

```

1  /*****
2  Module
3      MasterVehidle.c (based on TopHSMTemplate.c)
4
5  Description
6      This is a code for the top level Hierarchical state machine
7
8  *****/
9  /*----- Include Files -----*/
10 /* include header files for this state machine as well as any machines at the
11    next lower level in the hierarchy that are sub-machines to this machine
12 */
13 #include "ES_Configure.h"
14 #include "ES_Framework.h"
15 #include "MasterVehicle.h"
16 #include "ES_DeferRecall.h"
17 #include "inc/hw_memmap.h"
18 #include "inc/hw_types.h"
19 #include "inc/hw_gpio.h"
20 #include "inc/hw_sysctl.h"
21 #include "driverlib/sysctl.h"
22 #include "driverlib/pin_map.h" // Define PART_TM4C123GH6PM in project
23 #include "driverlib/gpio.h"
24 #include "inc/hw_timer.h"
25 #include "inc/hw_nvic.h"
26 #include "StagesSM.h"
27 #include "ShootSM.h"
28 #include "SupplySM.h"
29 #include "Location.h"
30 #include "OwnPWM.h"
31 #include "LOCMaster.h"
32 #include "Servo_Control.h"
33 #include "Ultrasonic.h"
34
35 /* Team Definition */
36 #define RED 0
37 #define GREEN 1
38
39 /* Time Definition*/
40 #define ONE_SEC 976
41 #define HALF_SEC (ONE_SEC /2)
42 #define TWO_SEC (ONE_SEC *2)
43 #define FIVE_SEC (ONE_SEC *5)
44 #define TEN_SEC (ONE_SEC *10)
45 #define TWENTY_SEC (ONE_SEC * 20)
46 #define FORTY_FIVE_DEGREE_ROTATE_TIME 600
47 #define TWO_MIN (ONE_SEC * 120)
48
49 /* PWM */
50 #define PWM_FREQ 1000
51
52 /* Status related masks*/
53 #define GREEN_SB1_MASK (BIT0HI|BIT1HI|BIT2HI|BIT3HI) << 20
54 #define RED_SB1_MASK (BIT0HI|BIT1HI|BIT2HI|BIT3HI) << 16
55 #define GREEN_SCORE_MASK (BIT0HI|BIT1HI|BIT2HI|BIT3HI|BIT4HI|BIT5HI) << 8
56 #define RED_SCORE_MASK (BIT0HI|BIT1HI|BIT2HI|BIT3HI|BIT4HI|BIT5HI)
57 #define GAME_STATUS_MASK (BIT7HI)
58
59 #define NUM_BALL_START 4
60
61
62 /*----- Module Defines -----*/
63
64 /*----- Module Functions -----*/
65 static ES_Event DuringInitState( ES_Event Event);
66 static ES_Event DuringIdleState( ES_Event Event);
67 static ES_Event DuringStagingState( ES_Event Event);
68 static ES_Event DuringShootingState( ES_Event Event);
69 static ES_Event DuringSupplyingState( ES_Event Event);
70 static void Init_LED_Port(void);
71 static void Team_Selection(void);
72 static void Set_Construction_LED(void);

```

```

73  static void Clear_Construction_LED(void);
74  static void update_Status(void);
75  uint32_t get_destination_y(void);
76  uint32_t get_destination_x(void);
77
78  /*----- Module Variables -----*/
79  // everybody needs a state variable, though if the top level state machine
80  // is just a single state container for orthogonal regions, you could get
81  // away without it
82  static MasterVehicleState_t CurrentState;
83  static uint8_t MyPriority;
84  static int team;
85  static uint32_t status = 0;
86  static uint32_t destination_x = 0;
87  static uint32_t destination_y = 0;
88  uint32_t frequency = 0;
89  static const uint32_t PERIOD_TABLE[16] = {1333,1277,1222,1166,1111,1055,1000
90  ,944,889,833,778,722,667,611,556,500};
91  static const int period_table_length = 16;
92  static int NUM_BALL_AVAILABLE;
93  static uint8_t green_score = 0;
94  static uint8_t red_score = 0;
95
96  /* Coordinate Variables for the locations on the stage (ME218B Winter 2017) */
97
98  // Green Coordinates
99  static const uint32_t StagingXCoordinateEncoderGreen[3]={58,85,59};
100 static const uint32_t StagingYCoordinateEncoderGreen[3]={23,147,197};
101
102 static const uint32_t ShootingXCoordinateEncoderGreen[3]={62,62,62};
103 static const uint32_t ShootingYCoordinateEncoderGreen[3]={17,59,191};
104
105 static const uint32_t green_supply_x = 24;
106 static const uint32_t green_supply_y = 2;
107
108 // Red Coordinates
109 static const uint32_t StagingXCoordinateEncoderRed[3]={37,12,37};
110 static const uint32_t StagingYCoordinateEncoderRed[3]={23,52,197};
111
112 static const uint32_t ShootingXCoordinateEncoderRed[3]={37,37,37};
113 static const uint32_t ShootingYCoordinateEncoderRed[3]={17,128,191};
114
115 static const uint32_t red_supply_x = 69;
116 static const uint32_t red_supply_y = 2;
117
118 /*----- Module Code -----*/
119 /*****
120  Function
121      InitMasterSM
122
123  Parameters
124      uint8_t : the priority of this service
125
