

Designnotat

Tittel: Designprosjekt 6 - Reaksjonstrener	
Forfattere: Temmam Nabil Altaie	
Versjon: 1.0	Dato: 28.09.25

Innhold

1 Problembeskrivelse **2**

2 Prinsipiell løsning **3**

2.1 Overordnet systemidé 3

2.2 Spilllogikk og flyt 4

2.3 Sanntidsaspekt og oppgavefordeling 5

2.4 Designhensyn 6

3 Realisering og test **7**

3.1 Maskinvare og kobling 7

3.2 Programvareimplementasjon 8

3.3 Testing 11

3.4 Resultater og vurdering 11

4 Konklusjon **12**

5 Takk **12**

Referanseliste **13**

Vedlegg **14**

A main.cpp 14

1 Problembeskrivelse

Oppstartsfirmaet *Sproing* har fått finansiering til å utvikle en reaksjonstrener kalt *Psycho!*. Produktet skal bestå av en boks med fire lysdioder og tilhørende knapper. Spillet går ut på at lysdioder tennes i tilfeldige kombinasjoner, og brukeren må reagere raskt ved å trykke på de riktige knappene.

Hovedutfordringen er å skape et system som tester og trener brukerens reaksjonsevne. Brukeren starter med et gitt antall liv N, og mister liv dersom lysene ikke slukkes innen en tidsgrense T, eller dersom feil knapp trykkes. Spillet skal gradvis øke i vanskelighetsgrad, både ved at flere lys kan tennes samtidig og ved at tidsgrensen T blir kortere. Når brukeren har mistet alle liv, er spillet over, og resultatene skal vises til spilleren.

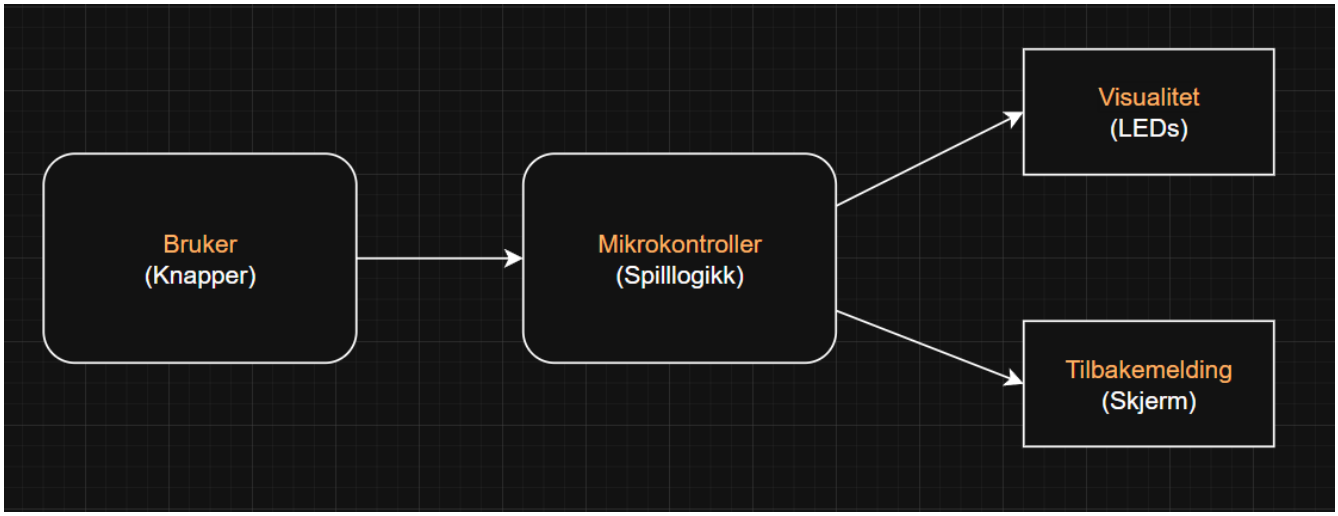
Problemstillingen kan oppsummeres slik:

- Hvordan designe et system som på en pålitelig måte kan teste brukerens reaksjonsevne gjennom lys- og knappestyring?
- Hvordan utforme spillets regler slik at vanskelighetsgraden øker naturlig over tid?
- Hvordan sikre at prototypen kan demonstreres for investorer innen en kort tidsfrist?

2 Prinsipiell løsning

I denne delen beskrives hvordan reaksjonstreneren kan realiseres på et prinsipielt nivå. Målet er å vise hvordan systemet kan bygges opp teoretisk, uavhengig av spesifikke komponentverdier eller implementasjonsdetaljer.

