UNSTPB

FIMM

DMMP

Program studii: Mecatronică și Robotică

VOID – The Robot Pet

Coord. Științific:

As. Drd. Ing. Alexandra – Gabriela VASILESCU

Student:

TUDORAȘCU Tiberiu - Constantin

Grupa 523B

București

-2025-

Cuprins

Introducere

Am ales acest proiect deoarece îl consider o oportunitate creativă de a îmbina concepte de electronică, programare, proiectare, design și interacțiune om-mașină. Ideea principală a proiectului a apărut atunci când am aflat de termenul “Rubber duck debugging” [1], un fenomen des întâlnit în Ingineria Software, cât și în altele. Acesta constă în explicarea fiecărei linii de cod cu voce tare unei rățuște de cauciuc care este aflată pe birou. Explicatul cu voce tare ajută la descoperirea erorilor din program sau a bug-urilor deoarece te forțează să parcurgi logic fiecare pas și să observi detalii care poate înainte au fost trecute cu vederea. Între timp, Void a evoluat în a fii un mic robot care pe lângă compania oferită, prezintă și funcții care au scop recreativ sau de jucărie.

Acest proiect mi-a oferit ocazia de a aprofunda concepte de bază cum ar fii controlul precis al servo-motoarelor, programare în C aplicată pe microcontrolere, interfața cu senzori de temperatură și butoane, dar și unele mai avansate precum controlul sistemelor embedded folosind un microcontroller ca ESP32, animații/afișare grafică pe un ecran OLED și logica stărilor (finite state machine).

Pe lângă partea de programare și electronică, proiectul a presupus și realizarea unei carcase care să protejeze toate componentele. Având deja experiență în proiectarea și modelarea 3D, am putut să creez rapid o formă adaptată nevoilor robotului. Totuși, pentru că acesta era primul meu proiect fizic de acest tip, a fost necesar să învăț cum să optimizez designul pentru imprimare 3D și cum să iau în considerare limitările practice ale materialelor și ale imprimantei. Am întâmpinat diverse probleme legate de stabilitatea carcasei, poziționarea componentelor, rutarea firelor și fixarea ecranului, pe care le-am rezolvat pe parcurs, folosindu-mă de testări iterative și ajustări succesive ale designului, ajungând la o tematică de spațiu și cosmos, alegere care mi s-a părut cea mai potrivită pentru personalitatea sa.

Prin realizarea acestui proiect, am urmărit nu doar să construiesc un dispozitiv funcțional, ci și să înțeleg mai bine cum poate fi creat un sistem interactiv care răspunde la stimulii din jur și oferă feedback utilizatorului. Void reprezintă o primă treaptă spre înțelegerea si dezvoltarea unor roboți accesibili.

Stadiul actual

Unul dintre primiele dispozitive interactive de tip „pet” digital este Tamagotchi [Fig 1], care este un tip de dispozitiv care încape în buzunar și are forma unui breloc. Acesta a apărut în Japonia 1996, urmând sa fie lansat și pe piața din Statele Unite ale Americii, devenind foarte popular la începutul aniilor 2000 [2]. Motivul popularității sale a fost design-ul său mic si atractiv [Fig 2], cât și lucrurile care le putea face [3].

La prima pornire, pe ecran apărea un ou, din care urma să „eclozeze” animalul digital. De acolo, utilizatorul era responsabil de îngrijirea acestuia: trebuia să îl hrănească, să îl curețe și să îi mențină starea de bine, într-un mod similar cu grija acordată unui animal de companie real.

Fig 1.

Fig 2.

Dispozitivul original dispunea 3 butoane fizice denumite conventional: A,B & C [4].

Butonul A (stânga) era utilizat pentru navigarea în meniu. Acesta permitea utilizatorului să acceseze opțiunile disponibile cum ar fii hrănirea, curățarea, joaca sau verificarea stării animalului digital. Practic, acționa ca un selector de funcții.

Butonul B (mijloc) avea rolul de a confirma selecția făcută anterior cu butonul A.

Butonul C (dreapta) reprezenta butonul de anulare a unei acțiuni sau revenirea la ecranul principal. In versiuni mai noi acesta și rolul de a oprii temporar unele funcții, de exemplu notificările, sau sunetul.

Deși Tamagotchi a fost un pas important către crearea unui animal de companie robotic și accesibil, interacțiunea cu acesta era limitată. În ultimii ani, tehnologia a permis dezvoltarea unor roboți mult mai avansați ca Eilik și Cozmo [5]. Pe lângă faptul că aceștia pot simula emoții si expresii faciale, pun la dispoziție senzori, motoare, un design ergonomic, și cel mai important lucru, o interacțiune foarte avansată cu utilizatorul.

Cozmo [Fig 3], lansat în 2016 de compania Anki, a fost creat pentru a aduce roboții și inteligența artificială în casele oamenilor sub forma unui gadget drăguț și educational [6]. Scopul companiei a fost de a dezvolta un robot capabil să creeze o legătură emoțională cu utilizatorul.

Utilizatorii puteau programa comportamentele lui Cozmo folosind Code Lab, o platformă bazată pe blocuri vizuale, similară cu Scratch [7], destinată în special copiilor și începătorilor pentru a învăța conceptele de bază ale programării. Pentru cei interesați de programare avansată, Anki a oferit și Cozmo SDK(Software Development Kit), care permitea programarea robotului utilizând limbajul Python, oferind acces la funcționalități complexe precum controlul mișcărilor, recunoașterea fețelor și manipularea obiectelor [8].

Cozmo este echipat cu o cameră de 2MP care îi permite să recunoască fețele utilizatorilor și să reacționeze în funcție de familiaritatea cu persoana respectivă. De asemenea, robotul poate manipula obiecte, în special cuburile interactive incluse în kitul său, utilizând brațul mecanic articulat [9].

Datorită succesului lui Cozmo, în 2018, Anki a lansat succesorul său mai avansat, Vector [Fig 4] [10]. Acesta a fost orientat către un public mai matur, având capacități extinse de programare și funcționalități avansate, permițând dezvoltatorilor și entuziaștilor să creeze aplicații personalizate și să extindă capabilitățile robotului. SDK-ul oferă acces direct la senzorii avansați ai lui Vector, la capabilitățile sale de inteligență artificială și la tehnologiile sale robotice, inclusiv viziunea computerizată și navigație [11].

Cozmo e un robot cu care te joci mai mult prin aplicație. Are jocuri, recunoaște fețe și poate fi programat ușor prin blocuri sau în Python. Se bazează pe interacțiune directă și e gândit mai mult pentru învățare și distracție. Vector e mai autonom. Nu are nevoie de aplicație tot timpul, reacționează la voce, se mișcă singur și răspunde la întrebări. Poate fi și el programat, dar e mai serios, mai „asistent” decât „jucărie”.

Fig 3.

Fig 4.

Chiar dacă Vector este foarte avansat din punct de vedere a navigației și a reîncarcarii autonome, interacțiunea cu mediul, robotul meu preferat este Eilik [Fig 5].

Eilik a fost lansat in 2021 de către Energize Lab și creat pentru a exprima emoții și a interacționa într-un mod cât mai natural cu utilizatorul [12]. Față de Vector, Eilik nu are funcții de asistent vocal sau AI avansat, dar se concentrează pe partea emoțională: are gesturi, expresii și reacții care îl fac să pară viu. În loc să răspundă la întrebări sau să ofere informații, Eilik reacționează la atingere, vibrații sau la alți Eilik din apropiere [Fig 6], punând accent pe interacțiune afectivă, nu pe utilitate [13].

Fig 5.

Fig 6.

