```cpp
#pragma region VEXcode Generated Robot Configuration
// Make sure all required headers are included.
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
#include <string.h>


#include "vex.h"

using namespace vex;

// Brain should be defined by default
brain Brain;


// START IQ MACROS
#define waitUntil(condition)                                                   \
  do {                                                                         \
    wait(5, msec);                                                             \
  } while (!(condition))

#define repeat(iterations)                                                     \
  for (int iterator = 0; iterator < iterations; iterator++)
// END IQ MACROS


// Robot configuration code.
inertial BrainInertial = inertial();
motor MotorCatapultH = motor(PORT10, false);
motor MotorCatapultV = motor(PORT5, true);
optical Optical8 = optical(PORT8);
distance Distance1 = distance(PORT1);
bumper Bumper3 = bumper(PORT3);
touchled TouchLED9 = touchled(PORT9);
motor MotorPhoneJail = motor(PORT6, false);
motor MotorDispensor = motor(PORT2, false);


// generating and setting random seed
void initializeRandomSeed(){
  wait(100,msec);
  double xAxis = BrainInertial.acceleration(xaxis) * 1000;
  double yAxis = BrainInertial.acceleration(yaxis) * 1000;
  double zAxis = BrainInertial.acceleration(zaxis) * 1000;
  // Combine these values into a single integer
  int seed = int(
    xAxis + yAxis + zAxis
  );
  // Set the seed
  srand(seed);
}

// Converts a color to a string
const char* convertColorToString(color col) {
  if (col == colorType::red) return "red";
  else if (col == colorType::green) return "green";
  else if (col == colorType::blue) return "blue";
  else if (col == colorType::white) return "white";
  else if (col == colorType::yellow) return "yellow";
  else if (col == colorType::orange) return "orange";
  else if (col == colorType::purple) return "purple";
```

```
64      else if (col == colorType::cyan) return "cyan";
65      else if (col == colorType::black) return "black";
66      else if (col == colorType::transparent) return "transparent";
67      else if (col == colorType::red_violet) return "red_violet";
68      else if (col == colorType::violet) return "violet";
69      else if (col == colorType::blue_violet) return "blue_violet";
70      else if (col == colorType::blue_green) return "blue_green";
71      else if (col == colorType::yellow_green) return "yellow_green";
72      else if (col == colorType::yellow_orange) return "yellow_orange";
73      else if (col == colorType::red_orange) return "red_orange";
74      else if (col == colorType::none) return "none";
75      else return "unknown";
76    }
77
78
79    // Convert colorType to string
80    const char* convertColorToString(colorType col) {
81      if (col == colorType::red) return "red";
82      else if (col == colorType::green) return "green";
83      else if (col == colorType::blue) return "blue";
84      else if (col == colorType::white) return "white";
85      else if (col == colorType::yellow) return "yellow";
86      else if (col == colorType::orange) return "orange";
87      else if (col == colorType::purple) return "purple";
88      else if (col == colorType::cyan) return "cyan";
89      else if (col == colorType::black) return "black";
90      else if (col == colorType::transparent) return "transparent";
91      else if (col == colorType::red_violet) return "red_violet";
92      else if (col == colorType::violet) return "violet";
93      else if (col == colorType::blue_violet) return "blue_violet";
94      else if (col == colorType::blue_green) return "blue_green";
95      else if (col == colorType::yellow_green) return "yellow_green";
96      else if (col == colorType::yellow_orange) return "yellow_orange";
97      else if (col == colorType::red_orange) return "red_orange";
98      else if (col == colorType::none) return "none";
99      else return "unknown";
100   }
101
102
103   void vexcodeInit() {
104
105     // Initializing random seed.