2.1 Overordnet systemidé



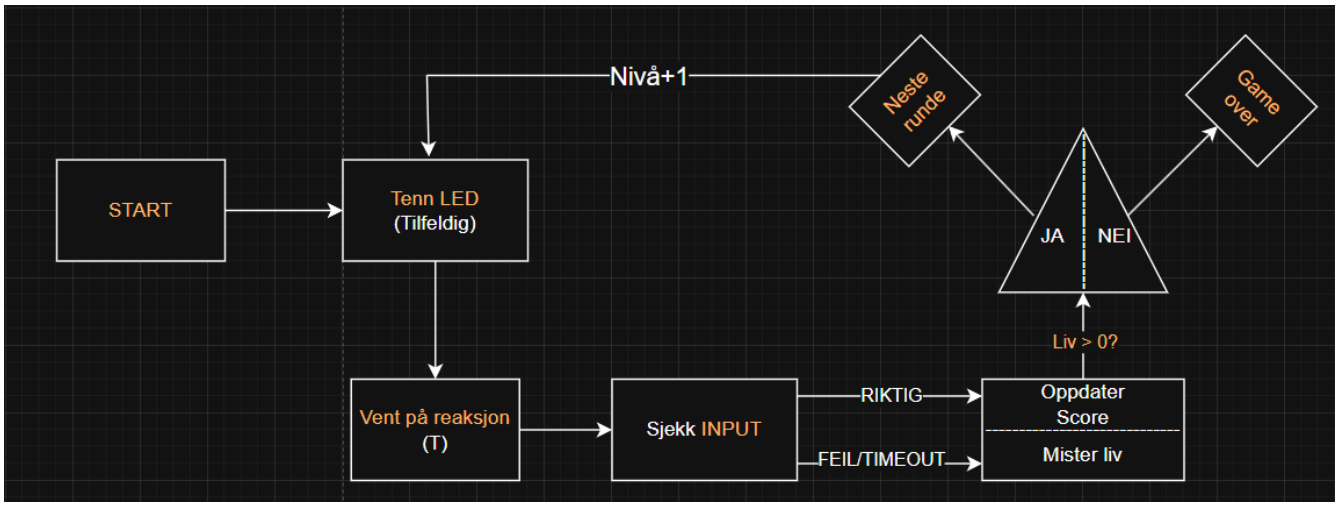
Figur 1: Abstrakt blokkskjema som viser hovedkomponentene i systemet: bruker (knapper), mikrokontroller (spilllogikk), LEDs og tilbakemelding (skjerm).

Systemet kan deles inn i fire hoveddeler:

- **Bruker** – interagerer med systemet gjennom knapper.
- **Mikrokontroller** – kjører spilllogikken og styrer informasjonsflyten.
- **Visualitet (LEDs)** – gir visuelle signaler til brukeren.
- **Tilbakemelding (skjerm)** – viser status, score og game over.

Systemets oppbygning er vist i figur 1. Her ser vi hvordan brukeren gir input via knapper, mikrokontrolleren styrer logikken, og utdata gis gjennom LEDs og tilbakemelding på skjerm.

2.2 Spilllogikk og flyt



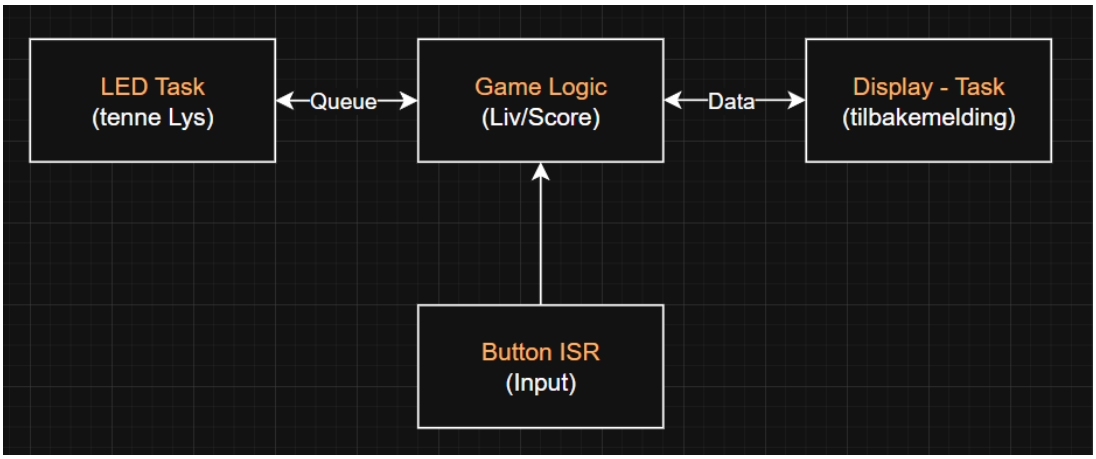
Figur 2: Prinsipielt spillflytskjema. Viser logikken fra start, reaksjon, evaluering og beslutning om neste runde eller game over.