Un alt exemplu relevant din zona jocurilor interactive este aplicația Pou [Fig 7] [14], lansată în 2012, care simulează îngrijirea unei creaturi extraterestre. Spre deosebire de Tamagotchi sau roboți fizici precum Cozmo sau Eilik, Pou este complet digital și se desfășoară într-o aplicație mobilă. Cu toate acestea, principiul de bază rămâne similar: utilizatorul trebuie să aibă grijă de Pou oferindu-i mâncare, somn și curățenie, gestionând astfel mai multe bare de status precum hunger, cleanliness, energy și fun. Aplicația include și o serie de minigame-uri care cresc nivelul de fun și oferă monede în joc, cu care utilizatorul poate cumpăra mâncare, haine sau decorațiuni pentru casă [15]. În plus, există și un shop unde Pou poate fi personalizat cu haine, pălării și fundaluri, încurajând astfel creativitatea și atașamentul față de personaj cu care trăiește într-o casă digitală împărțită în mai multe camere [Fig 8] [16].

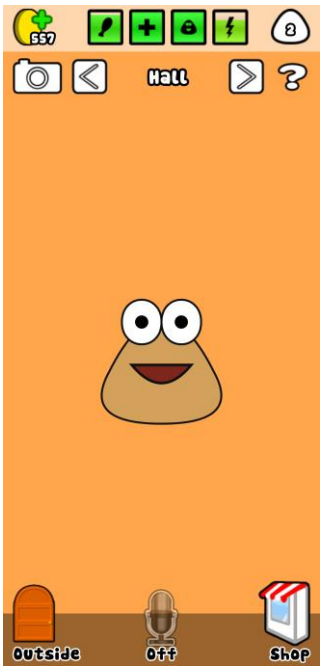

Fig 7.

Fig 8.

Fiecare cameră corespunde unei nevoi fundamentale: bucătăria pentru hrănire, baia pentru igienă, dormitorul pentru somn, camera de joacă pentru distracție, și așa mai departe. Fiecare dintre aceste aspecte este reprezentat vizual prin bare de status, precum hunger, health, cleanliness, energy sau fun, care trebuie menținute într-un interval optim de către utilizator [17]. Acest sistem de statusuri încurajează rutina zilnică, dar și atașamentul față de Pou, exact cum

se întâmplă și în cazul unui animal de companie virtual sau fizic. Față de roboți precum Vector, care mizează pe interacțiuni vocale și autonomie, sau Eilik, care pune accent pe emoții și expresii, Pou accentuează ideea de task based engagement, adică progresul se face prin îndeplinirea constantă a unor sarcini, iar recompensele sunt imediate și vizibile. Această abordare îl face ideal pentru a introduce utilizatorii în ideea de responsabilitate virtuală, dar și în logica interacțiunilor afective cu un companion digital [18].

Soluția propusă

Void țintește să combine emoțiile expresive ale lui Eilik cu interactivitatea și mini-jocurile inspirate din Cozmo, într-un prieten digital care nu încearcă să fie un asistent sau o unealtă, ci un personaj cu care să te distrezi, să creezi o legătură sau doar să-ți țină companie în serile lungi la birou. În plus, Void preia și elemente din aplicații precum Pou, unde grija față de personaj se manifestă prin menținerea unor bare de status, acces la mini-jocuri și camerele din casă, creând astfel un sistem complet de interacțiune și atașament afectiv între utilizator și robot.

Pentru a interacționa cu Void, sunt puse la dispoziție 3 butoane.

- Butonul de interacțiune este cel principal și este folosit în toate paginile. Acesta permite inceperea anumitor acțiuni, cum ar fi, tachinarea lui Void, accesarea informațiilor, a dulapului, frigiderului sau a magazinului, atat cât si cu obiectele din acestea.
- Butonul de navigare este folosit pentru schimbarea între cele 6 pagini, numite „taburi”, unde Void își afișează diferitele abilități.
- Ultimul buton este butonul de Exit sau de revenire la tab-ul principal.

Fiecare tab are un rol important. Tabul principal sau 0 este reprezentat de Void însăși. Aici el își arată emoțiile, interacționeaza cu utilizatorul și cu obiectele lui. Animațiile incluse în kit-ul său sunt:

- Idle: Atunci când Void este lăsat în pace, acesta stă liniștit, clipește natural, se uită în jur de parcă este atent la ceea ce faci.
- Happy: Apare atunci când butonul de interacțiune este apăsat de repetate ori. Acesta capătă un zambet și ecranul este acoperit de inimi. Picioarele din spate devin relaxate și gesticulează fericit către utilizator.
- Angry: Aveți grijă! Dacă Void este prins într-o stare proastă acesta o să arunce privire urâtă, dar după puțin timp îi trece.
- Hungry: Daca Void nu este hrănit la timp acesta poate devenii morocănos.
- Sleep: La fel ca oricine, Void iubește să doarmă, cu cât se joacă mai mult, obosește și are nevoie de o pauză bine-meritată. Atunci când doarme, picioarele lui sunt întinse și este afișată o animație de somn.
- Play: Această stare este opusul celei Sleepy. Depinzând de ce este echipat, Void ascultă muzică la un radio, sau se uită la televizor pentru a crește nivelul de diveristment.

Tabul 1 prezintă o interfață mai complexă: Câte o bară de progress pentru fiecare stare care duce la starea de bine complete a prietenului tău digital: Bara de viață (Health Bar), bara pentru mâncare (Hunger Bar), bara pentru nivelul de divertisment (Play Bar) și cea pentru oboseală(Sleep Bar). Pe aceeași pagina se observă, depinzând de temperatura din exterior, un soare, un fulg sau un termometru fierbinte sau înghețat. Acest factor de asemenea poate influența starea de bine.

- Soarele apare atunci când temperatura este între 19°C și 27°C, fiind un interval perfect pentru ca Void să se simtă bine. Bara de viață nu crește dar nici nu scade.
- Fulgul apare atunci când temperatura se află intre 15°C și 18°C . Chiar dacă nici aceasta nu influențează o bară de progres specifică, Void devine mai ușor de enervat, iar atunci când dorești sa interacționezi cu el s-ar putea să ai o surpriză.

- Termometrul înghețat apare atunci cand temperature pe care Void o prefer scade mult, și anume între 0°C și 14°C. Acest lucru face ca bara de divertisment sa crească într-un ritm mai rapid deoarece acesta iubește zăpada, dar devine mai înfometat mai repede.
- Termometrul fierbinte apare între 27°C si 40°C, ceea ce îl face să devină obosit mai ușor, iar bara de viață scade mai rapid.

Prin apăsarea butonului de interacțiune, se va deschide pe ecran un nou set de taburi, și anume cele de informații. Aici se pot vedea instrucțiuni detaliate despre fiecare necesitate a lui Void. Prima pagina afișeaza "On the next pages you will find general informations and how to keep your Void pet healthy, safe and active.". Prin apăsarea butonului de navigare este permisă trecerea prin fiecare status.

- Pentru health: "You can fill up Void's Health bar by petting him and keeping him well fed."
- Pentru hunger: "You can fill up Void's Hunger bar by feeding him in the kitchen."
- Pentru joy: "You can fill up Void's Joy bar by playing the different minigames. Don't try too hard, he doesn't mind losing."
- Pentru sleep: "After you're done with everything be sure to lay him to bed. He loves sleeping..Maybe too much."
- Pentru climat: "Be sure the temperature isn't too high or too low. He may be an alien, but he still likes comfort."

Dacă toate barele sunt ținute sub control, atât cât si temperatura este optima, Void se va simții bine si nu va avea un comportament imprevizibil.

