



Bits by DAJW - Rapport

av Daniel Johansson, Ahmed Alhasani, Jonathan Koitsalu, William Hansen.
2021-11-24

Introduktion

"Bits by DAJW" är inspirerat av Nokia:s snake. En orm som styrs med mekaniska lokala knappar för att äta "äpplen" som gör ormen större. Score skickas upp till en hemsida online. Top10 syns även lokalt i spelet. Hemsidan kommer med en interaktiv komponent då spelhastigheten går att ändra från hemsidan.

"Bits by DAJW" är ett samarbete mellan 4 IoT-studenter på Nackademin i samband med ett projektarbete i kursen "Inbyggda System". I Rapporten beskrivs systemet, idéer om vidareutveckling samt reflektioner om arbetet.

Table of Contents

Bits by DAJW - Rapport

Introduktion

Table of Contents

1. System/design

1.1. Webserver/Databas

1.2. ESP32 WiFi Brygga

1.3. SpelEnhet

1.3.1 Arduino Uno

1.3.2. OLED Display

1.3.4. Knappar

1.4. C-Kod

1.4.1. Meny/states

1.4.2. Spel

1.4.3. Interrupts

1.4.4. Timers

1.4.5. Grafik

2. Systemskiss och Elscheman

3. Tidsestimat

4. Avgränsningar

5. Utvecklingsmöjligheter

5.1. Generellt

5.2. Design av låda

5.3. Batteridrift

5.4. Spelfunktioner

5.5. Säkerhet

5.6. Hårdvara/Mjukvara

6. Reflektion

1. System/design

1.1. Webserver/Databas

Webservern är byggd i NodeJS med Express, i nuläget går det att skicka och hämta data via HTTP GET requests. För att skicka data används URL-parametrar som sedan hanteras av webservern och skickas vidare till databasen. För att hämta data finns olika endpoints beroende på vilken data som önskas.

API:

- `/getHighscore` används för att hämta top 10 highscore från databasen.
- `/score?name=NAMN&score=SCORE&apiKey=*****` används för att skicka ett nytt score till databasen.
- `/getspeed` används för att hämta speedvärdet från databasen.
- `/setspeed?speed=1&apiKey=*****` används för att skicka in nya värden från hemsidan. Giltiga värden är 1, 2 och 3.

För att prata med databasen används mongoose biblioteket i NodeJS. Webservern hostas på heroku.

Databasen är byggd i NoSQL MongoDB Atlas. Det är en NoSQL databas som gör det enkelt att skicka in objekt i tex JSON format.

1.2. ESP32 WiFi Brygga

ESP32 från Adafruit används för att hantera WiFi kommunikation mellan Arduino och internet då Arduino Uno inte har någon egen WiFi kapacitet. Här används biblioteken `<WiFi.h>` och `<HTTPClient.h>` för att skapa WiFi uppkoppling samt kunna prata med webservern över HTTP protokollet. Även Arduinos standard bibliotek används. För att skicka data från Arduinon till webservern så skickas ett meddelande på UART, beroende på vad första byten som skickas är så kan ESP32 tolka vilken typ av meddelande som ska hanteras och skicka rätt request till webservern.

- '!' Används för att meddela att ett nytt score är på väg och ska skickas till webservern. ESP32 börjar då ta emot alla bytes tills det att den hittar newline character ('\n') och sparar det i en sträng som sedan parsas och skickas vidare så att webservern kan ta emot ett namn och ett score och spara i databasen.
- '@' används för att skicka en förfrågan om att hämta top 10 highscore. ESP32 skickar då en GET request till highscore endpointen och sparar det värde som returneras och skickar det vidare till arduinon som parsar det och visar det på displayen.
- '+' används för att skicka en förfrågan om speed, ESP32 skickar förfrågan till webserver och skickar tillbaka svaret till arduinon som då ändrar speed i spelet om ett nytt värde kom tillbaka. Speed ändras i samband med att man äter ett äpple för att begränsa trafiken på herokuservern.