126  Returns
127      boolean, False if error in initialization, True otherwise
128
129  Description
130      Saves away the priority, and starts
131      the top level state machine
132
133  *****/
134 bool InitMasterVehicleSM ( uint8_t Priority )
135 {
136     ES_Event ThisEvent;
137     MyPriority = Priority; // save our priority
138     ThisEvent.EventType = ES_ENTRY;
139
140     // Initialize port and pin needed
141     Init_LED_Port();
142
143     // Initialize location module
144     Init_Location();

```

```

145
146     // Initialize PWM for Servo
147     InitServo();
148
149     // Start the Master State machine
150     StartMasterVehicleSM( ThisEvent );
151
152     //Initially start with 4 balls
153     NUM_BALL_AVAILABLE = NUM_BALL_START;
154
155     return true;
156 }
157
158 /*****
159  Function
160      PostMasterSM
161
162  Parameters
163      ES_Event ThisEvent , the event to post to the queue
164
165  Returns
166      boolean False if the post operation failed, True otherwise
167
168  Description
169      Posts an event to this state machine's queue
170
171  *****/
172 bool PostMasterVehicleSM( ES_Event ThisEvent )
173 {
174     return ES_PostToService( MyPriority, ThisEvent);
175 }
176
177 /*****
178  Function
179      RunMasterVehicleSM
180
181  Parameters
182      ES_Event: the event to process
183
184  Returns
185      ES_Event: an event to return
186
187  Description
188      the run function for the top level state machine
189
190  *****/
191 ES_Event RunMasterVehicleSM( ES_Event CurrentEvent )
192 {
193     bool MakeTransition = false; /* are we making a state transition? */
194     MasterVehicleState_t NextState = CurrentState;
195     ES_Event EntryEventKind = { ES_ENTRY, 0 }; // default to normal entry to new state
196     ES_Event ReturnEvent = { ES_NO_EVENT, 0 }; // assume no error
197
198     switch ( CurrentState )
199     {
200         case INIT_STATE : // If current state is Init State
201             // Execute During function for state one. ES_ENTRY & ES_EXIT are
202             // processed here allow the lower level state machines to re-map
203             // or consume the event
204             CurrentEvent = DuringInitState(CurrentEvent);
205             //process any events
206             if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
207             {
208                 switch (CurrentEvent.EventType)
209                 {
210                     case CONSTRUCTION_START: //If event is event one
211                         // Execute action function for state one : event one
212                         NextState = IDLE_STATE; //Decide what the next state will be
213                         // for internal transitions, skip changing MakeTransition
214                         MakeTransition = true; //mark that we are taking a transition
215                         // if transitioning to a state with history change kind of entry
216                         // EntryEventKind.EventType = ES_ENTRY_HISTORY;

```

```

217         // optionally, consume or re-map this event for the upper
218         // level state machine
219         ReturnEvent.EventType = ES_NO_EVENT;
220         break;
221
222     default:
223         break;
224 }
225 }
226 break;
227
228 case IDLE_STATE : // If current state is idle state
229     // Execute During function for state one. ES_ENTRY & ES_EXIT are
230     // processed here allow the lower level state machines to re-map
231     // or consume the event
232     CurrentEvent = DuringIdleState(CurrentEvent);
233     //process any events
234
235     if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
236     {
237
238         switch (CurrentEvent.EventType)
239         {
240             case STAGE_ACTIVE: //If event is stage active
241                 update_Status();
242                 // update game status to reflect current active staging area
243                 NextState = STAGING_STATE; //set next state to staging
244                 MakeTransition = true; //mark that we are taking a transition
245                 ReturnEvent.EventType = ES_NO_EVENT;
246                 break;
247
248             case SHOOT_ACTIVE: //If event is shoot active
249                 update_Status();
