```
//A: IMPORT THE LIBRARIES AND DEFINITIONS
//A1: DECLARATIONS FOR CURIEIMU
#include "CurieIMU.h"

int axIn, ayIn, azIn;

double vsumcurrentIn = 0;
double vsumcheckIn = 0;
double vsumnewIn = 0;
//A1: END

//A2: DECLARATIONS FOR EXTERNAL ACCELEROMETER
#include "I2Cdev.h"
#include "MPU6050.h"

#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
#include "Wire.h"
#endif

MPU6050 accelgyro;

int16_t axEx, ayEx, azEx, gxEx, gyEx, gzEx;
double byEx, bxEx, bzEx;

double vsumcurrentEx = 0;
double vsumcheckEx = 0;
double vsumnewEx = 0 ;
//A2: END

//A3: DECLARATIONS FOR SDCARD MODULE
#include <SPI.h>
#include "SdFat.h"

#define USE_SDIO 0 // Set USE_SDIO to zero for SPI card access

const uint8_t SD_CHIP_SELECT = SS; // Default SD Chip select is the SPI SS pin

#if USE_SDIO // Use faster SdioCardEX
SdFatSdioEX sd;
#else // USE_SDIO
SdFat sd;
#endif  // USE_SDIO

float cardSize; // global for card size
File myFile; //will be used for file creation
SdFile file; //will be used for getting filenames
//A3: END

//A4: DECLARATIONS FOR GPS GSM RTC SHIELD
#include <SoftwareSerial.h>

#define DEBUG true
#define GPSready A2

SoftwareSerial mySerial(7, 8);
//A4: END

//A5: OTHER DECLARATIONS
#define DeviceReady A1
#define FallMemory A0

#define falseAlarmButton A3

unsigned long int fallStart;

double degreesdiff = 0;

String lastOrientation = "";
String lastKnownTimeLoc = "";
String userName = "";
//A5: END

//A6: DECLARATIONS FOR KEEPING TRACK OF ACTIVITIES
#define maxAct 1000 //Maximum number of activities per text file
int actFileCountR; //activity file counter
int actCounterR;   //activity counter per file
//A6: END

//A7: DECLARATIONS FOR TIMER
#include "CurieTimerOne.h"
const int gpsTimer = 5000000; //5 seconds
const int memoryTimer = 9000000; //9 seconds
//A7: END
```

```
//A: END

void parseAndsave(char *buff) {
  char *name = strtok(buff, " =");
  if (name) {
    char *junk = strtok(NULL, " ");
    if (junk) {
      char *valu = strtok(NULL, " ");
      if (valu) {
        int val = atoi(valu);
        if (strcmp(name, "actFileCount") == 0) {
          actFileCountR = val;
        }
        if (strcmp(name, "actCounter") == 0) {
          actCounterR = val;
        }
      }
    }
  }
}

void readFileLog() {
  myFile = sd.open("filelog.txt");
  if (myFile) {
    char buffer[5];
    byte index = 0;

    while (myFile.available()) {
      //Serial.write(myFile.read());
      char c = myFile.read();
      if (c == '\n' || c == '\r') { //Check for carriage return or line feed
        parseAndsave(buffer);
        index = 0;
        buffer[index] = '\0'; //Keep buffer NULL terminated
      } else {
        buffer[index++] = c;
        buffer[index] = '\0'; //Keep buffer NULL terminated
      }
    }
    myFile.close();
  } else {
    Serial.println("ERROR READING FILE LOG!");
    readFileLog();
  }

  Serial.print("actFileCount: ");
  Serial.println(actFileCountR);
  Serial.print("actCounter: ");
  Serial.println(actCounterR);
}

void logData(String userOrientation) {
  if (!sd.exists("Activities")) {
    sd.mkdir("Activities");
    sd.chdir("Activities");
  } else {
    sd.chdir("Activities");
  }

  String fileName = "activity";
  fileName.concat(actFileCountR);
  fileName.concat(".txt");

  if (actCounterR < maxAct) {
    myFile = sd.open(fileName, FILE_WRITE);

    if (myFile) {
      myFile.println(lastKnownTimeLoc);