Tabul 2 este reprezentat de dormitor. Aici utilizatorul este întâmpinat de un dulap si abilitatea de a intra într-un meniu cu mai multe opțiuni. Prima este “Sleep”, care odată selectată se transformă in “Wake up”, ceea ce îl pune pe Void la culcare. Acest lucru poate fi observant prin navigarea la tabul 0, observând ca acesta și-a schimbat animația din IDLE, navigarea la tabul 0, observând că acesta și-a schimbat animația din IDLE în starea de SLEEP. În această stare, Void acumulează puncte de somn pe măsură ce trece timpul, simulând procesul de odihnă. Dacă utilizatorul dorește să-l trezească, poate reveni în dormitor și selecta opțiunea "Wake up", revenind astfel la starea normală de IDLE. În schimb dacă Void este mângâiat apăsând butonul de interacțiune de 15 ori, acesta se va trezii involuntar și va fii setat pe starea ANGRY.

Tot în dormitor, prin navigarea în meniul de opțiuni, se poate accesa și modul "Fun", dedicat funcției de divertisment sau experimentare muzicală, câștigând puncte de PLAY. Astfel, dormitorul nu este doar locul unde Void se odihnește, ci și un centru de activități recreative.

Tabul 3 este reprezentat de bucătărie. Aici, utilizatorul este întâmpinat de imaginea unui frigider și are posibilitatea de a intra într-un meniu cu mai multe opțiuni. Meniul principal oferă două selecții: “Food” și “Drink”. Alegerea opțiunii “Food” îi oferă lui Void o cantitate mai mare de puncte de hunger, accelerând astfel umplerea barei de foame. Pe de altă parte, opțiunea “Drink” adaugă o cantitate mai mică de puncte de hunger, reprezentând o gustare rapidă sau hidratare ușoară. Dacă resursele de mâncare sau băutură se termină, acestea pot fi reînnoite din shop folosind moneda virtuală acumulată în minijocuri precum coinflip.

Prin folosirea acestor două opțiuni, utilizatorul poate gestiona eficient starea de foame a lui Void, adaptând cantitatea de hrană în funcție de nevoile curente ale companionului. Astfel, bucătăria devine un spațiu esențial pentru menținerea echilibrului necesităților vitale ale robotului, contribuind la starea lui generală de bine.

Tabul 4 este dedicat minijocului "Coinflip". Aici utilizatorul poate încerca să câștige monedă virtuală pentru Void, într-un joc de tip "noroc sau ghinion". La accesarea tabului, este afișată suma totală de bani deținută, împreună cu fundalul tematic pentru această secțiune.

Prin apăsarea butonului de interacțiune, începe animația aruncării unei monede, care rulează pentru câteva secunde. Rezultatul este decis aleatoriu: există șansa ca moneda să cadă pe "câștig" sau "pierdere". Dacă rezultatul este favorabil, utilizatorul câștigă o unitate de monedă care se adaugă automat la totalul său. Dacă rezultatul este nefavorabil, nu se câștigă nimic, dar utilizatorul poate încerca din nou oricând dorește.

Coinflip oferă astfel o metodă simplă, dar antrenantă de a aduna bani pentru Void, bani care ulterior pot fi folosiți pentru a cumpăra mâncare, băuturi sau alte obiecte de personalizare din shop.

Tabul 5 reprezintă magazinul (Shop-ul) lui Void. Aici utilizatorul poate folosi moneda virtuală câștigată din minijocul Coinflip pentru a achiziționa diverse obiecte utile.

În magazin sunt disponibile opțiuni precum:

- Mâncare: necesară pentru a crește rapid nivelul de hunger.
- Apă: folosită pentru a menține nivelul de hunger la valori optime pe termen mai lung.
- Televizor: un obiect special care, odată cumpărat, poate fi instalat în dormitor și înlocuiește radioul standard ca sursă de divertisment, oferind o creștere mai mare a nivelului de joy.

Fiecare produs are un cost în monedă virtuală și poate fi achiziționat dacă utilizatorul are suficiente resurse. Această funcționalitate încurajează administrarea atentă a banilor câștigați și introduce un element suplimentar de personalizare și progres în experiența de joc alături de Void.

Prin aceste șase taburi, utilizatorul are posibilitatea să exploreze lumea lui Void într-un mod interactiv și bine organizat. De la informarea despre nevoile sale de bază, la îngrijirea zilnică, divertisment și gestionarea resurselor, fiecare secțiune contribuie la crearea unei experiențe echilibrate și captivante. Sistemul de navigare intuitiv, combinat cu animațiile și funcționalitățile implementate, ajută la formarea unei legături autentice între utilizator și Void, transformând simpla interacțiune într-o rutină plăcută și plină de sens. Fiecare tab adaugă un strat suplimentar, oferind nu doar sarcini de îndeplinit, ci și momente de relaxare, joacă și satisfacție personală.

Arhitectura software

Pentru a asigura funcționarea corectă și ușor de extins a proiectului Void, am organizat software-ul pe mai multe niveluri funcționale bine definite. În continuare, voi descrie principalele componente ale arhitecturii software și modul în care acestea interacționează între ele. Proiectul este alcătuit din 3 fișiere: Void.ino, care reprezintă fișierul principal, nativ pentru Arduino IDE, unde se întâmplă toate activitățile; icons.h, un fișier de tip header unde sunt declarate array-urile de pointeri pentru animații cât și inițializarea variabilelor pentru dimensiuni și timp; icons.h unde sunt declarate array-urile de bitmapuri in PROGMEM.

Fișierul principal este configurat încât să fie modular, astfel fiind posibil să fie modificat oricând este nevoie si dezvoltat, fiecare funcționalitate majoră fiind tratată prin funcții dedicate și gestionată de logica principală din loop(). Acesta începe prin declararea fiecărei biblioteci necesară, a fișierelor header, dar și a ecranului.

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <ESP32Servo.h>
#include "images.h"
#include "DHT.h"
```

```
#include "icons.h"

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
#define SCREEN_ADDRESS 0x3C
#define DHTPIN 17
#define DHTTYPE DHT11

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
DHT dht(DHTPIN, DHTTYPE);
```

Biblioteca “Wire.h” are rolul de a inițializa comunicarea I2C (Inter-Integrated Circuit), oferind posibilitatea microcontroller-ului de a lucre cu ecranul OLED, iar cele de la AdaFruit lucrează împreună pentru a putea crea obiectul “Display” si pentru a putea desena pe el folosind funcții specifice precum “draw.bitmap()” sau “display.clearDisplay()”. Mai jos sunt declarați parametrii ecranului în pixeli, butonul de reset, cât și adresa sa unică de comunicare I2C. Înainte sa fie create obiectul, biblioteca ESP32Servo.h permite controlul servo-motoarelor, iar DHT.h este responsabilă pentru citirea datelor de la senzorul de temperatură și umiditate DHT11.

Pentru funcționarea corectă a proiectului, am declarat un set de variabile globale care gestionează atât starea generală a robotului, cât și parametrii necesari pentru interacțiuni și animații. Variabilele page și infoPage controlează navigarea între paginile principale și paginile informative. Pentru tratarea corectă a apăsărilor de butoane, sunt folosite variabilele lastButtonPress1, lastButtonPress2 și lastButtonPress3 împreună cu o constantă debounceDelay pentru a evita citirile multiple accidentale. Stările butoanelor sunt monitorizate prin variabilele apasat, apasat2 și apasat3. Structura FSM (Finite State Machine) este implementată prin enumerarea Mode, care definește modurile posibile de funcționare ale lui Void (IDLE, CUTE, ANGRY, SLEEP, PLAY, HUNGRY), împreună cu variabilele currentMode și lastMode pentru a urmări tranzițiile dintre stări.

Statusurile interne ale robotului sunt păstrate prin variabile precum healthpoints, angrypoints, sleeppoints, playpoints și hungerpoints. Temperatura citită de senzor este salvată în variabila temperature, iar diferite momente importante pentru actualizarea datelor sunt memorate folosind lastTempCheck, sleepTime și HPtime. Pentru controlul servo-motoarelor sunt definite obiectele Servo stangaS, dreaptaF, dreaptaS și stangaF, împreună cu pozițiile și unghiurile curente ale acestora, precum și timpii de ultimă mișcare lastMoveTimeSS, lastMoveTimeDF, lastMoveTimeDS și lastMoveTimeSF.

Pentru minigame-ul de tip Coinflip, sunt declarate variabile suplimentare precum money (pentru gestionarea monedelor), starea actuală a minigame-ului prin enumerarea CoinflipState (COINFLIP_IDLE, COINFLIP_FLIPPING, COINFLIP_SHOW_RESULT), timpul de start pentru jocul de monedă coinStartTime, rezultatul aruncării winlose și un flag bool coinInProgress pentru a gestiona tranzițiile corecte între etape.