En spillsyklus består prinsipielt av følgende trinn:

1. Systemet starter en ny runde ved å tenne en tilfeldig valgt LED.
2. Brukeren får et tidsvindu T til å reagere ved å trykke riktig knapp.
3. Systemet evaluerer input:
   - Riktig knapp innen tidsfristen $\rightarrow$ score økes.
   - Feil knapp eller timeout $\rightarrow$ et liv trekkes fra.
4. Når liv > 0, fortsetter spillet i neste runde med økt vanskelighetsgrad.
5. Når liv $= 0$, avsluttes spillet og resultatet vises.

Den prinsipielle flyten i spillet er illustrert i figur 2. Flytskjemaet viser hvordan en runde starter, hvordan input evalueres, og hvordan spillet enten fortsetter eller avsluttes.

2.3 Sanntidsaspekt og oppgavefordeling



Figur 3: Task-oversikt. Viser parallellkjøring av oppgaver: LED-styring, knappelesing via avbrudd, spill-logikk og tilbakemelding til skjerm.

For å sikre korrekt reaksjon og samtidig håndtere flere prosesser parallelt, kan det benyttes et sanntidsoperativsystem. Funksjonaliteten fordeles på:

- **LED-task** – styrer tenning og slukking av lys.
- **Button-ISR** – registrerer knappetrykk og sender meldinger.
- **Game logic-task** – administrerer regler, score og liv.
- **Display-task** – viser status og tilbakemelding på skjerm.

For å håndtere samtidighet fordeles systemet på flere oppgaver, vist i figur 3. Diagrammet viser hvordan LED-styring, knapperegistrering, spilllogikk og tilbakemelding håndteres parallelt.

Prinsipielle formler

For å beskrive spillflyten kan noen generelle uttrykk innføres:

- **Reaksjonstid:**

$$t_{\text{reaksjon}} \leq T \tag{1}$$

  Spilleren må reagere innen tidsvinduet T for å beholde liv.

- **Vanskelighetsøkning:**

$$T_{n+1} = T_n - \Delta T \tag{2}$$

  Tidsvinduet reduseres gradvis per runde. Alternativt kan vanskelighetsgraden økes ved å la flere lys tennes samtidig.

- **Liv:**

$$N_{n+1} = \begin{cases} N_n - 1, & \text{hvis feil knapp eller timeout} \\ N_n, & \text{hvis korrekt reaksjon} \end{cases} \tag{3}$$

- **Score:**

$$S_{n+1} = S_n + 1 \quad \text{hvis korrekt reaksjon innen } T \tag{4}$$

Disse formlene uttrykker hvordan spillets tilstand utvikles over tid, uavhengig av implementasjonsdetaljer.