      myFile.println(userOrientation);
      myFile.println("-end_of_activity-");
      myFile.close();
      actCounterR++;

      Serial.println(actCounterR);
      //      Serial.println(sendData("AT+CGNSINF", 1000, DEBUG));
      Serial.println(lastKnownTimeLoc);
      Serial.println("User Orientation: " + userOrientation);
      Serial.println("-end_of_activity-");
    } else {
      Serial.println("Error Writing Activity File");
    }
  } else {
```

```
    actFileCountR++;
    actCounterR = 0;
  }

  if (!sd.chdir()) {
    Serial.println("Error Going back to the root folder");
  }

  updateFileLog();

}

void updateFileLog() {
  String a = "actFileCount = ";
  String b = "actCounter = ";

  a.concat(actFileCountR);
  b.concat(actCounterR);

  removeFile("filelog.txt");
  myFile = sd.open("filelog.txt", FILE_WRITE);
  if (myFile) {
    myFile.println(a);
    myFile.println(b);
    Serial.println(a);
    Serial.println(b);

    myFile.close();
  } else {
    Serial.println("ERROR UPDATING FILE LOG!");
  }
}

void removeFile(String toRemove) {
  myFile = sd.open(toRemove, FILE_WRITE);
  if (myFile) {
    if (!myFile.remove()) {
      Serial.println("ERROR REMOVING OLD FILE LOG!");
    }
    myFile.close();
  } else {
    Serial.println("ERROR OPENING FILE TO BE REMOVED!");
  }
}

void deleteOldFiles() {
  Serial.println("Deleting old files");
  sd.chdir("Activities");
  //sd.ls(LS_R);

  int counter = 0;
  int itemsToDelete = 5;

  char name[20];
  while (file.openNext(sd.vwd(), O_READ) && counter < itemsToDelete) {
    //    file.printName();
    file.getName(name, 20); //20bytes

    removeFile(String(name));

    file.close();
    counter++;
  }

  if (!sd.chdir()) {
    Serial.println("ERROR RETURNING BACK TO THE ROOT");
  }
  delay(2000);
}

void initMainBoard() {
  Serial.println("Initializing IMU device...");
  CurieIMU.begin();
  delay(1000);

  // Set the accelerometer range to 2G
  CurieIMU.setAccelerometerRange(2);
  Serial.println("IMU initialization successful!");
}

void initExtAccel() {
  Serial.println("Initializing External Acceleromter...");
```

```
  // Join I2C bus (I2Cdev library doesn't do this automatically)
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
  Wire.begin();
#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
  Fastwire::setup(400, true);
#endif

  accelgyro.initialize();
  delay(1000);

  //Verify Connection with the external accelerometer
  Serial.println("Testing external accelerometer connection...");
  Serial.println(accelgyro.testConnection() ? "MPU6050 connection successful" : "MPU6050 connection failed");
  accelgyro.setSleepEnabled(false);

  if (accelgyro.testConnection() == 0) {
    Serial.println("Reinitialization started!");
    delay(500);
    Serial.write(12);
    setup();
  }

  Serial.println("External Accelerometer initialization successful!");
}

void initSDCard() {
  Serial.println("Initializing Sd Card...");
#if USE_SDIO
  if (!sd.cardBegin()) {
    Serial.println("cardBegin failed");
    Serial.println("Re-initializing Sd Card...");
    initSDCard();
  }
  if (!sd.begin(SD_CHIP_SELECT)) {
    Serial.println("SD Chip Select initialization failed");
    Serial.println("Re-initializing Sd Card...");
    initSDCard();
  }
#else  // USE_SDIO
  // Initialize at the highest speed supported by the board that is
  // not over 50 MHz. Try a lower speed if SPI errors occur.