250                 // update game status to reflect current active shooting area
251                 NextState = SHOOTING_STATE; //set next state to shooting
252                 MakeTransition = true; //mark that we are taking a transition
253                 ReturnEvent.EventType = ES_NO_EVENT;
254                 break;
255
256             case SHOOT_ACTIVE_4: //If event is shoot active (for last 18 seconds)
257                 update_Status();
258                 // update game status to reflect current active staging area
259                 NextState = SHOOTING_STATE; //set next state to shooting
260                 MakeTransition = true; //mark that we are taking a transition
261                 ReturnEvent.EventType = ES_NO_EVENT;
262                 break;
263
264             case NO_BALL: //If event is event one
265                 update_Status();
266                 // update game status to reflect current active staging area
267                 NextState = SUPPLYING_STATE; //set next state to supplying
268                 MakeTransition = true; //mark that we are taking a transition
269                 ReturnEvent.EventType = ES_NO_EVENT;
270                 break;
271
272             case CONSTRUCTION_END: //If event is construction end
273                 NextState = INIT_STATE;
274                 //set next state to init state (we want to stop everything)
275                 MakeTransition = true; //mark that we are taking a transition
276                 ReturnEvent.EventType = ES_NO_EVENT;
277                 break;
278
279             default:
280                 break;
281         }
282     }
283     break;
284
285 case STAGING_STATE : // If current state is staging state
286     // Execute During function for state one. ES_ENTRY & ES_EXIT are
287     // processed here allow the lower level state machines to re-map
288     // or consume the event

```

```

289     CurrentEvent = DuringStagingState(CurrentEvent);
290     //process any events
291     if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
292     {
293         switch (CurrentEvent.EventType)
294         {
295
296             case ES_TIMEOUT: //if event is timeout, we go back to idle state
297                 NextState = IDLE_STATE; //set next state to idle state
298                 MakeTransition = true; //mark that we are taking a transition
299                 ReturnEvent.EventType = ES_NO_EVENT;
300                 check_active_event(); //check if there is any active event on our way out
301                 break;
302
303             case SHOOT_ACTIVE_4: //if event is shoot active (during the last 18 seconds)
304                 NextState = IDLE_STATE; //set next state to idle state
305                 MakeTransition = true; //mark that we are taking a transition
306                 ReturnEvent.EventType = ES_NO_EVENT;
307                 check_active_event(); //check if there is any active event on our way out
308                 break;
309
310             case FINISHED_STAGING: //if we successfully verify frequency
311                 NextState = IDLE_STATE; //set next state to idle state
312                 MakeTransition = true; //mark that we are taking a transition
313                 ReturnEvent.EventType = ES_NO_EVENT;
314                 break;
315
316             case CONSTRUCTION_END: //If event is construction end
317                 NextState = INIT_STATE; //set next state to init state
318                 MakeTransition = true; //mark that we are taking a transition
319                 ReturnEvent.EventType = ES_NO_EVENT;
320                 break;
321
322             default:
323                 break;
324         }
325     }
326     break;
327
328 case SHOOTING_STATE : // If current state is shooting state
329     // Execute During function for state one. ES_ENTRY & ES_EXIT are
330     // processed here allow the lower level state machines to re-map
331     // or consume the event
332     CurrentEvent = DuringShootingState(CurrentEvent);
333     //process any events
334     if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
335     {
336         switch (CurrentEvent.EventType)
337         {