```
int page = 0;
int infoPage = 0;

unsigned long lastButtonPress1 = 0;
unsigned long lastButtonPress2 = 0;
unsigned long lastButtonPress3 = 0;
const unsigned long debounceDelay = 200;
```

```
unsigned long lastTempCheck = 0;
int buttonPressCount = 0;
int buttonPressCount2 = 0;

bool apasat = false;
bool apasat2 = false;
bool apasat3 = false;
enum Mode
{
   IDLE,
   CUTE,
   ANGRY,
   SLEEP,
   PLAY,
   HUNGRY,
};
Mode currentMode = IDLE; // idle e activ initial
Mode lastMode = IDLE;

bool infoBool = false;
bool infoConfirm = false;

int totalPages = 5;
int totalInfoPages = 5;

float temperature;
int healthpoints = 0;
float angrypoints = 0;
int sleeppoints = 0;
int playpoints = 0;
int hungerpoints = 0;

int AngryTarget;

unsigned long sleepTime = 0;

unsigned long HPtime = 0;

Servo stangaS;
int PositionSS = 90;
int currentAngleSS;
Servo dreaptaF;
int PositionDF = 0;
int currentAngleDF;
Servo dreaptaS;
int PositionDS = 0;
int currentAngleDS;
Servo stangaF;
int PositionSF = 90;
int currentAngleSF;
int lastMoveTimeSS = 0;
int lastMoveTimeDF = 0;
```

```
int lastMoveTimeDS = 0;
int lastMoveTimeSF = 0;
int targetAngles1[] = {45, 0, 45, 0, 45, 0, 45, 0};

int money = 2000;
enum CoinflipState
{
  COINFLIP_IDLE,
  COINFLIP_FLIPPING,
  COINFLIP_SHOW_RESULT
};

CoinflipState coinState = COINFLIP_IDLE;
unsigned long coinStartTime = 0;
int winlose = 0;
bool coinInProgress = false;

int food = 4, drinks = 40;
bool haveTV = false, tvON = false;
bool haveCards = false, cardsON = false;
bool haveBook = false, bookON = false;
bool haveRadio = false, radioON = false;
int itemPage = 0;
bool pickingItem = false;
```

Funcția setup() este responsabilă pentru inițializarea componentelor hardware esențiale ale proiectului. În această secțiune, comunicarea serială este pornită pentru debugging prin Serial.begin(9600), iar senzorul DHT11 este inițializat folosind funcția dht.begin(). Display-ul OLED este pornit și verificat prin funcția display.begin(), iar în cazul în care inițializarea nu reușește, programul se blochează într-un loop infinit pentru a preveni funcționarea defectuoasă. Servo-motoarele sunt atașate la pinii corespunzători (stangaS pe pinul 18, dreaptaF pe 19, dreaptaS pe 15 și stangaF pe 23) și sunt setate în pozițiile lor inițiale, stocate în variabilele PositionSS, PositionDF, PositionDS și PositionSF. De asemenea, sunt configurate și cele trei butoane fizice ca intrări cu rezistență internă de pull-up. În final, ecranul este șters pentru a porni cu o imagine curată, pregătit pentru afișarea informațiilor necesare.

```
void setup()
{
  Serial.begin(9600);
  dht.begin();
  if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS))
  {
    Serial.println(F("Eroare la initializarea OLED!"));
    for (;;)
      ;
  }

  stangaS.attach(18);
  dreaptaF.attach(19);
  dreaptaS.attach(15);
  stangaF.attach(23);
```

```
  stangaS.write(PositionSS);
  currentAngleSS = PositionSS;
  dreaptaF.write(PositionDF);
  currentAngleDF = PositionDF;
  dreaptaS.write(PositionDS);
  currentAngleDS = PositionDS;
  stangaF.write(PositionSF);
  currentAngleSF = PositionSF;

  pinMode(4, INPUT_PULLUP);
  pinMode(16, INPUT_PULLUP);
  pinMode(5, INPUT_PULLUP);
  display.clearDisplay();
  display.display();
}
```

În cadrul funcției loop(), butoanele sunt gestionate printr-un sistem de citire periodică a stării fiecărui pin digital, cu implementarea unui mecanism de debounce pentru a prevenii apăsările multiple. Sunt folosite trei butoane, fiecare cu rol specific: Butonul 1 (pin 4) permite navigarea între paginile principale sau paginile de informații, în funcție de contextul activ (page sau infoPage). Butonul 2 (pin 16) are rol de interacțiune, permițând accesarea unui meniu suplimentar sau confirmarea unei acțiuni în cadrul acestuia. De asemenea, gestionează logica de acumulare a punctelor pentru stările speciale precum CUTE sau ANGRY. Butonul 3 (pin 5) resetează pagina activă sau închide un meniu secundar dacă acesta este deschis.

Apăsările sunt tratate non-blocking, adică fără a întrerupe restul execuției programului, folosind verificări pe bază de millis() pentru a implementa timpul minim dintre două citiri valide ale fiecărui buton (debounceDelay).

```
 if (digitalRead(4) == LOW && apasat && (currentMillis - lastButtonPress1 > debounceDelay))
  {
    apasat = false;
    lastButtonPress1 = currentMillis;
    if (infoBool)
      infoPage++;
    else
      page++;
    if (page > totalPages)
      page = 0;
    if (page == 1 && infoPage > totalInfoPages)
      infoPage = 0;
    if (pickingItem)
      itemPage++;
    if (itemPage > totalItems)
      itemPage = 0;
  }
  if (digitalRead(4) == HIGH)
    apasat = true;
```

```
    if (digitalRead(5) == LOW && apasat3 && (currentMillis - lastButtonPress3 > debounceDelay))
  {
    apasat3 = false;
    lastButtonPress3 = currentMillis;
    if (infoBool)
    {
      infoBool = false;
      infoConfirm = false;
      pickingItem = false;
      itemPage = 0;
    }
    else
    {
      page = 0;
    }
  }
  if (digitalRead(5) == HIGH)
    apasat3 = true;

    if (digitalRead(16) == LOW && apasat2 && (currentMillis - lastButtonPress2 > debounceDelay))
  {
    apasat2 = false;
    lastButtonPress2 = currentMillis;

    if (page != 0)
    {
      if (!infoBool)
      {
        infoBool = true;
        infoConfirm = false;
      }
      else
      {
        infoConfirm = true;
        Serial.println("Confirmare in meniu activata.");
      }
      if (!infoBool)
        previousTime2 = currentMillis;
    }

    else if (page == 0)
    {
      buttonPressCount++;
      Serial.println(buttonPressCount);
      Serial.print("AngryPoints: ");
      Serial.println(angrypoints);
       AngryTarget = temperature >= 19 && temperature <= 27 ? 15 : temperature >= 15 && temperature <= 18 ? 10
                                                                                                        : 5;
      if (buttonPressCount == AngryTarget)
```