106     initializeRandomSeed();
107   }
108
109   #pragma endregion VEXcode Generated Robot Configuration
110
111   //----------------------------------------------------------------------------
112   //
113   //
114   //  Vex Final Project Code (Group 15)
115   //
116
117  //
118   //----------------------------------------------------------------------------
119
120   // Include the IQ Library
121   #include "iq_cpp.h"
122   #include  "vex.h"
123   #include <string>
124
125   // Allows for easier use of the VEX Library
   using namespace vex;
```

```
126
127 void configureAllSensors(){
128   BrainInertial.calibrate();
129   wait(2,seconds);
130   BrainInertial.setHeading(0,degrees);
131   BrainInertial.setRotation(0,degrees);
132   MotorCatapultH.setPosition(0,turns);
133   MotorCatapultV.setPosition(0,turns);
134   Brain.Screen.clearScreen();
135   Brain.Screen.setFont(mono15);
136   //Optical3.setLight(ledState::on);
137 }
138
139 // ===== Constants =====
140 const int PERSON_THRESHOLD = 5;
141 const int SWEEP_SPEED = 10;
142 const int LAUNCH_SPEED = 100;
143 const int RESET_SPEED = 10;
144 const int SWEEP_DELAY = 50;
145 const int LAUNCH_DURATION = 2000;
146 const int RESET_DURATION = 2000;
147
148 void touchPressed()
149 {
150   while(!TouchLED9.pressing())
151   {}
152   while(TouchLED9.pressing())
153   {}
154 }
155
156
157 bool doorLock = false;
158 const int PJ_DOOR_SPEED = 50;
159
160 void phoneJailDoor()
161 {
162   Brain.Screen.clearScreen();
163   Brain.Screen.setCursor(1,1);
164   Brain.Screen.print("Press Touch ");
165   Brain.Screen.setCursor(2,1);
166   Brain.Screen.print("To Open/Close");
167   Brain.Screen.setCursor(3,1);
168   Brain.Screen.print("Phone Jail!");
169
170   touchPressed();
171
172   Brain.Screen.clearScreen();
173
174   if(doorLock)
175   {
176     Brain.Screen.clearScreen();
177     Brain.Screen.setCursor(1,1);
178     Brain.Screen.print("Closing Door!");
179     MotorPhoneJail.spinFor(forward, 615, degrees, PJ_DOOR_SPEED, velocityUnits::pc
t);
180
181   }
182
183   else
184   {
185     Brain.Screen.clearScreen();
186     Brain.Screen.setCursor(1,1);
187     Brain.Screen.print("Opening Door!");
```

```
188       MotorPhoneJail.spinFor(reverse, 615, degrees, PJ_DOOR_SPEED, velocityUnits::pc
t);
189     }
190
191     doorLock = !doorLock;
192   }
193
194
195   void monitorPhone()
196   {
197     Optical8.setLight(ledState::on);
198
199     bool phoneNotDetected = true;
200     while(phoneNotDetected){
201
202       if (Optical8.color() == blue)
203       {
204       Brain.Screen.clearScreen();
205       Brain.Screen.setCursor(1,1);
206       Brain.Screen.print("PUT YOUR PHONE BACK");
207       wait(1, seconds);
208
209       }
210
211       else
212       {
213       Brain.Screen.clearScreen();
214       Brain.Screen.setCursor(1,1);
215       Brain.Screen.print("GOOD JOB!");
216       wait(1, seconds);
217       phoneNotDetected = false;
218       }
219     }
220   }
221
222
223   std::string pickQuote()
224   {
225     std::string quotes[5] =
226     {
227       "KEEP GOING",
228       "YOU GOT THIS",
229       "LOCK IN TWIN",
230       "YOU ROCK",
231       "NEVER BACK DOWN"
232     };
233
234     int randomNum = 0;
235     randomNum = rand() % 5;
236
237     return quotes[randomNum];
238   }
239
240
241   void launchCatapult(int launchSpeed, int launchDur, int resetDur, int resetSpeed)
242   {
243     MotorCatapultV.setPosition(0, degrees);
244     MotorCatapultV.setVelocity(launchSpeed, percent);
245     MotorCatapultV.spin(forward);
246     wait(launchDur, msec);
247     MotorCatapultV.stop();
248
249     MotorCatapultV.setVelocity(resetSpeed, percent);
```

```
250     MotorCatapultV.spin(reverse);
251     MotorCatapultH.setPosition(0, degrees);
252     wait(resetDur, msec);
253     MotorCatapultV.stop();
254   }
255
256   void searchForPerson()
257   {
258     bool movingRight = true;
259     double angle = 0.0;
260
261     MotorCatapultH.setPosition(0, degrees);
262     MotorCatapultH.setVelocity(SWEEP_SPEED, percent);
263
264     while (true)
265     {
266       // Move horizontally
267       if (movingRight)
268       {
269         MotorCatapultH.spin(forward);
270         angle += 1;
271         if (angle >= 20)
272         {
273           movingRight = false;
274         }
275       }
276
277       else
278       {
279         MotorCatapultH.spin(reverse);
280         angle -= 1;
281
282         if (angle <= -20)
283         {
284           movingRight = true;
285         }
286       }
287
288       // Check distance:
289       double distance = Distance1.objectDistance(mm);
290
291       // If a person detected, launch:
292       if (distance > 0 && distance <= PERSON_THRESHOLD)
293       {
294         MotorCatapultH.stop();
295         launchCatapult(LAUNCH_SPEED, LAUNCH_DURATION, RESET_DURATION, RESET_SPEED);
296         break;
297       }
298
299       wait(SWEEP_DELAY, msec);
300     }
301
302     MotorCatapultH.setVelocity(RESET_SPEED, percent);
303
304     double currentPos = MotorCatapultH.position(degrees);
305
306     if (currentPos > 0)