1.3. SpelEnhet

1.3.1 Arduino Uno

Datablad: [ATmega328p](#)

Arduino Uno:s ATmega328P används för att driva själva spelet. Här används embedded C för att bygga spel och kommunikation med olika hårdvarudelar.

I ATmegan utnyttjas Flash, RAM, uart, i2c och timer.

- Flash (32Kb): C-koden flashas med make.
- RAM (2Kb): RAM används för att hantera grafik, spellogik och score.
- Uart: används för debug men främst för kommunikation med ESP32:an för att få information till och från hemsidan.
- I2C: kommunikation med OLED Display.
- timers: två stycken timers initieras, timer0 används för debounce of knappar och timer2 används för att driva spelet fram.

1.3.2. OLED Display


Återförsäljare: [OLED Display](#).

Datablad: [SSD1306](#)

SSD1306 är ett singel-chip CMOS OLED (organic / polymer light emitting diode). Chippet har inbyggt display RAM och oscillator. Kommandon skickas från MCU genom I2C men SPI är ett alternativ.

i2c är reserverad för kommunikation med OLED Displayen. För i2c används Matiasus färdiga bibliotek (<https://github.com/Matiasus/SSD1306>) som kommer med lämpliga funktioner för att driva grafik i meny och spel.

Matiasus biblioteket är GNU Licenserat:



Matiasus/SSD1306 is licensed under the
GNU General Public License v3.0

Permissions of this strong copyleft license are conditioned on making available complete source code of licensed works and modifications, which include larger works using a licensed work, under the same license. Copyright and license notices must be preserved. Contributors provide an express grant of patent rights.

Permissions	Limitations	Conditions
✓ Commercial use	✗ Liability	④ License and copyright notice
✓ Modification	✗ Warranty	④ State changes
✓ Distribution		④ Disclose source
✓ Patent use		④ Same license
✓ Private use		

1.3.4. Knappar

Fem stycken knappar, fyra knappar för att fungera som "pilar" för menykontroll och en "confirm"-knapp för att bekräfta val. Dessa kopplas till ATmega:s interrupt (PCINT0) samt debounce:as med timer0.

1.4. C-Kod

1.4.1. Meny/states

Hela programmet körs i en while-loop i funktionen "stateControl" som styr en struct med states för varje meny. Den innehåller "stateHandler" som läser av knapparna och "stateExecuter" som kör menygrafiken och snake.

1.4.2. Spel

När spelet startar går programmet in i "SnakeEngine" som sköter spellogik och grafik separat från stateHandler och stateExecuter som en egen statemachine tills spelet tar slut.

1.4.3. Interrupts

Interrupt vektor PCINT0 för digitalpin 8 till 12 initieras för hantering av knappar.