  if (!sd.cardBegin(SD_CHIP_SELECT, SD_SCK_MHZ(50))) {
    Serial.println("cardBegin failed");
    Serial.println("Re-initializing Sd Card...");
    initSDCard();
  }
  if (!sd.begin(SD_CHIP_SELECT, SD_SCK_MHZ(50))) {
    Serial.println("SD Chip Select initialization failed");
    Serial.println("Re-initializing Sd Card...");
    initSDCard();
  }
#endif  // USE_SDIO

  cardSize = sd.card()->cardSize();

  if (cardSize == 0) {
    Serial.println("cardSize failed");
    Serial.println("Re-initializing Sd Card...");
    initSDCard();
  }

  if (!sd.fsBegin()) {
    Serial.println("\nFile System initialization failed.\n");
    Serial.println("Re-initializing Sd Card...");
    initSDCard();
  }

  Serial.println("Sd Card initialization successful!");
}

void checkGPSConnection() {
  String response = sendData("AT+CGNSINF", 1000, DEBUG);

  //Check if GPS is already connected/fixed
  if (response[25] == '1') {
    digitalWrite(GPSready, HIGH);
    lastKnownTimeLoc = response;
    Serial.println("GPS READY!");
  } else {
    digitalWrite(GPSready, !digitalRead(GPSready));
    Serial.println("GPS Connecting...");

    //uncomment if not testing
```

```
    //lastKnownTimeLoc = "1,1,20180228034035.000,14.253815,121.056955,84.500,0.43,292.5,1,,0.9,1.2,0.9,,11,9,,,34,,";
  }

  Serial.println(response);
}

void fallBuzz() {
  tone(FallMemory, 400, 200);
  delay(200);
  noTone(FallMemory);
  tone(FallMemory, 500, 200);
  delay(200);
  noTone(FallMemory);
  tone(FallMemory, 600, 200);
  delay(200);
  noTone(FallMemory);
}

void memoryBuzz() {
  tone(FallMemory, 300, 300);
  delay(300);
  noTone(FallMemory);
}

void initGPSModule() {
  Serial.println("Initializing GPS,GSM,RTC Shield...");
  mySerial.begin(38400);

  pinMode(GPSready, OUTPUT);

  onGPS();

  while (lastKnownTimeLoc == "") {
    checkGPSConnection();
  }

  delay(10000);
}

void onGPS() {
  sendData("AT+CGNSPWR=1", 1000, DEBUG);
  Serial.println("GPS Turned ON!");
}

void offGPS() {
  sendData("AT+CGNSPWR=0", 1000, DEBUG);
  Serial.println("GPS Turned OFF!");
}

String sendData(String command, const int timeout, boolean debug) {
  String response = "";
  mySerial.println(command);

  delay(5);

  if (debug) {
    long int time = millis();
    while ( (time + timeout) > millis()) {
      while (mySerial.available()) {
        response += char(mySerial.read());
      }
    }
  }

  return response;
}

void checkSpace() {
  float totalSize = 0.000512 * cardSize;
  float freeSize = 0.000512 * sd.vol()->freeClusterCount() * sd.vol()->blocksPerCluster();

  float lowLevel = 0.1 * totalSize; //10% of total size
  if (freeSize <= lowLevel) {
    memoryBuzz();
    deleteOldFiles();
    Serial.println("LOW MEMORY SPACE!");
    Serial.print("Remaining Space: ");
    Serial.print(freeSize);
    Serial.println(" MB (MB = 1,000,000 bytes)");
  } else {
    Serial.print("Remaining Space: ");
    Serial.println(freeSize);
  }
```

```
}

String setMessage() {
  String message = userName;
  message.concat(" fell! at:  ");
  String date = "";
  String longitude = "";
  String latitude = "";
  int comma = 0;
  int i = 0;

  while (comma <= 5) {
    if (lastKnownTimeLoc[i] == ',' ) {
      comma++;
      i++;
    }

    if (comma == 2) {
      date = date + lastKnownTimeLoc[i];
    }
    if (comma == 3) {
      longitude = longitude + lastKnownTimeLoc[i];
    }
    if (comma == 4) {
      latitude = latitude + lastKnownTimeLoc[i];
    }
    i++;

  }

  String newTime = "";
  String newDate = "";

  for (int j = 0; j < 14; j++) {
    if (j < 8) {
      newDate = newDate + date[j];
      if (j == 3 || j == 5 || j == 5) {
        newDate.concat("-");
      }
    }
    if (j > 7) {
      newTime = newTime + date[j];
    }
  }

  String hh = "";
  for (int j = 0; j < 2; j++){
    hh = hh + newTime[j];
  }

  int timeUTC8 = hh.toInt();
  if(timeUTC8 > 12){
    timeUTC8 = timeUTC8 - 12;
  }

  timeUTC8 = hh.toInt() + 8;
  if(timeUTC8 > 24){