307     {
308       MotorCatapultH.spin(reverse);
309       while (MotorCatapultH.position(degrees) > 0)
310       {}
311     }
312
```

```
313     else if (currentPos < 0)
314     {
315       MotorCatapultH.spin(forward);
316       while (MotorCatapultH.position(degrees) < 0)
317       {}
318     }
319
320     MotorCatapultH.stop();
321     MotorCatapultV.stop();
322   }
323
324   void dispensor(bool timer)
325   {
326       MotorDispensor.setPosition(0, degrees);
327
328       if( timer == true)
329       {
330           MotorDispensor.spin(forward, 10, percent);
331           while(MotorDispensor.position(degrees) <= 135)
332           {}
333           MotorDispensor.stop();
334
335           wait(1, msec);
336
337           MotorDispensor.spin(reverse, 10, percent);
338           while(MotorDispensor.position(degrees) > -10)
339           {}
340
341       MotorDispensor.stop();
342       }
343
344       else if ( timer == false )
345       {
346           MotorDispensor.spin(reverse, 10, percent);
347           while(MotorDispensor.position(degrees) >= -135)
348           {}
349           MotorDispensor.stop();
350
351           wait(2, seconds);
352
353           MotorDispensor.spin(forward, 10, percent);
354           while(MotorDispensor.position(degrees) < 10)
355           {}
356
357       MotorDispensor.stop();
358       }
359   }
360
361
362   void endBot()
363   {
364     if(TouchLED9.pressing() && Bumper3.pressing())
365     {
366       phoneJailDoor();
367       Brain.programStop();
368     }
369   }
370
371   // ----- Break duration -----
372   const int BREAK_DURATION_SEC = 5 * 60;
373
374   void runBreakSession()
375   {
```

```
376     Brain.Timer.reset();
377     bool breakDone = false;
378
379     while (!breakDone)
380     {
381       int elapsed = int(Brain.Timer.value());
382       int remaining = BREAK_DURATION_SEC - elapsed;
383
384       if (remaining < 0)
385       {
386         remaining = 0;
387       }
388
389       Brain.Screen.clearScreen();
390       Brain.Screen.setCursor(1, 1);
391       Brain.Screen.print("Break: %d sec left", remaining);
392
393       if (remaining == 0)
394       {
395         breakDone = true;
396       }
397
398       wait(100, msec);
399     }
400
401     Brain.Screen.clearScreen();
402     Brain.Screen.setCursor(1,1);
403     Brain.Screen.print("Break Over");
404     wait(3, seconds);
405   }
406
407
408
409   void runStudySession()
410   {
411
412     monitorPhone();
413     phoneJailDoor();
414
415     for(int study_dur = 1500; study_dur > 0; study_dur--)
416     {
417
418       Brain.Screen.clearScreen();
419       Brain.Screen.setCursor(1,1);
420       Brain.Screen.print("Study: %d sec left", study_dur);
421       wait(1, seconds);
422
423       if(Bumper3.pressing())
424       {
425         wait(4,seconds);
426
427         if(!Bumper3.pressing())
428         {
429         Brain.Screen.clearScreen();
430         Brain.Screen.setCursor(1, 1);
431         Brain.Screen.print("Task Complete");
432         wait(3, seconds);
433         Brain.Screen.clearScreen();
434         break;
435         }
436
437         else
438         {
```

```
439             Brain.Screen.clearScreen();


440           Brain.Screen.setCursor(1, 1);
441           Brain.Screen.print(pickQuote().c_str());
442           wait(3, seconds);
443           study_dur = study_dur - 8;
444           if(study_dur < 0)
445           {
446             study_dur = 0;
447           }
448         }
449       }
450     }
451
452     Brain.Screen.clearScreen();
453     Brain.Screen.setCursor(1, 1);
454     Brain.Screen.print("Session Complete");
455     wait(3, seconds);
456   }
457
458   void startPomodoroCycle()
459   {
460     while (true)
461     {
462     phoneJailDoor();
463     runStudySession();
464
465     Brain.Screen.clearScreen();
466     Brain.Screen.setCursor(1, 1);
467     Brain.Screen.print("Press + hold bumper");
468     Brain.Screen.setCursor(2, 1);
469     Brain.Screen.print("if task complete");
470     wait(3, seconds);
471
472     if(Bumper3.pressing())
473     {
474       dispensor(true);
475       searchForPerson();
476       phoneJailDoor();
477       runBreakSession();
478       phoneJailDoor();
479     }
480
481     else
482     {
483       dispensor(false);
484       searchForPerson();
485     }
486
487     Brain.Screen.clearScreen();
488     Brain.Screen.setCursor(1, 1);
489     Brain.Screen.print("End cycles?");
490     wait(2, seconds);
491
492     Brain.Screen.setCursor(2, 1);
493     Brain.Screen.print("Press bumper + touch");
494     wait(2, seconds);
495
496     endBot();
497
498     Brain.Screen.clearScreen();
499     Brain.Screen.setCursor(1, 1);
500     Brain.Screen.print("Starting Study Cycle!");
501     wait(2, seconds);
```

```
502
503     }
504
505   }
506
507   int main()
508   {
509     configureAllSensors();
510     Brain.Screen.clearScreen();
511     Brain.Screen.setCursor(1, 1);
512     Brain.Screen.print("Touch to start");
513     wait(2, seconds);
514     touchPressed();
515     startPomodoroCycle();
516
517   }
518
519
```