DcMotor lF, DcMotor rF, DcMotor lB,
    DcMotor rB, Servo lFo, Servo rFo, DcMotor t, DcMotor l,
    DcMotor ar, CRServo lC, CRServo rC, DriveTrain dt, Armstrong
    armstrong, ElapsedTime rt)
135     {
136
137         /*Constructor for our class to be used in our auto
    programs.
138          *Assigns the variables that we set up above using
    the
139          *configuration to assign motors to motors and so on.
140          */
141
142         leftFront = lF;
143         rightFront = rF;
144         leftBack = lB;
145         rightBack = rB;
146
147         leftFoundation = lFo;
148         rightFoundation = rFo;
149
150         turret = t;
151
152         lift = l;
153
154         arm = ar;
155
156         leftClaw = lC;
```

```
157         rightClaw = rC;
158
159         dT = dt;
160
161         a = armstrong;
162
163         runtime = rt;
164
165     }
166
167     void runFull ()
168     {
169
170         //Method to run the state machine
171
172         //Looks at what cOS equals
173         switch (cOS)
174         {
175
176             //If cOS equals BEGIN
177             case BEGIN:
178
179                 //Runs method below
180                 begin();
181
182                 //Exits case
183                 break;
184
185             //If cOS equals NAVIGATE_TO_STONE
186             case NAVIGATE_TO_STONE:
187
188                 //Runs method below
189                 navigateToStoneFull();
190
191                 //Exits case
192                 break;
193
194         }
195
196     }
197
198     void runInside ()
```

```
199     {
200
201         //Work in progress method for alternative programs
202
203         switch (cOS)
204         {
205
206             case BEGIN:
207
208                 begin();
209
210                 break;
211
212             case NAVIGATE_TO_STONE:
213
214                 navigateToStoneInside();
215
216                 break;
217
218         }
219
220     }
221
222     String getState ()
223     {
224
225         //Returns the state to the phone for telemetry
226         return currentState;
227
228     }
229
230     void begin ()
231     {
232
233         //Sets the state tracker
234         currentState = "begin";
235
236         //Sets cOS to run to the next case/state
237         cOS = AutoStates.NAVIGATE_TO_STONE;
238
239     }
240
```

```
241     void navigateToStoneFull ()
242     {
243
244         //Move to the stone "wall" to detect a skystone
245
246         //Checks to see if the states are finished
247         if (cDS != DriveStates.IDLE && cLS != LiftStates.IDLE
    && cAS != ArmStates.IDLE)
248         {
249
250             //Looks at what cDS equals
251             switch (cDS)
252             {
253
254                 //If cDS equals BEGIN
255                 case BEGIN:
256
257                     //Runs method below
258                     cDSBegin1();
259
260                     //Exits case
261                     break;
262
263                 //If cDS equals FORWARD
264                 case FORWARD:
265
266                     //Runs method below
267                     cDSForwardFull();
268
269                     //Exits case
270                     break;
271
272                 //If cDS equals IDLE
273                 case IDLE:
274
275                     //Exits case
276                     break;
277
278             }
279
280             //The state machines below do the same as the one
    above
```

```
281
282             switch (cLS)
283             {
284
285                 case BEGIN:
286
287                     cLSBegin();
288
289                     break;
290
291                 case LOWER:
292
293                     cLSLower();
294
295                     break;
296
297                 case IDLE:
298
299                     break;
300
301
302             }
303
304             switch (cAS)
305             {
306
307                 case BEGIN:
308
309                     cASBegin();
310
311                     break;
312
313                 case RAISE:
314
315                     cASRaise();
316
317                     break;
318
319                 case IDLE:
320
321                     break;
322
```

```
323             }
324
325         } else
326         {
327
328             //When all states are idle
329
330             //Sets AutoStates to the next state
331             cOS = AutoStates.MOVE_TO_FOUNDATION;
332             //Telemetry for the phone
333             currentState = "finished";
334
335         }
336
337     }
338
339     void navigateToStoneInside ()
340     {
341
342         //work in progress method
343
344         currentState = "Stone";
345
346         if (cDS != DriveStates.IDLE && cLS != LiftStates.IDLE
    && cAS != ArmStates.IDLE)
347         {
348
349             switch (cDS)
350             {
351
352                 case BEGIN:
353
354                     cDSBegin1();
355
356                     break;
357
358                 case FORWARD:
359
360                     cDSForwardInside();
361
362                     break;
363
```