    timeUTC8 = timeUTC8 - 24;
  }

  String UTC8 = String(timeUTC8);
  for(int j = 0; j < 6; j++){
    if(j == 1 || j == 4){
      UTC8.concat(":");
    }
    if(j > 1){
      UTC8 = UTC8 + newTime[j];
    }
  }

  message.concat("\n");
  message.concat("LONGITUDE and LATITUDE: ");
  message.concat("\n");
  message.concat(longitude);
  message.concat(", ");
  message.concat(latitude);
  message.concat("\n");
  message.concat("DATE: ");
  message.concat(newDate);
  message.concat("\n");
  message.concat("TIME: ");
  message.concat(UTC8);
  message.concat(" \r");
```

```
  Serial.println(message);

  return message;
}

void initUsername() {
  myFile = sd.open("profile.txt");
  if (myFile) {
    char buffer[15];
    byte index = 0;
    int lineCount = 0;
    while (myFile.available() && lineCount < 3) {
      char c = myFile.read();
      if (c == '\n' || c == '\r') { //Check for carriage return or line feed
        userName.concat(buffer);
        lineCount++;
        index = 0;
        buffer[index] = '\0'; //Keep buffer NULL terminated
      } else {
        buffer[index++] = c;
        buffer[index] = '\0'; //Keep buffer NULL terminated
      }
    }

    myFile.close();
  } else {
    Serial.println("ERROR READING PROFILE!");
  }
}

void SendTextMessage() {
  offGPS(); //turn off GPS to prevent interruption

  Serial.println("======================");
  Serial.println("===SENDING MESSAGE!===");

  String message = setMessage();

  myFile = sd.open("respondents.txt");
  if (myFile) {
    char buffer[15];
    byte index = 0;

    while (myFile.available()) {
      char c = myFile.read();
      boolean sent = false;
      if (c == '\n' || c == '\r') { //Check for carriage return or line feed
        if (buffer[0] == '+') {
          String toContact(buffer);

          Serial.print("CONTACT NUMBER: ");
          Serial.println(toContact);

          while (sent == false) {
            String response = "";
            String receiver = "AT+CMGS=\"";
            receiver.concat(toContact);
            receiver.concat("\"\r");
            Serial.println(receiver);

            mySerial.print("\r");
            mySerial.print("AT+CMGF=1\r");    //Because we want to send the SMS in text mode
            mySerial.print(receiver);
            delay(1000);
            mySerial.print(message);
            mySerial.write(0x1A);

            long int time = millis();
            while ((time + 1000) > millis()) {
              while (mySerial.available()) {
                response += char(mySerial.read());
              }
            }

            Serial.println("RESPONSE: ");
            Serial.println(response);

            int bracketCount = 0;
            for (int i = 0; i < response.length(); i++) {
              if (response[i] == '>') {
                bracketCount++;
              }
```

```
            }
            if (bracketCount >= 2) {
              sent = true;
              Serial.println("===MESSAGE SENT!===");
            } else {
              Serial.println("===MESSAGE WAS NOT SENT!===");
              Serial.println("Resending Message...");
              sent = false;
            }
          }

        }
        index = 0;
        buffer[index] = '\0'; //Keep buffer NULL terminated
      } else {
        buffer[index++] = c;
        buffer[index] = '\0'; //Keep buffer NULL terminated
      }
    }
    myFile.close();
  } else {
    Serial.println("ERROR READING RESPONDENTS FILE!");
  }

  Serial.println("===DONE!===");
  Serial.println("======================");
  delay(5000);
  onGPS(); //turon on GPS again
}

String getOrientation() {
  byEx = ayEx;

  accelgyro.getMotion6(&axEx, &ayEx, &azEx, &gxEx, &gyEx, &gzEx); //mpu6050
  CurieIMU.readAccelerometer(axIn, ayIn, azIn); // curie

  vsumcurrentEx = sqrt(pow(axEx, 2) + pow(ayEx, 2) + pow(azEx, 2));
  vsumcheckEx = abs((abs(vsumnewEx) - abs(vsumcurrentEx)) / (abs(vsumnewEx))) * 100;

  vsumcurrentIn = sqrt(pow(axIn, 2) + pow(ayIn, 2) + pow(azIn, 2));