```
        {
          if (currentMode == SLEEP)
          {
            currentMode = ANGRY;
            if (lastMode != currentMode)
              lastMode = currentMode;
            buttonPressCount = 0;
          }
          else if (angrypoints == 3)
          {
            currentMode = ANGRY;
            if (lastMode != currentMode)
              lastMode = currentMode;
            buttonPressCount = 0;
          }
          else if (angrypoints < 3)
          {
            currentMode = CUTE;
            if (lastMode != currentMode)
              lastMode = currentMode;
            angrypoints = angrypoints + 0.5;
            Serial.print("Health: ");
            Serial.println(healthpoints);
            buttonPressCount = 0;
          }
        }
      }
    }
    if (digitalRead(16) == HIGH)
      apasat2 = true;
```

După logica de citire a butoanelor, programul gestionează comportamentul robotului în funcție de pagina activă (page) și de starea curentă (currentMode). Pentru pagina principală (pagina 0), funcția loop() folosește o mașină de stări finite (FSM), apelând funcția corespunzătoare fiecărei stări (idle(), cute(), angry(), sleepy()). În funcție de acțiunile utilizatorului sau de diverși factori (de exemplu, temperatura sau nivelurile interne de status), robotul poate schimba automat starea de funcționare. Pentru celelalte pagini (progress(), stats(), bedroom(), kitchen(), coinflip(), shop()), programul redă informații sau permite interacțiuni specifice contextului selectat. În paralel, în funcție de starea activă (SLEEP, IDLE, CUTE), se gestionează poziția servo-motoarelor, astfel încât Void să aibă o postură diferită: în poziție normală, relaxată sau într-o mișcare de joacă. Funcția servoMovement() este apelată pentru fiecare motor, cu unghiurile țintă corespunzătoare modului activ, asigurând o mișcare lină și realistă a robotului.

```
static int lastPage = -1;
  if (page != lastPage)
  {
    Serial.print("Pagina curenta: ");
    Serial.println(page);
    lastPage = page;
  }
  static bool wasOnPage0 = false;
```

```
if (page == 0)
{
   if (!wasOnPage0)
   {
      currentFrames = 0;
      wasOnPage0 = true;
   }
   if (currentMode == IDLE)
      idle();
   else if (currentMode == CUTE)
      cute(healthpoints);
   else if (currentMode == ANGRY)
      angry(healthpoints);
   else if (currentMode == SLEEP)
      sleepy(sleeppoints);
   else if (currentMode == PLAY)
      idleplay(playpoints);
}
else
{
   wasOnPage0 = false;
}
if (page == 1)
{
   if (infoBool)
      stats();
   else
      progress(healthpoints, sleeppoints, hungerpoints, playpoints);
}
else if (page == 2)
{
   bedroom();
}
else if (page == 3)
{
   kitchen(hungerpoints, drinks, food);
}
else if (page == 4)
{
   coinflip(money);
}
else if (page == 5)
{
   shop(money, drinks, food, haveCards, haveBook, haveRadio, haveTV);
}
if (currentMode == SLEEP)
{
   servoMovement(stangaS, 0, currentAngleSS, lastMoveTimeSS);
   servoMovement(dreaptaF, 90, currentAngleDF, lastMoveTimeDF);
   servoMovement(dreaptaS, 90, currentAngleDS, lastMoveTimeDS);
   servoMovement(stangaF, 0, currentAngleSF, lastMoveTimeSF);
}
```

```
  if (currentMode == IDLE)
  {
    servoMovement(stangaS, 90, currentAngleSS, lastMoveTimeSS);
    servoMovement(dreaptaF, 0, currentAngleDF, lastMoveTimeDF);
    servoMovement(dreaptaS, 0, currentAngleDS, lastMoveTimeDS);
    servoMovement(stangaF, 90, currentAngleSF, lastMoveTimeSF);
  }
  if (currentMode == CUTE)
  {
    servoMovement(stangaS, 45, currentAngleSS, lastMoveTimeSS);
    servoMovement(dreaptaS, 45, currentAngleDS, lastMoveTimeDS);
    servoMovement(dreaptaF, 0, currentAngleDF, lastMoveTimeDF);
    servoMultipleMovement(stangaF, targetAngles1, 8, currentAngleSF, lastMoveTimeSF);
  }
```

Mișcările servo-motoarelor sunt controlate prin funcții dedicate pentru a asigura o deplasare treptată și fluidă, fără blocarea execuției programului. Funcția servoMovement() gestionează mișcarea unui singur servo către un unghi țintă într-un mod gradual, folosind o întârziere controlată (stepDelay) și o incrementare sau decrementare a poziției (step) la fiecare apel. Această abordare permite mișcări realiste și evită schimbările bruște de poziție.

Pentru cazuri în care un servo trebuie să urmeze o succesiune de unghiuri țintă, funcția servoMultipleMovement() parcurge un array de poziții predefinite. Această funcție folosește o variabilă statică internă pentru a reține progresul în cadrul array-ului între apeluri succesive și actualizează poziția servo-ului la intervale regulate. Dacă s-a ajuns la finalul secvenței, indexul este resetat, permițând reluarea mișcării.

```
void servoMovement(Servo &s, int targetAngle, int &currentAngle, int &lastMoveTime, int
step = 5, unsigned long stepDelay = 10)
{
  unsigned long currentTime7 = millis();
  if (currentTime7 - lastMoveTime >= stepDelay)
  {
    lastMoveTime = currentTime7;
    if (currentAngle < targetAngle)
    {
      currentAngle += step;
      if (currentAngle > targetAngle)
        currentAngle = targetAngle;
      s.write(currentAngle);
    }
    else if (currentAngle > targetAngle)
    {
      currentAngle -= step;
      if (currentAngle < targetAngle)
        currentAngle = targetAngle;
      s.write(currentAngle);
    }
  }
}
```

```
void servoMultipleMovement(Servo &s, int targetAngles[], int size, int &currentAngle, int &lastMoveTime, int step = 5, unsigned long stepDelay = 15)
{
  static int i = 0;
  unsigned long currentTime = millis();

  if (i >= size)
    i = 0;

  if (currentTime - lastMoveTime >= stepDelay)
  {
    lastMoveTime = currentTime;
    int target = targetAngles[i];

    if (currentAngle < target)
    {
      currentAngle += step;
      if (currentAngle > target)
        currentAngle = target;
      s.write(currentAngle);
    }
    else if (currentAngle > target)
    {
      currentAngle -= step;
      if (currentAngle < target)
        currentAngle = target;
      s.write(currentAngle);
    }

    if (currentAngle == target)
    {
      i++;
    }
  }
}
```

Un exemplu clar de modularitate în cod este gestionarea dormitorului prin funcția bedroom(). Această funcție controlează atât partea grafică (afișarea dormitorului și a dulapului static sau a animației de fundal), cât și logica de interacțiune a utilizatorului. În funcție de variabila infoBool, funcția decide dacă afișează o simplă animație a camerei sau intră într-un meniu interactiv unde utilizatorul poate selecta acțiuni precum "Sleep", "Wake Up" sau "Pick Item". Se folosește navigarea între pagini prin infoPage și confirmarea acțiunilor prin infoConfirm, schimbând starea robotului (currentMode) în funcție de selecție.

În plus, funcția bedroom() gestionează și sistemul de echipare și dezechipare a obiectelor (itemelor), cum ar fi Cards, Book, Radio sau TV. La selectarea unui item, acesta devine activ, dezactivând automat celelalte pentru a evita conflictele, iar starea robotului trece în modul PLAY. Dacă utilizatorul selectează din nou același item, acesta este dezechipat și robotul revine în modul IDLE. Această logică asigură un flux intuitiv al interacțiunii, păstrând în același timp controlul asupra afișajului și comportamentului intern al sistemului.

În paralel, funcția sleepy() gestionează comportamentul robotului atunci când acesta doarme. Aceasta redă o animație de somn cadru cu cadru, crește periodic punctajul de somn (sleeppoints), și verifică dacă Void a acumulat suficient somn pentru a reveni automat în starea

de principală (IDLE). În plus, dacă utilizatorul abuzează de butonul de interacțiune în timp ce robotul doarme, acesta poate deveni nervos, schimbându-și astfel comportamentul. Acest lucru se întamplă la și la celelalte camere, folosind același principiu de funcționare: O funcție dedicată pentru partea grafică, iar una care lucrează împreună pentru a modifica starea și interacțiunea cu robotul.