```
364                 case LEFT:
365
366                     cDSLeft();
367
368                     break;
369
370                 case IDLE:
371
372                     break;
373
374             }
375
376             switch (cLS)
377             {
378
379                 case BEGIN:
380
381                     cLSBegin();
382
383                     break;
384
385                 case LOWER:
386
387                     cLSLower();
388
389                     break;
390
391                 case IDLE:
392
393                     break;
394
395
396             }
397
398             switch (cAS)
399             {
400
401                 case BEGIN:
402
403                     cASBegin();
404
405                     break;
```

```
406
407                 case RAISE:
408
409                     cASRaise();
410
411                     break;
412
413                 case IDLE:
414
415                     break;
416
417             }
418
419         } else
420         {
421
422             cOS = AutoStates.MOVE_TO_FOUNDATION;
423             currentState = "finished";
424
425         }
426
427     }
428
429     void cDSBegin1 ()
430     {
431
432         //Sets the robot to move forward
433         dT.forwardNoStop(.25, 850);
434
435         //Sets state to FORWARD
436         cDS = DriveStates.FORWARD;
437
438     }
439
440     void cDSForwardFull ()
441     {
442
443         //Waits for the encoders to reach target position
444         if (!leftFront.isBusy() && !rightFront.isBusy() && !
    leftBack.isBusy() && !rightBack.isBusy())
445         {
446
```

```
447             //Stops wheelbase
448             dT.kill();
449 
450             //Sets wheelbase to idle
451             cDS = DriveStates.IDLE;
452 
453         }
454 
455     }
456 
457     void cDSForwardInside ()
458     {
459 
460         //Waits for the encoders to reach target position
461         if (!leftFront.isBusy() && !rightFront.isBusy() && !
    leftBack.isBusy() && !rightBack.isBusy())
462         {
463 
464             //Stops wheelbase
465             dT.kill();
466 
467             //Sets the drive train states to LEFT
468             cDS = DriveStates.LEFT;
469 
470             //Moves wheelbase to the left
471             dT.leftNoStop(0.25, 500);
472 
473         }
474 
475     }
476 
477     void cDSLeft ()
478     {
479 
480         //Waits for the encoders to reach target position
481         if (!leftFront.isBusy() && !rightFront.isBusy() && !
    leftBack.isBusy() && !rightBack.isBusy())
482         {
483 
484             //Stops wheelbase
485             dT.kill();
486 
```

```
487             //Sets wheel base to idle
488             cDS = DriveStates.IDLE;
489 
490         }
491 
492     }
493 
494     void cLSBegin ()
495     {
496 
497         //Resets Timer
498         runtime.reset();
499 
500         //Sets power of the lift to 0.75
501         lift.setPower(0.75);
502 
503         //Sets LiftStates to LOWER
504         cLS = LiftStates.LOWER;
505 
506     }
507 
508     void cLSLower ()
509     {
510 
511         //Waits for the timer to hit half a second or 500
    milliseconds
512         if (runtime.milliseconds() >= 500)
513         {
514 
515             //Stops lift
516             lift.setPower(0);
517 
518             //Sets lift to idle
519             cLS = LiftStates.IDLE;
520 
521         }
522 
523     }
524 
525     void cASBegin ()
526     {
527 
```

```
528             //Sets arm to rotate up
529             a.rUpNoStop(0.25, 250);
530 
531             //Sets state to RAISE
532             cAS = ArmStates.RAISE;
533 
534         }
535 
536         void cASRaise ()
537         {
538 
539             //Waits until arm encoder hits target position
540             if (!arm.isBusy())
541             {
542 
543                 //Stops arm
544                 arm.setPower(0);
545 
546                 //Sets state to idle
547                 cAS = ArmStates.IDLE;
548 
549             }
550 
551         }
552 
553 }
```