  vsumcheckIn = abs((abs(vsumnewIn) - abs(vsumcurrentIn)) / (abs(vsumnewIn))) * 100;

  if (abs(vsumcheckEx) >= 1.5 && abs(vsumcheckIn) >= 1.5) {
    vsumnewEx = vsumcurrentEx;
    vsumnewIn = vsumcurrentIn;
    return dynamicmode();
  } else {
    vsumnewEx = (vsumnewEx + vsumcurrentEx) / 2; //average vector sum
    vsumnewIn = (vsumnewIn + vsumcurrentIn) / 2; //average vector sum
    return staticmode();
  }
}

void setup() {
  Serial.begin(38400);
  delay(1000);
  while (!Serial) {} //wait for serial port to connect

  //Initialize the devices
  initMainBoard();
  initExtAccel();
  initSDCard();
  readFileLog();
  initUsername();
  initGPSModule();

  pinMode(FallMemory, OUTPUT);
  pinMode(DeviceReady, OUTPUT);
  pinMode(falseAlarmButton, INPUT_PULLUP);

  CurieTimerOne.start(memoryTimer, &checkSpace);

  //Prompt a welcome message
  Serial.println("Device is ready!");
  digitalWrite(DeviceReady, HIGH);
  Serial.println("================================");
}

boolean falling = false;

void loop() {
  checkGPSConnection();
```

```
    falling = false;
    String currentOrientation = getOrientation();

    if (currentOrientation != "UNKNOWN") {
      if (currentOrientation != lastOrientation) {
        logData(currentOrientation);

        Serial.println("");
        Serial.print("=====");
        Serial.print(currentOrientation);
        Serial.println("=====");
        Serial.println("");

        lastOrientation = currentOrientation;
      }
    }

    //if falling
  //  falling = true;         //just for testing
  //  fallStart = millis();   //just for testing
    if (falling == true) {
      //check wether to send an alarm in 10 seconds
      boolean flag = true;
      while (flag == true) {
        if (millis() - fallStart <= 10000) {
          if (digitalRead(falseAlarmButton) == LOW) {
            Serial.println("False Alarm!");
            flag = false;
          } else {
            //Serial.println(millis() - fallStart);
            fallBuzz();
          }
        } else {
          SendTextMessage();
          //setMessage();
          flag = false;
        }
      }
    }
  }

  static String staticmode() {

    String staticstr = "UNKNOWN";

    if (axEx <= -13900 && axIn <= -16000 && axIn >= -17500) {
      staticstr = "Standing Position";
    } else if (ayEx >= 14000 && axIn <= -12500) {
      staticstr = "Sitting Position";
    } else if ((abs(azIn) >=  13500 || abs(ayIn) >=  13500) && (abs(azEx) >=  13500 || abs(ayEx) >=  13500)) {
      staticstr = "Lying Position";
    }

    return staticstr;
  }

  static String dynamicmode() {

    String dynastr = "UNKNOWN";
    falling = false;

    degreesdiff = abs(((180 / 3.14) * (acos(ayEx / vsumnewEx))) - ((180 / 3.14) * (acos(byEx / vsumnewEx))));

    //  Serial.print((abs((vsumnew-abs(ayEx))/vsumnew))*9.8);
    //  Serial.println(" m/s^2");
    //

    int fallTreshold = 10;

    if (abs(vsumcheckEx) >= fallTreshold || abs(vsumcheckIn) >= fallTreshold ) {
      if (ayIn >= 13500 && ayEx >= 15900 && (abs(vsumcheckEx) >= fallTreshold || abs(vsumcheckIn) >= fallTreshold )) {
        fallStart = millis();
        dynastr = "Falling! : Backwards";
        falling = true;
      }
      else if (ayIn <= -14000 && ayEx <= -15400 && (abs(vsumcheckEx) >= fallTreshold || abs(vsumcheckIn) >= fallTreshold )) {
        fallStart = millis();
        dynastr = "Falling! : Forward";
        falling = true;
      }
      else if (abs(azIn) >=  15900 && abs(azEx) >= 15400 && (abs(vsumcheckEx) >= fallTreshold || abs(vsumcheckIn) >= fallTreshold )) {
        fallStart = millis();
        dynastr = "Falling! : Sideways";
```

```
        falling = true;
      }

      //  Serial.print((abs(vsumcheck/100))*9.8);
      //  Serial.println(" m/s^2");
    }
    if ((axIn <= -15000 && axIn >= -17500) && degreesdiff <= 55 && degreesdiff >= 10) {
      dynastr = "Walking";
    }
    return dynastr;
  }
```