338             case ES_TIMEOUT: //If event is 20 SECOND SHOOTING TIMEOUT
339                 printf("ES TIMEOUT\r\n");
340                 NextState = IDLE_STATE; //set next state to idle state
341                 MakeTransition = true; //mark that we are taking a transition
342                 ReturnEvent.EventType = ES_NO_EVENT;
343                 check_active_event(); //check if there is any active event on our way out
344                 break;
345
346             case SCORE_CHANGED: //If event is SCORE_CHANGED
347                 printf("Score Changed\r\n");
348                 NextState = IDLE_STATE; //set next state to idle state
349                 MakeTransition = true; //mark that we are taking a transition
350                 ReturnEvent.EventType = ES_NO_EVENT;
351                 check_active_event(); //check if there is any active event on our way out
352                 break;
353
354             case NO_BALL: //If event is no ball
355                 printf("No Ball\r\n");
356                 NextState = SUPPLYING_STATE; //set next state to supply state
357                 MakeTransition = true; //mark that we are taking a transition
358                 ReturnEvent.EventType = ES_NO_EVENT;
359                 break;
360

```

```

361         case CONSTRUCTION_END: //If event is construction end
362             printf("Construction Ends\r\n");
363             NextState = INIT_STATE; //set next state to init state
364             MakeTransition = true; //mark that we are taking a transition
365             ReturnEvent.EventType = ES_NO_EVENT;
366             break;
367
368         default:
369             break;
370     }
371 }
372 break;
373
374 case SUPPLYING_STATE : // If current state is supplying state
375     // Execute During function for state one. ES_ENTRY & ES_EXIT are
376     // processed here allow the lower level state machines to re-map
377     // or consume the event
378     CurrentEvent = DuringSupplyingState(CurrentEvent);
379     //process any events
380     if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
381     {
382         switch (CurrentEvent.EventType)
383         {
384             case LOADED_COMPLETE: //If event is loaded complete
385                 NextState = IDLE_STATE; //set next state to idle state
386                 MakeTransition = true; //mark that we are taking a transition
387                 ReturnEvent.EventType = ES_NO_EVENT;
388                 break;
389
390             case CONSTRUCTION_END: //If event is construction end
391                 NextState = INIT_STATE; //set next state to init state
392                 MakeTransition = true; //mark that we are taking a transition
393                 ReturnEvent.EventType = ES_NO_EVENT;
394                 break;
395
396             default:
397                 break;
398         }
399     }
400     break;
401
402 default:
403     break;
404     // repeat state pattern as required for other states
405 }
406
407 // If we are making a state transition
408 if (MakeTransition == true)
409 {
410     // Execute exit function for current state
411     CurrentEvent.EventType = ES_EXIT;
412     RunMasterVehicleSM(CurrentEvent);
413
414     CurrentState = NextState; //Modify state variable
415
416     // Execute entry function for new state
417     // this defaults to ES_ENTRY
418     RunMasterVehicleSM(EntryEventKind);
419 }
420
421 // in the absence of an error the top level state machine should
422 // always return ES_NO_EVENT, which we initialized at the top of function
423 return (ReturnEvent);
424 }
425
426
427 /*****
428 Function
429     StartMasterVehicleSM
430
431 Parameters
432     ES_Event CurrentEvent

```

```

433
434     Returns
435     nothing
436
437     Description
438     Does any required initialization for this state machine
439
440     *****/
441 void StartMasterVehicleSM ( ES_Event CurrentEvent )
442 {
443     // if there is more than 1 state to the top level machine you will need
444     // to initialize the state variable
445     CurrentState = INIT_STATE; //start currentstate with init state
446     // now we need to let the Run function init the lower level state machines
447     // use LocalEvent to keep the compiler from complaining about unused var