2.4 Designhensyn

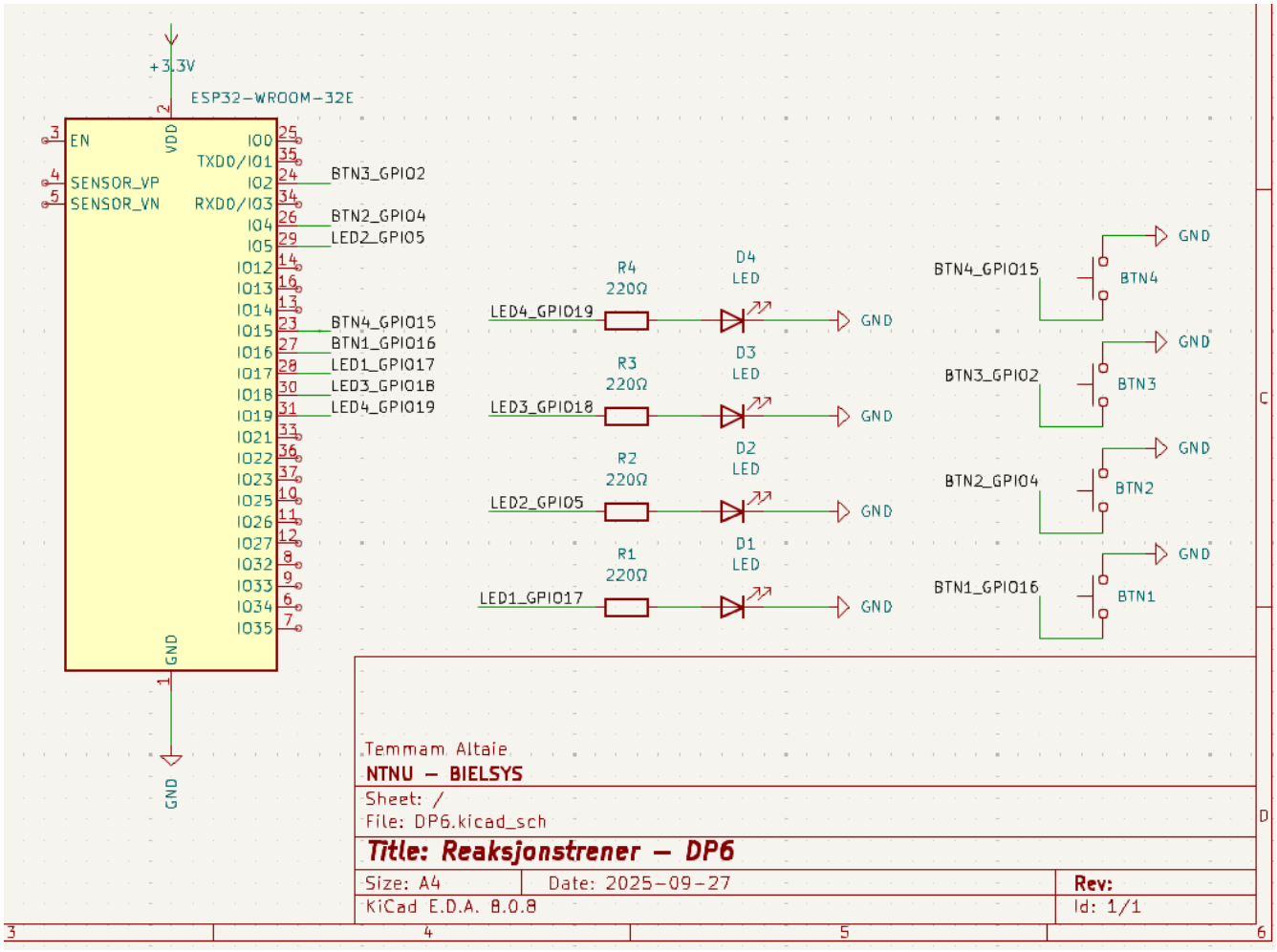
Et prinsippielt design for en reaksjonstrener må ta hensyn til følgende faktorer:

- **Tidskritikalitet** – systemet må reagere innen et gitt tidsvindu (T).
- **Samtidighet** – flere prosesser (LED-styring, knappelesing, spilllogikk) må kunne kjøres parallelt.
- **Skalerbarhet** – designet skal kunne utvides med flere LEDs og knapper uten store endringer.
- **Robusthet** – systemet må tåle feilinput og samtidige hendelser uten å miste konsistens.

3 Realisering og test

I denne delen beskrives hvordan systemet ble implementert i praksis og hvilke tester som ble gjennomført for å verifisere funksjonaliteten. Her presenteres maskinvare, programvare, testoppsett og resultater.

