BILAGA D

Hackathon-lösning

En fullständig, kommenterad referens-sketch för tjuvlarmet.

När du ska titta hit

Den här bilagan innehåller en **fullständig körbar** version av hackathon-koden. Den finns här av två skäl:

1. **Om du fastnar hopplöst under hackathonen** och kursledaren redan har hjälpt tre gånger, så att du inte blockeras av en bug som inte är pedagogiskt värdefull att lösa.
2. **Som referens efter kursen** när du vill bygga vidare eller när du lägger en gammal sketch åt sidan och vill återvända.

Titta inte hit för tidigt. Halva värdet av hackathonen är att brottas med integrationen själv. Att läsa svaret innan du provat är som att titta på baksidan av ett korsord innan du försökt.

Regler för larmet (repeterade)

1. **Knappen** på pin 9 togglar `larmPaslaget` med edge-detection.
2. Om **larmet är på och tilt-sensorn är lutad** → buzzern tjuter och RGB-LED blinkar rött.
3. Om **larmet är av och fotocellen läser under en tröskel** → RGB-LED tänds mjukt som stämningsljus.
4. Annars → allt är tyst och mörkt.

Komplett sketch

```
// ─── Pin-tilldelning ──────────────────────────────────────
const int knappPin  = 9;   // Modul 3 – digital in
```

```
const int tiltPin   = 2;   // Modul 4 – digital in
const int ldrPin    = A0;  // Modul 4 – analog in
const int buzzerPin = 12;  // Modul 3 – digital out
const int ledR      = 6;   // Modul 2 – PWM out
const int ledG      = 5;   // Modul 2 – PWM out
const int ledB      = 3;   // Modul 2 – PWM out

// ─── Tröskel för fotocellen ──────────────────────────
// Kalibrera själv i Serial Monitor. Låg siffra = mörkt.
const int morkTroskel = 300;

// ─── Tillståndsvariabler (globala, lever mellan loopar) ─
bool larmPaslaget = false;
int  lastKnappState = HIGH;  // HIGH = släppt (INPUT_PULLUP)

// ─── Hjälpfunktion för att skriva RGB ────────────────
void sattFarg(int r, int g, int b) {
  analogWrite(ledR, r);
  analogWrite(ledG, g);
  analogWrite(ledB, b);
}

// ─── SETUP ───────────────────────────────────────────
void setup() {
  Serial.begin(9600);
  pinMode(knappPin,  INPUT_PULLUP);
  pinMode(tiltPin,   INPUT_PULLUP);
  pinMode(buzzerPin, OUTPUT);
  pinMode(ledR, OUTPUT);
  pinMode(ledG, OUTPUT);
  pinMode(ledB, OUTPUT);

  Serial.println("Larmet startat. Larm av som default.");
}

// ─── LOOP ────────────────────────────────────────────
void loop() {
  // 1. LÄS INPUTS
  int  knappState = digitalRead(knappPin);
  bool tiltLutad  = (digitalRead(tiltPin) == LOW);
  delay(50);  // debounce tilt – kulan bouncar i hylsan
  int  ljus       = analogRead(ldrPin);

  // 2. EDGE-DETECTION för knappen (toggla larmläget)
  if (knappState == LOW && lastKnappState == HIGH) {
    larmPaslaget = !larmPaslaget;
    Serial.print("Larm nu ");
    Serial.println(larmPaslaget ? "PÅ" : "AV");
  }
  lastKnappState = knappState;

  // 3. BESTÄM VAD SOM SKA HÄNDA
  if (larmPaslaget && tiltLutad) {
```

```
    // LARM AKTIVT + RÖRT → tjut + rött blink
    digitalWrite(buzzerPin, HIGH);
    sattFarg(255, 0, 0);
    delay(100);
    digitalWrite(buzzerPin, LOW);
    sattFarg(0, 0, 0);
    delay(100);
  } else if (!larmPaslaget && ljus < morkTroskel) {
    // LARM AV + MÖRKT → stämningsljus (mjukt varmt)
    digitalWrite(buzzerPin, LOW);
    sattFarg(120, 60, 20);
  } else {
    // TYST OCH MÖRKT
    digitalWrite(buzzerPin, LOW);
    sattFarg(0, 0, 0);
  }

  // 4. Kort paus mot kortkrets-looping
  delay(10);
}
```

Rad-för-rad-förklaring

PIN-TILLDELNING (RAD 2–8)

Alla pin-nummer samlade överst i filen. `const int` så de inte kan råka ändras. Den här bara-deklarations-stilen är standard bland Arduino-sketchar — allt läsbart från toppen, inga magiska siffror gömda i mitten av koden.

`MORKTROSKEL` (RAD 12)

`300` är en gissning. Du ska kalibrera själv med Serial Monitor. Lämna värdet som en `const int` i toppen så du enkelt kan ändra det utan att leta genom hela filen.

`LASTKNAPPSTATE` (RAD 16)

Startas på `HIGH` eftersom `INPUT_PULLUP` gör att en **släppt** knapp läses som HIGH. Om du startar på `LOW` kan edge-detection tro att knappen var tryckt första loopen och toggla larmet direkt.

> **NAMING-SKIFTE MOT MODUL 3**
>
> I Modul 3:s slides och ”Bygg från minnet” hette variablerna `state` och `lastState`. Här i hackathon-koden står det `knappState` och `lastKnappState`. Anledningen är att Bilaga D läser flera tillstånd (knapp, tilt, ljus) i samma loop — då är `state` för otydligt. När du själv skriver hackathon-koden kan du behålla `state` / `lastState` om du bara har en knapp; namnen är bara etiketter, mönstret är detsamma.