448     RunMasterVehicleSM(CurrentEvent);
449     return;
450 }
451
452
453 /* This function return the current state of this state machine */
454 MasterVehicleState_t QueryMasterVehicleSM ( void )
455 {
456     return(CurrentState);
457 }
458
459 /*****
460 private functions
461 *****/
462
463 static ES_Event DuringInitState( ES_Event Event)
464 {
465     ES_Event ReturnEvent = Event;
466     if ( (Event.EventType == ES_ENTRY) || (Event.EventType == ES_ENTRY_HISTORY) ){
467         Team_Selection(); //read team input and set team collar
468         Clear_Construction_LED(); //make sure construction led is off
469         stopMotor(); //make sure motor is not running when we start the game
470     }
471     else if ( Event.EventType == ES_EXIT ){
472         //when we exit init state, that means game is on. So turn on the led
473         Set_Construction_LED();
474     }
475     else{
476         // No lower state machine
477     }
478     return ReturnEvent;
479 }
480
481
482 static ES_Event DuringIdleState( ES_Event Event)
483 {
484     ES_Event ReturnEvent = Event;
485     return ReturnEvent;
486 }
487
488 static ES_Event DuringStagingState( ES_Event Event)
489 {
490     ES_Event ReturnEvent = Event;
491     if ( (Event.EventType == ES_ENTRY) || (Event.EventType == ES_ENTRY_HISTORY) ){
492         //when we enter this state, post STAGE_ACTIVE event to StageSM
493         Event.EventType = STAGE_ACTIVE;
494         StartStageSM(Event);
495     }
496     else if ( Event.EventType == ES_EXIT ){
497         RunStageSM(Event);
498         //We have 2 sec limit in staging SM, we want to stop this timer
499         //when we are in shooting or supplying state
500         ES_Timer_StopTimer(STAGE_TIMER);
501     }
502     else{
503         ReturnEvent = RunStageSM(Event);
504     }

```

```

505     return ReturnEvent;
506 }
507
508 static ES_Event DuringShootingState( ES_Event Event)
509 {
510     ES_Event ReturnEvent = Event;
511     if ( (Event.EventType == ES_ENTRY) || (Event.EventType == ES_ENTRY_HISTORY) ){
512         //when we enter this state, post SHOOT_ACTIVE event to ShootSM
513         Event.EventType = SHOOT_ACTIVE;
514         StartShootSM(Event);
515         // start 20 seconds timer limit for shooting
516         ES_Timer_InitTimer(SHOOT_20S_TIMER, TWENTY_SEC);
517     }
518     else if ( Event.EventType == ES_EXIT ){
519         RunShootSM(Event);
520     }
521     else{
522         ReturnEvent = RunShootSM(Event);
523     }
524     return ReturnEvent;
525 }
526
527 static ES_Event DuringSupplyingState( ES_Event Event)
528 {
529     ES_Event ReturnEvent = Event;
530     if ( (Event.EventType == ES_ENTRY) || (Event.EventType == ES_ENTRY_HISTORY) ){
531         //when we enter this state, post NO_BALL event to SupplySM
532         Event.EventType= NO_BALL;
533         StartSupplySM(Event);
534     }
535     else if ( Event.EventType == ES_EXIT ){
536         RunSupplySM(Event);
537         //we we exit, we want to check which event is currently active
538         check_active_event();
539     }
540     else{
541         ReturnEvent = RunSupplySM(Event);
542     }
543     return ReturnEvent;
544 }
545
546 static void Init_LED_Port(){
547     //Initializing Port B, E, F for led and team selection switch
548     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
549     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
550     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
551     GPIOPinTypeGPIOInput(GPIO_PORTB_BASE, GPIO_PIN_0); //TEAM SELECTION SWITCH
552     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_1); //TEAM RED LED
553     GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2); //RESUPPLYING LED
554     GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_3); //CONSTRUCTION LED
555     GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_4); //TEAM GREEN LED
556
557     //Initialize PF0
558     HWREG(GPIO_PORTF_BASE + GPIO_O_PUR) = HWREG(GPIO_PORTF_BASE + GPIO_O_PUR) | (1<<0);
559     HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
560     HWREG(GPIO_PORTF_BASE + GPIO_O_CR) = (1<<0);
561     GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_0); //IR LED (FOR RESUPPLY)
562
563     //Initialize PF1