3.1 Maskinvare og kobling

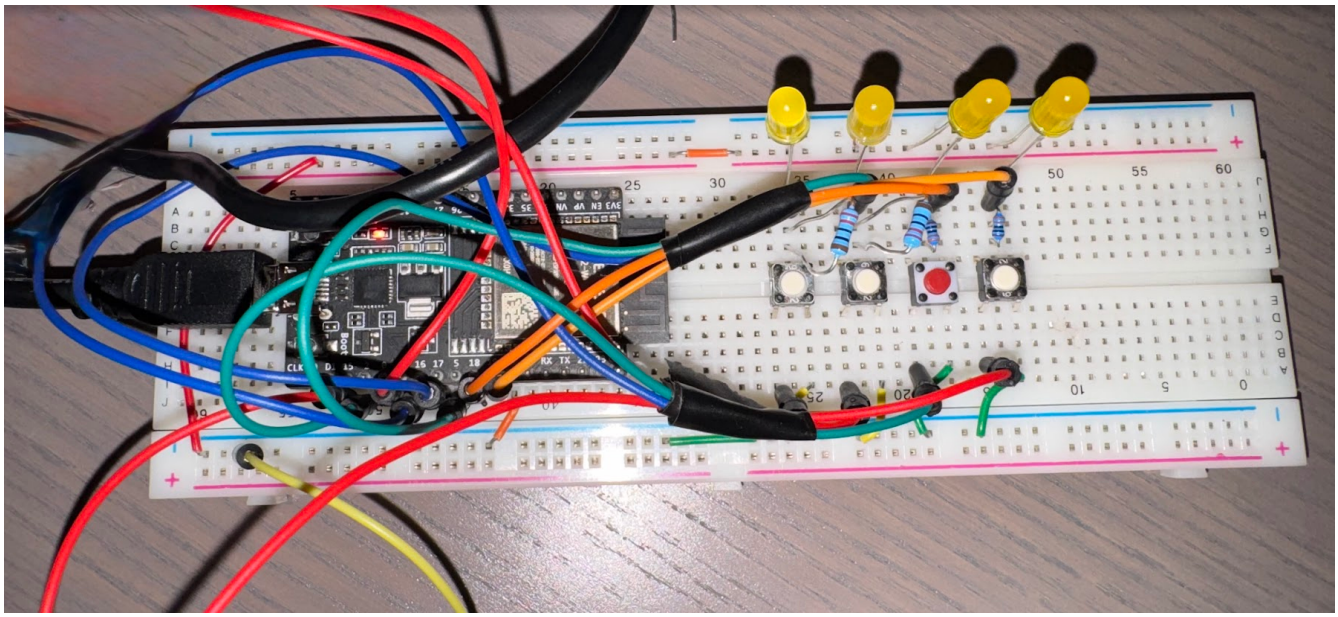


Figur 4: Endelig koblingsskjema for reaksjonstreneren.

Systemet ble realisert på en ESP32 Wroom32E, databladet til ESP [3] ble brukt for å sjekke hvilke I/O pinner som passer til formålet. De sentrale komponentene var:

- Fire trykknapper (BTN1–BTN4) koblet til `GPIO`-pinner.
- Fire lysdioder (LED1–LED4) for visualitet.
- Fire motstander $R_1 - R_4$ med verdi 220Ω.
- En seriell monitor på PC for visning av resultat.
- Felles ground for ESP32, LEDs og knapper.

Oppkoblingen er vist i figur 4, og implementasjonen ble bygget slik at systemet kan utvides med flere knapper og LEDs.

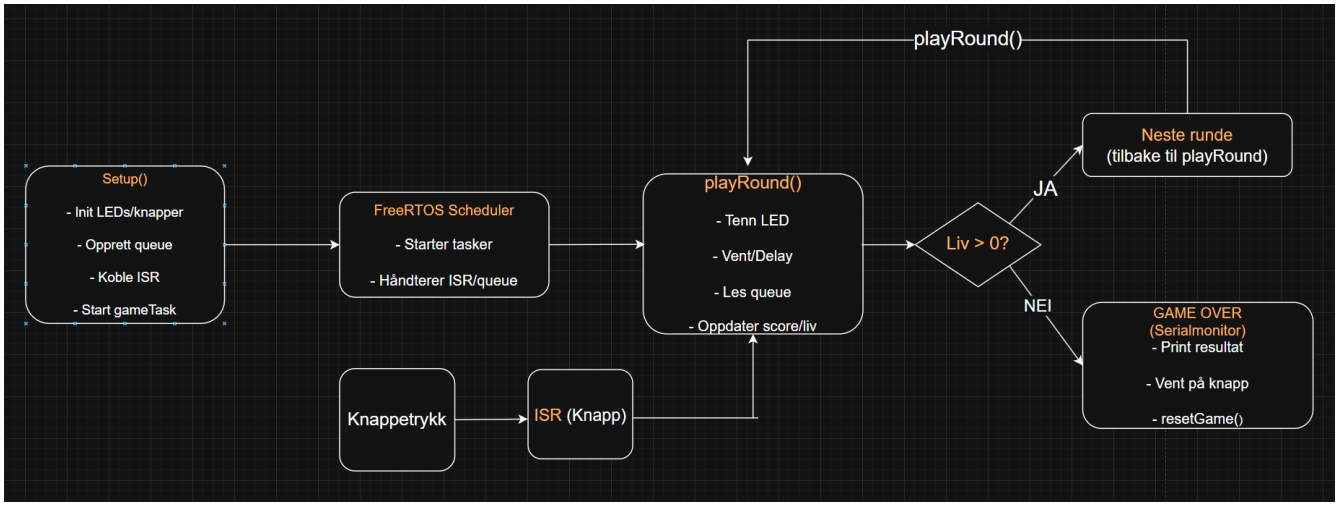


Figur 5: Fysisk kobling av reaksjonstreneren realisert ved hjelp av ESP32.

I figur 5 ser vi den fysiske koblingen realisert på samme måte som i figur 4. For å oppnå et ryddig oppsett ble det brukt elektrikerteip for å hindre at ledningene er i veien for knappene.

3.2 Programvareimplementasjon

Koden ble strukturert i flere oppgaver (tasks) i FreeRTOS, med en egen interrupt service routine (ISR) for å fange knappetrykk. Disse samsvarer med prinsippene beskrevet i kapittel 2 og det ble tatt utgangspunkt fra [1] og [4].



Figur 6: Flytskjema over RTOS-struktur og implementasjon.