```
void bedroom()
{
   unsigned long currentTime6 = millis();

   if (!pickingItem)
   {
      if (!infoBool)
      {
         if (currentTime6 - previousTime6 >= 300)
         {
            previousTime6 = currentTime6;
            display.clearDisplay();
           display.drawBitmap(0, 0, bedrooms[currentFrames6], 128, 64, SSD1306_WHITE);
            display.setTextColor(SSD1306_WHITE);
            display.setTextSize(1);
            display.setCursor(0, 0);
            display.println("Bedroom");

            display.display();
            currentFrames6++;
            if (currentFrames6 >= totalFrames6)
               currentFrames6 = 0;
         }
      }
      else
      {
         display.clearDisplay();
         display.drawBitmap(0, 0, roomStatic[0], 128, 64, SSD1306_WHITE);
         display.setCursor(0, 0);
         display.setTextColor(SSD1306_WHITE);

         if (currentMode == SLEEP)
         {
            if (infoPage == 0)
            {
               display.setTextSize(1.5);
               display.println(">Wake Up");
               display.setTextSize(1);
               display.println("Items");

               if (infoConfirm)
               {
                  currentMode = IDLE;
                  infoConfirm = false;
               }
            }
```

```
            else if (infoPage == 1)
            {
               display.setTextSize(1);
               display.println("Wake Up");
               display.setTextSize(1.5);
               display.println(">Pick Item");

               if (infoConfirm)
               {
                  pickingItem = true;
                  itemPage = 0;
                  infoConfirm = false;
               }
            }
         }
         else
         {
            if (infoPage == 0)
            {
               display.setTextSize(1.5);
               display.println(">Sleep");
               display.setTextSize(1);
               display.println("Items");

               if (infoConfirm)
               {
                  currentMode = SLEEP;
                  infoConfirm = false;
               }
            }
            else if (infoPage == 1)
            {
               display.setTextSize(1);
               display.println("Sleep");
               display.setTextSize(1.5);
               display.println(">Pick Item");

               if (infoConfirm)
               {
                  pickingItem = true;
                  itemPage = 0;
                  infoConfirm = false;
               }
            }
         }

         if (infoPage > 1)
            infoPage = 0;

         display.display();
      }
   }
```

```
else
{
  display.clearDisplay();
  display.drawBitmap(0, 0, roomStatic[0], 128, 64, SSD1306_WHITE);
  display.setCursor(0, 0);
  display.setTextColor(SSD1306_WHITE);

  if (itemPage == 0)
    display.println(">Cards");
  else if (itemPage == 1)
    display.println(">Book");
  else if (itemPage == 2)
    display.println(">Radio");
  else if (itemPage == 3)
    display.println(">TV");

  if (infoConfirm)
  {
    if (itemPage == 0 && haveCards)
    {
      if (cardsON)
      {
        cardsON = false;
        currentMode = IDLE;
      }
      else
      {
        cardsON = true;
        bookON = false;
        radioON = false;
        tvON = false;
        currentMode = PLAY;
      }
    }
    else if (itemPage == 1 && haveBook)
    {
      if (bookON)
      {
        bookON = false;
        currentMode = IDLE;
      }
      else
      {
        cardsON = false;
        bookON = true;
        radioON = false;
        tvON = false;
        currentMode = PLAY;
      }
    }
    else if (itemPage == 2 && haveRadio)
    {
```

```
            if (radioON)
            {
               radioON = false;
               currentMode = IDLE;
            }
            else
            {
               cardsON = false;
               bookON = false;
               radioON = true;
               tvON = false;
               currentMode = PLAY;
            }
         }
         else if (itemPage == 3 && haveTV)
         {
            if (tvON)
            {
               tvON = false;
               currentMode = IDLE;
            }
            else
            {
               cardsON = false;
               bookON = false;
               radioON = false;
               tvON = true;
               currentMode = PLAY;
            }
         }
         else
         {
            display.clearDisplay();
            display.setCursor(0, 0);
            display.setTextSize(1);
            display.println("LOCKED ITEM");
            display.display();
            delay(1000);
         }

         infoConfirm = false;
         pickingItem = false;
         infoBool = false;
         infoPage = 0;
         itemPage = 0;
      }

      if (itemPage > 3)
         itemPage = 0;

      display.display();
   }}
```

Un exemplu de funcție care are un singur rol este shop(), care desenează si asigură cumpărarea obiectelor în același timp, folosind multe bool-uri.

```
void shop(int &money, int &drinks, int &food, bool &haveCards, bool &haveBook, bool &haveRadio, bool &haveTV)
{
  unsigned long currentTime9 = millis();

  if (!infoBool)
  {
    if (currentTime9 - previousTime9 >= 300)
    {
      previousTime9 = currentTime9;
      display.clearDisplay();
      display.drawBitmap(0, 0, shops[currentFrames9], 128, 64, SSD1306_WHITE);
      display.setTextColor(SSD1306_WHITE);
      display.setTextSize(1);
      display.setCursor(0, 0);
      display.println("Shop");

      display.display();
      currentFrames9++;
      if (currentFrames9 >= totalFrames9)
        currentFrames9 = 0;
    }
  }
  else
  {
    display.clearDisplay();
    if (currentMode != SLEEP)
    {
      display.setCursor(0, 0);
      display.setTextColor(SSD1306_WHITE);
      display.setTextSize(1);

      if (infoPage == 0)
      {
        display.print(">Buy Water $5\n");
        display.println("Buy Burger");
        display.println("Buy Cards");
        display.println("Buy Book");
        display.println("Buy Radio");
        display.println("Buy TV");

        if (infoConfirm && money >= 5)
        {
          drinks++;
          money -= 5;
          Serial.println("Bought Water");
          infoConfirm = false;
          infoPage = 0;
        }
```

```
    }
    else if (infoPage == 1)
    {
      display.println("Buy Water");
      display.print(">Buy Burger $10\n");
      display.println("Buy Cards");
      display.println("Buy Book");
      display.println("Buy Radio");
      display.println("Buy TV");

      if (infoConfirm && money >= 10)
      {
        food++;
        money -= 10;
        Serial.println("Bought Burger");
        infoConfirm = false;
        infoPage = 0;
      }
    }
    else if (infoPage == 2)
    {
      display.println("Buy Water");
      display.println("Buy Burger");
      display.print(">Buy Cards $20\n");
      display.println("Buy Book");
      display.println("Buy Radio");
      display.println("Buy TV");

      if (infoConfirm && money >= 20 && !haveCards)
      {
        haveCards = true;
        money -= 20;
        Serial.println("Bought Cards");
        infoConfirm = false;
        infoPage = 0;
      }
      else if (infoConfirm && haveCards)
      {
        Serial.println("Already own Cards!");
        infoConfirm = false;
      }
    }
    else if (infoPage == 3)
    {
      display.println("Buy Water");
      display.println("Buy Burger");
      display.println("Buy Cards");
      display.print(">Buy Book $30\n");
      display.println("Buy Radio");
      display.println("Buy TV");

      if (infoConfirm && money >= 30 && !haveBook)
```

```
        {
          haveBook = true;
          money -= 30;
          Serial.println("Bought Book");
          infoConfirm = false;
          infoPage = 0;
        }
        else if (infoConfirm && haveBook)
        {
          Serial.println("Already own Book!");
          infoConfirm = false;
        }
      }
      else if (infoPage == 4)
      {
        display.println("Buy Water");
        display.println("Buy Burger");
        display.println("Buy Cards");
        display.println("Buy Book");
        display.print(">Buy Radio $50\n");
        display.println("Buy TV");

        if (infoConfirm && money >= 50 && !haveRadio)
        {
          haveRadio = true;
          money -= 50;
          Serial.println("Bought Radio");
          infoConfirm = false;
          infoPage = 0;
        }
        else if (infoConfirm && haveRadio)
        {
          Serial.println("Already own Radio!");
          infoConfirm = false;
        }
      }
      else if (infoPage == 5)
      {
        display.println("Buy Water");
        display.println("Buy Burger");
        display.println("Buy Cards");
        display.println("Buy Book");
        display.println("Buy Radio");
        display.print(">Buy TV $200\n");

        if (infoConfirm && money >= 200 && !haveTV)
        {
          haveTV = true;
          money -= 200;
          Serial.println("Bought TV");
          infoConfirm = false;
          infoPage = 0;
```