564     HWREG(GPIO_PORTF_BASE + GPIO_O_PUR) = HWREG(GPIO_PORTF_BASE + GPIO_O_PUR) | (1<<1);
565     HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
566     HWREG(GPIO_PORTF_BASE + GPIO_O_CR) = (1<<0);
567     GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1); //BUMPER SWITCH
568 }
569
570 /* This function read value from PB0 and set team accordinly */
571 static void Team_Selection(){
572     //if PB0 is low, set team to RED
573     if((GPIOPinRead(GPIO_PORTB_BASE, GPIO_PIN_0) & BIT0HI) == 0){
574         team = RED;
575     }
576     //if PB0 is high, set team to GREEN

```



```

577     else{
578         team = GREEN;
579     }
580     if(team == RED){
581         //light up PB1 which is connected to red led
582         GPIOWrite(GPIO_PORTB_BASE, GPIO_PIN_1, BIT1HI);
583         GPIOWrite(GPIO_PORTF_BASE, GPIO_PIN_4, BIT4LO);
584     }
585     else if(team == GREEN){
586         //light up PF4 which is connected to green led
587         GPIOWrite(GPIO_PORTF_BASE, GPIO_PIN_4, BIT4HI);
588         GPIOWrite(GPIO_PORTB_BASE, GPIO_PIN_1, BIT1LO);
589     }
590 }
591
592 static void Set_Construction_LED(){
593     //Construction In Progress. set LED (set PF3)
594     GPIOWrite(GPIO_PORTF_BASE, GPIO_PIN_3, BIT3HI);
595 }
596
597 static void Clear_Construction_LED(){
598     //Clear Construction LED (clear PB4)
599     GPIOWrite(GPIO_PORTF_BASE, GPIO_PIN_3, BIT3LO);
600 }
601
602 static void update_Status(){
603     //Get status update from LOC master module
604     status = queryStatusBytes();
605 }
606
607 /* This function return the current team */
608 int get_Team(void){
609     return team;
610 }
611
612 /* This function return the current supply location (x)*/
613 uint32_t get_Supply_location_x(){
614     if (team == RED){
615         return red_supply_x;
616     }
617     else{
618         return green_supply_x;
619     }
620 }
621
622 /* This function return the current supply location (y)*/
623 uint32_t get_Supply_location_y(){
624     if (team == RED){
625         return red_supply_y;
626     }
627     else{
628         return green_supply_y;
629     }
630 }
631
632 /* The following functions return the loation of stage and shoot area for red and green */
633 uint32_t get_Stage_Green_X(uint8_t index){
634     //we have to subtract one from index as the input is
635     //from 1 to 4 but our array index start at 0
636     return StagingXCoordinateEncoderGreen[index-1];
637 }
638 uint32_t get_Stage_Green_Y(uint8_t index){
639     return StagingYCoordinateEncoderGreen[index-1];
640 }
641 uint32_t get_Stage_Red_X(uint8_t index){
642     return StagingXCoordinateEncoderRed[index-1];
643 }
644 uint32_t get_Stage_Red_Y(uint8_t index){
645     return StagingYCoordinateEncoderRed[index-1];
646 }
647 uint32_t get_Shoot_Green_X(uint8_t index){
648     return ShootingXCoordinateEncoderGreen[index-1];

```

```
649     }
650     uint32_t get_Shoot_Green_Y(uint8_t index){
651         return ShootingYCoordinateEncoderGreen[index-1];
652     }
653     uint32_t get_Shoot_Red_X(uint8_t index){
654         return ShootingXCoordinateEncoderRed[index-1];
655     }
656     uint32_t get_Shoot_Red_Y(uint8_t index){
657         return ShootingYCoordinateEncoderRed[index-1];
658     }
659
660     /* This function return the current number of ball */
661     int get_num_ball(void){
662         return NUM_BALL_AVAILABLE;
663     }
664
665     /* This function increases the current number of ball by 1*/
666     void increment_num_ball(void){
667         NUM_BALL_AVAILABLE++;
668     }
669
670     /* This function decreases the current number of ball by 1*/
671     void decrement_num_ball(void){
672         NUM_BALL_AVAILABLE--;
673     }
674
675     /* This function updates the score variable using the inputs */
676     void update_score(uint8_t red_input, uint8_t green_input){
677         red_score = red_input;
678         green_score = green_input;
679     }
680
681     /* This function returns the current red score */
682     uint8_t get_red_score(void){
683         return red_score;
684     }
685
686     /* This function returns the current green score */
687     uint8_t get_green_score(void){
688         return green_score;
689     }
690
691
```