Knappetrykk og kø

Knappetrykk håndteres i en ISR, som ikke bearbeider logikken direkte, men sender hendelsen videre til en kø. Dette er viktig for å unngå at avbruddet blokkerer systemet, og viser hvordan FreeRTOS, med utgangspunkt i brukes til synkronisering mellom ISR og oppgaver.

```
void IRAM_ATTR buttonISR(void* arg) {
  int id = (int)arg;
  BaseType_t xHigherPriorityTaskWoken = pdFALSE;
  xQueueSendFromISR(buttonQueue, &id, &xHigherPriorityTaskWoken);
  if (xHigherPriorityTaskWoken) portYIELD_FROM_ISR();
}
```

Kodeutdrag 1: ISR for knappetrykk

ISR-en sender knapp-ID til `buttonQueue`, som senere hentes av spill-logikken i `playRound`.

Spillflyt: funksjonen `playRound`

Kjernen i systemets spilllogikk er implementert i funksjonen `playRound`, som tilsvarer flyten vist i figur 2 og deler av figur 6. Funksjonen styrer en hel runde av spillet, fra LED-tenning til evaluering av resultat.

```
void playRound(GameState &s) {
  // Tenn tilfeldig LED basert paa level
  int numLEDs = (s.level == 1 ? 1 : (s.level == 2 ? 2 : 3));
  int chosen[3];
  for (int i = 0; i < numLEDs; i++) {
    chosen[i] = random(0, 4);
  }
  setLEDs(chosen, numLEDs, true);

  // Vent paa knapp eller timeout
  int pressed;
  TickType_t startTick = xTaskGetTickCount();
  bool success = false;

  while (xTaskGetTickCount() - startTick < pdMS_TO_TICKS(s.reactionT)) {
    if (xQueueReceive(buttonQueue, &pressed, 0)) {
      for (int i = 0; i < numLEDs; i++) {
        if (pressed == chosen[i]) success = true;
      }
      break;
    }
    vTaskDelay(pdMS_TO_TICKS(10));
  }
  setLEDs(chosen, numLEDs, false);

  // Oppdater liv/score
  if (success) { s.score++; }
  else { s.lives--; }
}
```

Kodeutdrag 2: Kjerne i spillflyten (playRound)

Denne implementasjonen kobler direkte til de prinsipielle formlene:

- Reaksjonstid evalueres mot ligning 1.

- Score oppdateres i henhold til ligning 4.
- Liv reduseres etter ligning 3.

Vanskelighetsgrad

Vanskelighetsgrad økes gradvis ved at flere LEDs tennes samtidig og reaksjonstiden forkortes. Dette implementeres med et enkelt nivåsystem.

```
if (s.score % LEVEL_UP_EVERY == 0 && s.level < MAX_LEVEL) {
  s.level++;
  if (s.reactionT > MIN_REACTION_TIME + 200) {
    s.reactionT -= 500; // kortere tid
  }
}
```

Kodeutdrag 3: Oppgradering av vanskelighetsgrad

Dette svarer til prinsippet uttrykt i ligning 2, og sikrer at spillet blir gradvis mer utfordrende.

Game over og restart

Når spilleren mister alle liv, avsluttes spillet. Resultatet skrives til seriellmonitor, og systemet venter på nytt knappetrykk for å starte en ny runde.

```
if (state.lives <= 0) {
  Serial.println("\n=== GAME OVER ===");
  Serial.print("Endelig score: "); Serial.println(state.score);
  Serial.println("Trykk en knapp for    starte p   nytt...");

  int dummy;
  xQueueReceive(buttonQueue, &dummy, portMAX_DELAY);
  resetGame(state);
}
```

Kodeutdrag 4: Game over-håndtering

Setup

I `setup()` initialiseres alle nødvendige komponenter slik at systemet er klart til bruk. Funksjonen konfigurerer seriell kommunikasjon, setter opp I/O-pinner for LEDs og knapper, oppretter køen for kommunikasjon mellom ISR og oppgaver, og knytter avbrudd til knappene. Deretter initialiseres spilltilstanden, og hovedoppgaven `gameTask` startes under FreeRTOS.

```
// --- SETUP ---
void setup() {
  Serial.begin(115200);
  delay(500);
  Serial.println("Starter Reaksjonstrener...");

  randomSeed(analogRead(0));

  // LEDs
  for (int i = 0; i < 4; i++) {
    pinMode(LED_PINS[i], OUTPUT);
```

```
    digitalWrite(LED_PINS[i], LOW);
  }

  // Knapper
  for (int i = 0; i < 4; i++) {
    pinMode(BTN_PINS[i], INPUT_PULLUP);
  }

  // Queue
  buttonQueue = xQueueCreate(10, sizeof(int));

  // Interrupts
  for (int i = 0; i < 4; i++) {
    attachInterruptArg(BTN_PINS[i], buttonISR, (void*)i, FALLING);
  }

  // Init game state
  resetGame(state);

  // Start FreeRTOS task
  xTaskCreate(gameTask, "Game", 4096, NULL, 1, NULL);
}
```

Kodeutdrag 5: Setup funksjon

Dette tilfredsstiller kravspesifikasjonen om at spilleren skal få mulighet til å prøve igjen etter et tapt spill.