`sattFarg()` HJÄLPFUNKTION (RAD 19–23)

Att skriva `analogWrite(ledR, ...)` + `analogWrite(ledG, ...)` + `analogWrite(ledB, ...)` på varje rad blir tröttsamt. En funktion som tar tre tal och skriver dem i en svep är både kortare och lättare att läsa. `sattFarg(255, 0, 0)` betyder ”helt rött”.

Det är det första ”egenskrivna” verktyget i koden. Du kunde klarat dig utan, men det **gör koden läsbar**. Det är i princip vad programmering i större skala handlar om — att skapa små, tydliga verb för vad du vill göra.

EDGE-DETECTION (RAD 46–51)

Samma mönster som i Modul 3: **agera bara när knappen just nu övergår från HIGH till LOW**. Utan detta skulle larmet toggla 100+ gånger per tryckning, och värdet skulle vara slumpmässigt vid släpp.

`larmPaslaget ? "PÅ" : "AV"` är en kort ternär if — ”om larmPaslaget, använd ’PÅ’, annars ’AV’”. En rad istället för fyra.

TRE GRENAR AV IF/ELSE (RAD 55–73)

Den första grenen har en liten detalj värd att notera: den alternerar mellan rött och svart med `delay(100)` — det är hur blink skapas inne i ett `loop`-varv. Varje gång loopen kommer tillbaka till den här grenen, kör det ett fullt blink-cykel på 200 ms.

Det betyder att knapptrycket inte reagerar förrän **efter** en blink-cykel. 200 ms är knappt märkbart för en människa, men om du vill att larmet ska svara snabbare på knappen kan du göra blinkningen med `millis()` istället för `delay()`. Det är en ren utökning — ingår inte i kursen men finns i alla online-tutorials under sökordet ”non-blocking Arduino”.

`delay(10)` SIST (RAD 77)

En liten paus mellan loop-varv. Gör att Arduinon inte snurrar CPU:n på tomgång. Tio millisekunder är kort nog att interaktionen känns omedelbar men tillräckligt för att knappens studs lägger sig innan nästa läsning.

Varianter att prova

VARIATION 1 — DUBBELT PIP VID LARM PÅ/AV

Istället för att bara ändra `larmPaslaget` tyst, låt buzzern säga till med ett kort pip:

```
if (knappState == LOW && lastKnappState == HIGH) {
  larmPaslaget = !larmPaslaget;

  // Pip-bekräftelse
  digitalWrite(buzzerPin, HIGH);
  delay(50);
  digitalWrite(buzzerPin, LOW);
  if (!larmPaslaget) {
    delay(50);
```

```
    digitalWrite(buzzerPin, HIGH);
    delay(50);
    digitalWrite(buzzerPin, LOW);
  }
}
```

Ett pip = larm på, två pip = larm av. Bekräftar aktiveringen utan att säga något.

VARIATION 2 — HYSTERES PÅ MÖRKERTRÖSKELN

Ett problem: om `ljus` ligger precis på `morkTroskel = 300`, kan värdet darra mellan 299 och 301 och lampan blinkar av och på. Lösningen är två olika trösklar — en för att tända, en för att släcka:

```
const int tandVid = 280;
const int slackVid = 320;

// Statisk variabel som behåller sitt värde mellan loopar
static bool stamningslampaPa = false;

if (ljus < tandVid) {
  stamningslampaPa = true;
} else if (ljus > slackVid) {
  stamningslampaPa = false;
}
// (lampan behåller sitt senaste tillstånd när ljus är mellan trösklarna)

if (stamningslampaPa) {
  sattFarg(120, 60, 20);
} else {
  sattFarg(0, 0, 0);
}
```

`static` betyder att variabeln behåller sitt värde mellan loop-varv trots att den är lokal. Det är ett alternativ till global `bool stamningslampaPa;` längst upp.

Hysteres är konceptet att två trösklar (tänd, släck) ligger olika, så att systemet inte oscillerar på gränsen.

VARIATION 3 — FADE-IN PÅ STÄMNINGSLJUS

Istället för att LED-en bara **tänds** när det blir mörkt, låt den långsamt tona upp. Detta kräver en lokal `int`-variabel som minns nuvarande ljusstyrka mellan loop-varv:

```
static int stamningsStyrka = 0;

if (!larmPaslaget && ljus < morkTroskel) {
  if (stamningsStyrka < 120) stamningsStyrka++;
} else {
  if (stamningsStyrka > 0) stamningsStyrka--;
}
```

```
  analogWrite(ledR, stamningsStyrka);
  analogWrite(ledG, stamningsStyrka / 2);
  analogWrite(ledB, stamningsStyrka / 6);
```

Varje loop-varv justerar styrkan med 1. Över 120 loop-varv (ca 1,2 sekunder vid `delay(10)`) tonar LED:en från 0 till 120. Mjukare beteende, inget hopp.

Sista rådet

Om du använder den här koden rakt av: **skriv om den för hand**. Inte för att den här versionen är dålig, utan för att du lär dig genom att skriva. En Arduino-bok med 500 sidor är inte lika mycket värd som 200 rader kod du skrivit själv från minnet.

Och om något här känns onödigt komplicerat — **skit i det**. En fungerande enkel lösning är värt hundra eleganta som inte kompilerar.