```
            }
            else if (infoConfirm && haveTV)
            {
               Serial.println("Already own TV!");
               infoConfirm = false;
            }
         }

         if (infoPage > 5)
            infoPage = 0;
      }
      else
      {
         display.drawBitmap(0, 0, shops[currentFrames8], 128, 64, SSD1306_WHITE);
         display.setCursor(0, 0);
         display.setTextColor(SSD1306_WHITE);
         display.setTextSize(1);
         display.println("Shop");
         display.println("Void is asleep");
      }
      display.display();
   }
}
```

Pentru reprezentarea vizuală a stării robotului, am implementat patru bare de progres care indică nivelurile de sănătate, foame, bucurie și somnolență. Aceste bare sunt desenate pe ecran și sunt umplute dinspre bază către partea superioară, pentru a reflecta intuitiv creșterea valorilor respective. Realizarea acestui efect a necesitat o adaptare a poziției de desenare: înălțimea fiecărei bare este calculată prin funcția map(), care convertește valorile de la 0 la 100 puncte într-o scară de 0 până la 40 pixeli, corespunzând dimensiunii maxime a barelor. Poziția verticală de început (y) este ajustată dinamic în funcție de înălțimea obținută, astfel încât umplerea să înceapă de jos în sus. Această abordare oferă o reprezentare vizuală corectă și plăcută a progresului fiecărui parametru, ajutând utilizatorul să observe rapid starea generală a companionului virtual. De asemenea, aici se poate observa și cum temperature influențează bara de viață sau somn.

```
void progress(int &healthpoints, int &sleeppoints, int &hungrypoints, int &playpoints)
{
   unsigned long currentTime2 = millis();
   display.clearDisplay();

   if (currentTime2 - previousTime2 >= 300)
   {
      previousTime2 = currentTime2;
      currentFrames2++;
      if (currentFrames2 >= 3)
         currentFrames2 = 0;
   }

   if (temperature >= 19 && temperature <= 27)
```

```
    {
      display.drawBitmap(0, 0, progiconsSun[currentFrames2], 128, 64, SSD1306_WHITE);
      if (currentTime2 - HPtime >= 300000)
    {
      healthpoints-=5;
      sleeppoints-=5;
      hungrypoints -=5;
      playpoints -=5;
      HPtime = currentTime2;
    }
    }
    else if (temperature >= 15 && temperature <= 18)
    {
      display.drawBitmap(0, 0, progiconsCold[currentFrames2], 128, 64, SSD1306_WHITE);

    }
    else if (temperature >= 0 && temperature <= 14)
    {
                display.drawBitmap(0, 0, progiconsFreeze[currentFrames2], 128, 64,
SSD1306_WHITE);
      if (currentTime2 - HPtime >= 300000)
    {
      healthpoints-=5;
      sleeppoints-=5;
      hungrypoints -=7;
      playpoints += 2;
      HPtime = currentTime2;
    }
    }
    else if (temperature >= 28 && temperature <= 40)
    {
      display.drawBitmap(0, 0, progiconsHot[currentFrames2], 128, 64, SSD1306_WHITE);
      if (currentTime2 - HPtime >= 300000)
    {
      healthpoints-=10;
      sleeppoints-=10;
      hungrypoints -=5;
      playpoints -=5;
      HPtime = currentTime2;
    }
    }



    display.drawRoundRect(5, 16, 10, 40, 5, SSD1306_WHITE);   // Health
    display.drawRoundRect(25, 16, 10, 40, 5, SSD1306_WHITE);  // Hunger
    display.drawRoundRect(45, 16, 10, 40, 5, SSD1306_WHITE);  // Joy
    display.drawRoundRect(65, 16, 10, 40, 5, SSD1306_WHITE);  // Sleep

    int barHeightHealth = map(healthpoints, 0, 100, 0, 40);
    int barYHealth = 16 + (40 - barHeightHealth);
    display.fillRoundRect(5, barYHealth, 10, barHeightHealth, 5, SSD1306_WHITE);
```

```
    // HUNGER BAR
    int barHeightHunger = map(hungrypoints, 0, 100, 0, 40);
    int barYHunger = 16 + (40 - barHeightHunger);
    display.fillRoundRect(25, barYHunger, 10, barHeightHunger, 5, SSD1306_WHITE);

    // JOY BAR
    int barHeightJoy = map(playpoints, 0, 100, 0, 40);
    int barYJoy = 16 + (40 - barHeightJoy);
    display.fillRoundRect(45, barYJoy, 10, barHeightJoy, 5, SSD1306_WHITE);

    // SLEEP BAR
    int barHeightSleep = map(sleeppoints, 0, 100, 0, 40);
    int barYSleep = 16 + (40 - barHeightSleep);
    display.fillRoundRect(65, barYSleep, 10, barHeightSleep, 5, SSD1306_WHITE);

    display.display();
}
```

Mecanica mini-game-ului „Coinflip” a fost implementată printr-o funcție structurată în trei stări distincte, utilizând o instrucțiune switch-case. Prima stare (COINFLIP_IDLE) inițiază procesul de aruncare a monedei, generând aleatoriu un rezultat câștigător sau pierzător. În a doua stare (COINFLIP_FLIPPING), este redată o animație fluidă a rotirii monedei folosind un set de cadre succesive.

După finalizarea animației funcția trece în ultima stare (COINFLIP_SHOW_RESULT), unde este afișat rezultatul final, iar în caz de câștig, utilizatorului îi este adăugat un punct la scorul total de bani.

```
void coinflip(int &money)
{
    unsigned long currentTime7 = millis();
    display.clearDisplay();
    display.setTextColor(SSD1306_WHITE);
    display.setTextSize(1);
    display.setCursor(0, 0);
    display.print("$");
    display.println(money);

    if (!infoBool)
    {
        display.drawBitmap(0, 0, roomStatic[1], 128, 64, SSD1306_WHITE);
    }
    else
    {
        switch (coinState)
        {
        case COINFLIP_IDLE:
            coinInProgress = true;
            coinStartTime = currentTime7;
            currentFrames7 = 0;
            winlose = random(1, 3); // 1 win, 2 lose
```