1.4.4. Timers

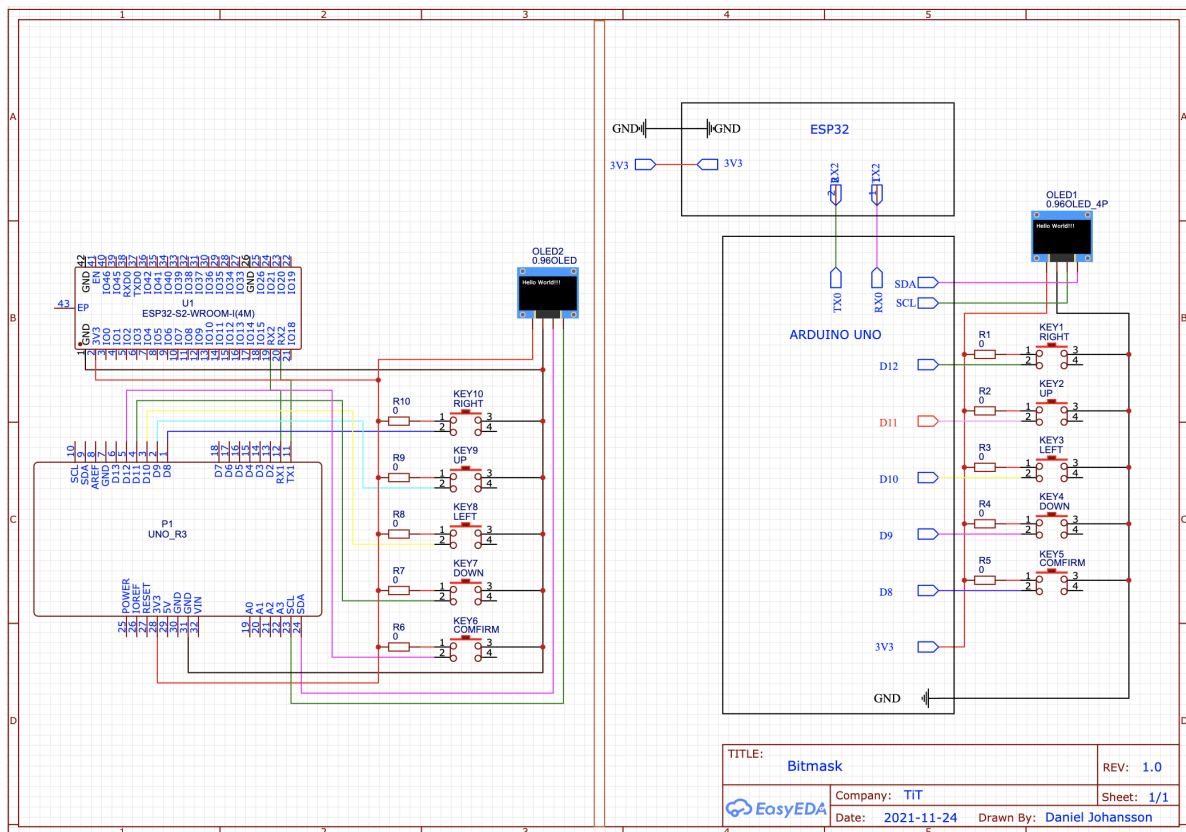
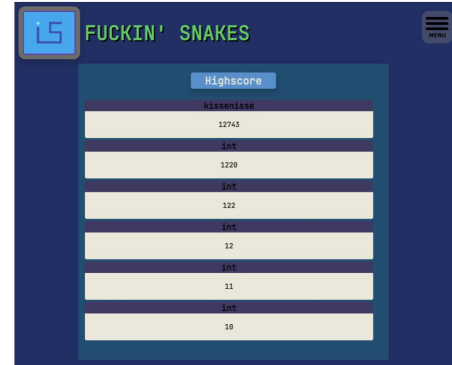
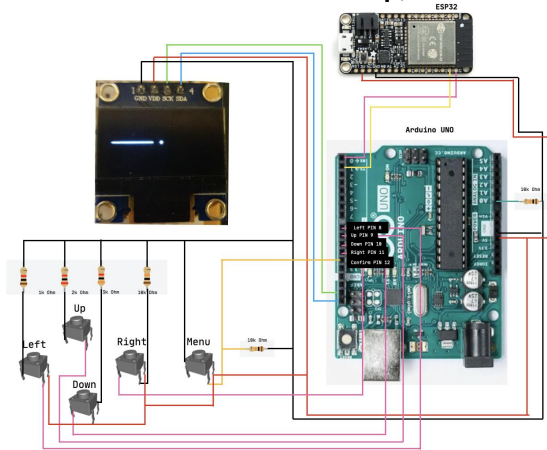
- Timer0: Initieras för att inkrementera en millisvariabel som används för att driva fram snake i valfri hastighet.
- Timer2: Initieras för att debounce:a knappar.