3.3 Testing

Etter implementasjonen ble systemet testet i ulike scenarier for å verifisere at funksjonaliteten samsvarer med kravspesifikasjonen:

- **Riktig knappetrykk** – brukeren trykket riktig knapp innen tidsvinduet T.
- **Feil knappetrykk** – brukeren trykket en annen knapp enn den som var tent.
- **Timeout** – brukeren trykket ingen knapp innen T.
- **Flere runder** – systemet økte vanskelighetsgraden gradvis.

Tabell 1: Resultater fra funksjonstester

Scenario	Forventet resultat	Faktisk resultat	OK/Feil
Riktig knappetrykk	Score økes	Score økes	OK
Feil knappetrykk	Liv reduseres	Liv reduseres	OK
Timeout	Liv reduseres	Liv reduseres	OK
Økt vanskelighet	Kortere T / flere LEDs	Kortere T / flere LEDs	OK

3.4 Resultater og vurdering

Systemet fungerte som forventet og oppfylte kravene til minimum tre vanskelighetsgrader. Alle testscenarier stemte med forventede resultater (tabell 1). Prototypen demonstrerer dermed en fungerende reaksjonstrener. Videre forbedringer kan inkludere mer avansert skjerm for tilbakemelding.

4 Konklusjon

Prosjektets mål var å utvikle en prototyp av en reaksjonstrener som tester brukerens reaksjonsevne gjennom stimuli, tidsavhengige responser og økende vanskelighetsgrad.

Løsningen ble realisert på en ESP32 med FreeRTOS, der oppgaver for LED-styring, knappetrykk, spilllogikk og tilbakemelding ble implementert. Testing viste at systemet oppførte seg i tråd med spesifikasjonen: riktige knappetrykk ga poeng, feil eller manglende respons reduserte liv, og vanskelighetsgraden økte gradvis med progresjon. Alle kravene ble dermed oppfylt.

Konklusjon: Prototypen fungerer som en reaksjonstrener i tråd med problemstillingen og demonstrerer et skalerbart og modulært design. Systemet gir et solid utgangspunkt for videreutvikling, for eksempel med OLED-skjerm.

5 Takk

Takk til Mathias Ekse som har bidratt til et godt samarbeid for å løse dette designprosjektet. I tillegg har jeg brukt ChatGPT for forklaringer av begreper, tilbakemeldinger og forbedringspotensialer i teksten min.

Referanseliste

[1] FreeRTOS, *FreeRTOS Documentation Overview*, tilgjengelig på: `https://freertos.org/Documentation/00-Overview`, lest 28. september 2025.

[2] Institutt for elektroniske systemer, NTNU, *Designprosjekt 6: Reaksjonstrener*, 2025. Vedlegg gitt i emnet IELS2022.

[3] Espressif Systems, *ESP32-WROOM-32E / ESP32-WROOM-32UE Datasheet*, 2021. Tilgjengelig på: `https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf`, lest 28. september 2025.

[4] Shawn Hymel, Digi-Key Electronics, *Introduction to RTOS*, YouTube-serie. Tilgjengelig på: `https://www.youtube.com/playlist?list=PLEBQazB0HUyQ4hAPU1cJED6t3DU0h34bz`, lest 28. september 2025.