1.4.5. Grafik

Mattiasus Display-lib funktioner används i GFX.c för att bygga egna grafiska funktioner.

Exempel: Vi gör en "drawPixelSnake"-funktion som drar ner upplösningen från 128x64 pixlar till 32x16 pixlar. Detta gör spelet mer lätthanterligt samt sparar på det begränsade tillgängliga minnet då spelytan blir mindre och ormen därmed inte kan bli lika lång.

2. Systemskiss och Elschema



3. Tidsestimat

ITERATION 1			
Artikel	Ursprunglig Tidsestimering (timmar)	Uppdaterad Tidsestimering (timmar)	Startad/ Avslutad
Kanban board/Git repo	5	3	Avslutad
Implementera display lib	15	20	Avslutad
<u>Knappavläsning</u> +timer interrupts	5	15	Avslutad
Koppla upp Arduino mot internet	5	23	Avslutad
I2C kommunikation: Arduino och OLED Skärm*	20	*(Fanns i display lib)	Avslutad
Koppla databas mot nodejs	5	2	Avslutad
summa (h):	55	63	

ITERATION 2			
Artikel	Ursprunglig Tidsestimering (timmar)	Uppdaterad Tidsestimering (timmar)	Startad/ Avslutad
Snake funktionalitet	25	30	Avslutad
Grafisk interface	10	10	Avslutad
Statemachine: (delar nedan)	15	20	Avslutad
-Menysystem	5	10	Avslutad
-Highscore	5	5	Avslutad
-Snake	5	5	Avslutad
REST API för uppdatering av databas	20	20	Avslutad
summa (h):	70	80	

ITERATION 3	45 timmar (+ 10 extra marginal)		
Artikel	Ursprunglig Tidsestimering (timmar)	Uppdaterad Tidsestimering (timmar)	Startad/ Avslutad
Uart kommunikation: Arduino till ESP-enhet	15	2	Avslutad
Skicka score vid avslutat spel	10	3	Avslutad
Ta emot score från REST API	10	10	Avslutad
Highscore databas som uppdaterar hemsida	10	2	Avslutad
<i>summa (h):</i>	45	17	

ITERATION 4	20 timmar		
Artikel	Ursprunglig Tidsestimering (timmar)	Uppdaterad Tidsestimering (timmar)	Startad/ Avslutad
Kontrollera <u>spelparametrar</u>	20	2	Avslutad

ITERATION 5	80 timmar		
Artikel	Ursprunglig Tidsestimering (timmar)	Uppdaterad Tidsestimering (timmar)	Startad/ Avslutad
Kontrollera spelet från hemsidan	30	30	-
Multiplayer	30	30	-
Större spelyta	20	20	-

4. Avgränsningar

Under projektets gång har flera avgränsningar behövt göras. De avgränsningar som gjorts är baserade på att gruppen prioriterat att få klart en fungerande prototyp inom den fastställda tidsramen. De flesta avgränsningarna som gjorts är beskrivna i tidigare projekt rapporter men en kort summering av de avgränsningar som gjorts beskrivs nedan.

Gruppen valde att att spelfunktionalitet skulle byggas mot en Arduino Uno istället för ESP32. Detta på grund av Atmega chipets mer användarvänliga bibliotek samt gruppens tidigare erfarenhet av Atmega programmering. Då en del av kriterierna för projektet var att skriva egen embedded C kod så valdes Arduinon framför ESP32.

Avgränsning gällande styrning av spel från hemsidan samt multiplayer gjordes och dessa funktioner skulle endast byggas i mån av tid. Tyvärr har dessa funktioner inte implementerats under den givna tidsramen.

Avgränsning gällande eget bibliotek för skärm, denna avgränsning gjordes på grund av komplexiteten att skriva en egen drivrutin för grafikhantering. Även här fanns plan på att skriva ett eget om tid fanns.

Avgränsningar som gjort sen den senaste rapporten är att använda oss av ESP32 istället för ESP01 WiFi-modul. Detta då det tog för lång tid att få ESP01 till ett fungerande stadié.

5. Utvecklingsmöjligheter

5.1. Generellt

Det finns många utvecklingsmöjligheter som skulle göra produkten mer komplett än vad den är i nuläget men som inte kunde utvecklas på grund av den begränsade tiden.

5.2. Design av låda

En snyggare mer portabel design för själva lådan hade kunnat utvecklas med mer resurser och tid. I nuläget är den 3D printad hemma och baserad på att utvecklingskortet inte ska behöva lödas fast, därför används ett prototypframtagningsskort inuti lådan som gör att designen blir bulkigare än vad den behöver vara.

5.3. Batteridrift

För att göra produkten portabel hade batteridrift kunnat implementeras tillsammans med en avstängningsfunktion samt någon typ av "sleep"-funktion för att spara batteri när enheten inte används.

5.4. Spelfunktioner

En av många potentiella utvecklingsmöjligheter av spelet är att styra ormen från hemsidan. Detta implementerades inte på grund av brist på tid. Idén är då att med hjälp av knappar på hemsidan skicka samma kommando som hårdvaruknapparna gör idag. Någon typ av multiplayer funktionalitet hade också kunnat implementeras med mer resurser.

5.5. Säkerhet

Säkerheten kring kommunikationen mellan enheterna och webservern hade kunnat förbättras. Till exempel genom att använda certifikat för att verifiera att datat kommer från rätt avsändare. Ett alternativ som inte hann implementeras var att skicka HTTP POST requests för att på så sätt komma bort från url-parametrar när score skickas från spelet till webservern.

5.6. Hårdvara/Mjukvara

Med mer resurser hade bättre och mer ändamålsenlig hårdvara kunnat användas. Ett exempel är en större skärm för en större spelyta. Ett annat är att istället för att använda hela utvecklingskort hade vi kunnat använda oss av endast de komponenter vi behöver.

Vi hade även kunnat utveckla en egen drivrutin för display kommunikation och grafik. Med mer tid/erfarenhet skulle vi även kunnat bygga allt på ett chip som har inbyggd WiFi funktionalitet som exempelvis en ESP32 för att slippa ha två MCU:er.

6. Reflektion

På det stora hela har arbetet gått bra. Att bygga upp idén och implementera den i kod och hårdvara har varit utmanande men väldigt roligt. Det var en del moment och funktioner som behövde ses över och justeras men funktionaliteten från original idén är mestadels implementerad. Den största utmaningen har varit att förhålla sig till tidsbegränsningen då en del idéer inte kunde implementeras som det var tänkt från början pga tidsbrist.

Vi arbetade mycket tillsammans med de olika delarna för att se till att kommunikationen mellan de olika enheterna fungerade och för att lösa exempelvis parsing av data. Samarbetet mellan gruppmedlemmar fungerade väl och var en viktig del i att ta projektet i mål. Så fort en medlem i gruppen stötte på något problem så jobbade vi tillsammans för att lösa det och det blev naturligt för alla att sätta sig in i alla delar av systemet.

En av utmaningarna var att följa den inledande projektplanen. En del idéer som lät bra i teorin visade sig vara för tidskrävande för att implementera med de resurser som fanns och behövde lösas på ett annat sätt. Ett exempel är att det från början var tänkt att använda en WiFi-modul (ESP01) för WiFi kommunikation, men då det tog längre tid än planerat valde vi att byta till en ESP32 för att hinna klart inom den satta tidsramen.

Projektet gav en bra inblick i hur det är att gå från idé till implementation inom en begränsad tidsram. Att tvingas göra avgränsningar och justera delar av projektet på vägen gav en bra inblick i varför dokumentation av projekt är viktigt och att det är nyttigt med planering även om planen inte alltid följs exakt.