Vedlegg

A main.cpp

I kodevedlegg 6 ser vi hvordan systemet ble realisert ved å benytte oss av FreeRTOS.

```
#include <Arduino.h>

// --- PIN DEFINISJONER ---
constexpr int LED_PINS[4] = {17, 5, 18, 19};
constexpr int BTN_PINS[4] = {16, 4, 2, 15};

// --- KONSTANTER ---
constexpr int INITIAL_LIVES = 3;
constexpr unsigned long INITIAL_REACTION_TIME = 2500; // ms
constexpr unsigned long MIN_REACTION_TIME = 1000;      // ms
constexpr int LEVEL_UP_EVERY = 5;  // antall riktige for level-up
constexpr int MAX_LEVEL = 3;

// --- STRUKTUR FOR GAMESTATE ---
struct GameState {
  int lives;
  int score;
  int level;
  unsigned long reactionT;
} state;

// FreeRTOS
QueueHandle_t buttonQueue;

// --- HJELPEFUNKSJONER ---
void allLEDsOff() {
  for (int i = 0; i < 4; i++) {
    digitalWrite(LED_PINS[i], LOW);
  }
}

void setLEDs(const int* indices, int count, bool state) {
  for (int i = 0; i < count; i++) {
    digitalWrite(LED_PINS[indices[i]], state ? HIGH : LOW);
  }
}

void resetGame(GameState &s) {
  s.lives = INITIAL_LIVES;
  s.score = 0;
  s.level = 1;
  s.reactionT = INITIAL_REACTION_TIME;
  Serial.println("\n=== Nytt spill starter! ===\n");
}

// --- ISR FOR KNAPPER ---
void IRAM_ATTR buttonISR(void* arg) {
  int id = (int)arg;
  BaseType_t xHigherPriorityTaskWoken = pdFALSE;
  xQueueSendFromISR(buttonQueue, &id, &xHigherPriorityTaskWoken);
  if (xHigherPriorityTaskWoken) portYIELD_FROM_ISR();
}

// --- GameLogikk ---
void playRound(GameState &s) {
```

```
56      //START RUNDEN:
57    allLEDsOff();
58
59    // Velg LED(er) basert paa level
60    int numLEDs = (s.level == 1 ? 1 : (s.level == 2 ? 2 : 3));
61    int chosen[3];
62    for (int i = 0; i < numLEDs; i++) {
63      chosen[i] = random(0, 4);
64    }
65
66    // Tenn LED(s) basert paa Random valg
67    setLEDs(chosen, numLEDs, true);
68    Serial.print("LED tent: ");
69    for (int i = 0; i < numLEDs; i++) { Serial.print(chosen[i]); Serial.print(" ")
        ; }
70    Serial.println();
71
72    // Vent paa knapp eller timeout
73    int pressed;
74    TickType_t startTick = xTaskGetTickCount();
75    bool success = false;
76
77    while (xTaskGetTickCount() - startTick < pdMS_TO_TICKS(s.reactionT)) {
78      if (xQueueReceive(buttonQueue, &pressed, 0)) {
79        Serial.print("Knapp trykket: "); Serial.println(pressed);
80
81        for (int i = 0; i < numLEDs; i++) {
82          if (pressed == chosen[i]) {
83            success = true;
84            break;
85          }
86        }
87        break; // forste trykk teller
88      }
89      vTaskDelay(pdMS_TO_TICKS(10));
90    }
91
92    // Slukk LEDs
93    setLEDs(chosen, numLEDs, false);
94
95    // Vurder resultat
96    if (success) {
97      s.score++;
98      Serial.print("Riktig! Score: "); Serial.println(s.score);
99
100     // Oppgrader vanskelighetsgrad
101     if (s.score % LEVEL_UP_EVERY == 0 && s.level < MAX_LEVEL) {
102       s.level++;
103       if (s.reactionT > MIN_REACTION_TIME + 200) {
104         s.reactionT -= 500;
105       }
106       Serial.print("LEVEL UP! Nytt level: "); Serial.print(s.level);
107       Serial.print(" | Reaksjonstid: "); Serial.println(s.reactionT);
108     }
109   } else {
110     s.lives--;
111     Serial.print("Feil eller timeout! Liv igjen: "); Serial.println(s.lives);
112   }
113
114   vTaskDelay(pdMS_TO_TICKS(500)); // kort pause
115 }
116
```

```
// --- TASK: LOGIKK ---
void gameTask(void* pv) {
  while (true) {
    if (state.lives <= 0) {
      Serial.println("\n=== GAME OVER ===");
      Serial.print("Endelig score: "); Serial.println(state.score);
      Serial.println("Trykk en knapp for aa starte paa nytt...");

      // Vent paa knapp
      int dummy;
      xQueueReceive(buttonQueue, &dummy, portMAX_DELAY);
      resetGame(state);
    }
    playRound(state);
  }
}

// --- SETUP ---
void setup() {
  Serial.begin(115200);
  delay(500);
  Serial.println("Starter Reaksjonstrener...");

  randomSeed(analogRead(0));

  // LEDs
  for (int i = 0; i < 4; i++) {
    pinMode(LED_PINS[i], OUTPUT);
    digitalWrite(LED_PINS[i], LOW);
  }

  // Knapper
  for (int i = 0; i < 4; i++) {
    pinMode(BTN_PINS[i], INPUT_PULLUP);
  }

  // Queue
  buttonQueue = xQueueCreate(10, sizeof(int));

  // Interrupts
  for (int i = 0; i < 4; i++) {
    attachInterruptArg(BTN_PINS[i], buttonISR, (void*)i, FALLING);
  }

  // Init game state
  resetGame(state);

  // Start FreeRTOS task
  xTaskCreate(gameTask, "Game", 4096, NULL, 1, NULL);
}

void loop() {
  // FreeRTOS haandterer alt
}
```

Kodeutdrag 6: Main fil for programflyt på ESP32