```
        coinState = COINFLIP_FLIPPING;
        break;

    case COINFLIP_FLIPPING:
            if (currentTime7 - previousTime7 >= framestime7[currentFrames7] &&
currentFrames7 < totalFrames7)
        {
          previousTime7 = currentTime7;
                display.drawBitmap(0, 0, coinflipanimation[currentFrames7], 128, 64,
SSD1306_WHITE);
          currentFrames7++;
        }
        else if (currentFrames7 >= totalFrames7)
        {
          coinStartTime = currentTime7;
          coinState = COINFLIP_SHOW_RESULT;
        }
        break;

    case COINFLIP_SHOW_RESULT:
        if (winlose == 1)
        {
          display.drawBitmap(0, 0, winloseframes[0], 128, 64, SSD1306_WHITE);
        }
        else
        {
          display.drawBitmap(0, 0, winloseframes[1], 128, 64, SSD1306_WHITE);
        }

        if (currentTime7 - coinStartTime >= 2300)
        {       if (winlose == 1){
          money++;}
          infoBool = false;
          coinInProgress = false;
          coinState = COINFLIP_IDLE;
        }
        break;
    }
  }

  display.display();
}
```

Fișierul icons.h are rolul de a organiza și declara toate resursele grafice necesare pentru animațiile și interfețele grafice ale robotului Void. În acest fișier sunt definite array-uri de pointeri către imagini bitmap stocate în memoria flash (PROGMEM). Aceste imagini sunt utilizate pentru a reda animații sau imagini statice pe ecranul OLED.

De exemplu, array-ul frames[] conține cadrele animației de stare IDLE, în timp ce frames3[] conține cadrele animației din starea CUTE, iar frames4[] pentru starea ANGRY. Alte array-uri, precum bedrooms[], progiconsSun[], progiconsCold[], progiconsHot[] și progiconsFreeze[], sunt utilizate pentru a anima zonele speciale ale aplicației, cum ar fi dormitorul sau prognoza climatică vizualizată în tabul de status.

Pe lângă bitmap-urile grafice, icons.h definește și array-uri pentru timpul de afișare al fiecărui cadru (framestime[], framestime3[], framestime4[], etc.), permițând un control precis al vitezei de redare a animațiilor. Fiecare array de timpi asociază un interval de timp în milisecunde fiecărui cadru individual, rezultând animații fluente și adaptabile.

De asemenea, fișierul conține variabile de stare pentru gestionarea animațiilor (currentFrames, currentFrames2, currentFrames3, etc.) și timpi anteriori (previousTime, previousTime2, etc.), necesare pentru implementarea logicii non-blocate de animare folosind funcția millis().

```
const unsigned char *frames[] = { happyidle1, happyidle2, happyidle3, happyidle4, happyidle5, happyidle6,
                    happyidle7, happyidle8, happyidle9, happyidle8, happyidle7, happyidle6, happyidle5, happyidle4, happyidle3, happyidle2,
                    happyidle1, happyidle1, happyidle2, happyidle3, happyidle4, happyidle5, happyidle6, happyidle7, happyidle8, happyidle9,
                    happyidle8, happyidle7, happyidle6, happyidle5, happyidle4, happyidle3, happyidle2, happyidle1, happyidle1, happyidle2,
                    happyidle3, happyidle4, happyidle5, happyidle6, happyidle7, happyidle8, happyidle9, happyidle8, happyidle7, happyidle6,
                                happyidle5, happyidle4, happyidle3, happyidle2, happyidle1, happyidle10, happyidle11, happyidle12, happyidle13, happyidle14,
                     happyidle13, happyidle12, happyidle11, happyidle10, happyidle1 };
const unsigned char *frames3[] = { cute1, cute2, cute3, cute4, cute5, cute6, cute7, cute6, cute5, cute4, cute3, cute2, cute1 };
const unsigned char *frames4[] = { angry1, angry2, angry3, angry4, angry5, angry6, angry7, angry8, angry7, angry8, angry7, angry8, angry7, angry8, angry7, angry8, angry7, angry8, angry6, angry5, angry4, angry3, angry2, angry1 };
const unsigned char *frames5[] = { sleep1, sleep2, sleep3, sleep4 };
const unsigned char *bedrooms[] = { bedroompic2, bedroompic3, bedroompic4 };
const unsigned char *progiconsSun[] = { infosun1, infosun2, infosun3 };
const unsigned char *progiconsCold[] = { infocold1, infocold2, infocold3 };
const unsigned char *progiconsHot[] = { infohot1, infohot2, infohot3 };
const unsigned char *progiconsFreeze[] = { infofreeze1, infofreeze2, infofreeze3 };

const unsigned char *roomStatic[] = { bedroompic, coin};
const unsigned char *winloseframes [] = {winner,loser};
const unsigned char *coinflipanimation[]={coin1,coin2,coin3,coin4,coin5,coin6,coin7,coin8,coin9,coin10,coin11,coin12,coin13,coin14,coin15,coin17,coin18};
const unsigned framestime[] = { 10, 10, 10, 10, 10, 10, 10, 10, 30, 10, 10, 10, 10, 10, 10, 10, 5000,
                    10, 10, 10, 10, 10, 10, 10, 10, 30, 10, 10, 10, 10, 10, 10, 10, 5000,
                    10, 10, 10, 10, 10, 10, 10, 10, 30, 10, 10, 10, 10, 10, 10, 10, 5000,
                    15, 15, 15, 15, 3000, 15, 15, 15, 15 };
const unsigned framestime3[] = { 10, 10, 10, 10, 10, 10, 2000, 5000, 10, 10, 10, 10, 10, 10 };
const unsigned int framestime4[] = { 20, 20, 20, 20, 50, 200, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 200, 100, 80, 50, 40, 30, 20, 20 };
const unsigned int framestime5[] = { 4000, 200, 200, 200 };
const unsigned int framestime7[] = {50,50,50,50,50,50,50,50,50,50,50,50,50,50,50,50,50,50,50,50,500};

const int totalFrames = sizeof(frames) / sizeof(frames[0]);
const int totalFrames3 = sizeof(frames3) / sizeof(frames3[0]);
```

```
const int totalFrames4 = sizeof(frames4) / sizeof(frames4[0]);
const int totalFrames5 = sizeof(frames5) / sizeof(frames5[0]);
const int totalFrames6 = sizeof(bedrooms) / sizeof(bedrooms[0]);
const int totalFrames7 = sizeof(coinflipanimation) / sizeof(coinflipanimation[0]);

unsigned long previousTime = 0;
unsigned long previousTime2 = 0;
unsigned long previousTime3 = 0;
unsigned long previousTime4 = 0;
unsigned long previousTime5 = 0;
unsigned long previousTime6 = 0;
unsigned long previousTime7 = 0;

int currentFrames = 0;
int currentFrames2 = 0;
int currentFrames3 = 0;
int currentFrames4 = 0;
int currentFrames5 = 0;
int currentFrames6 = 0;
int currentFrames7 = 0;
```

Fișierul images.h conține definițiile efective ale imaginilor bitmap utilizate de Void pentru animații și interfață grafică. Fiecare imagine este declarată sub forma unui array de tip const unsigned char, stocat în memoria flash (PROGMEM) pentru a optimiza utilizarea RAM-ului. Aceste imagini reprezintă cadrele individuale ale animațiilor, fiind afișate secvențial pentru a crea iluzia de mișcare. Pentru a crea imaginile necesare afișării pe ecranul OLED, acestea au fost desenate manual folosind site-ul PixilArt [19], am început prin utilizarea unui script scris în Python care convertea fișierele grafice în format bitmap compatibil cu limbajul C. Ulterior, am trecut la utilizarea programului LCD Assistant [20], deoarece era mai rapid și mai convenabil pentru generarea bitmap-urilor, simplificând procesul de pregătire a imaginilor necesare proiectului. Procesul folosind LCD Assistant este la fel, doar ca după realizarea imaginilor a fost necesară convertirea manuală in fișier .bmp monocromatic folosind Paint [21].

```
from PIL import Image


def image_to_bitmap(image_path):
    img = Image.open(image_path).convert("1")
    img = img.resize((128, 64))

    pixels = img.getdata()
    width, height = img.size

    bitmap = []
    for y in range(height):
        row = 0
        for x in range(width):
            pixel = pixels[y * width + x]
            row = (row << 1) | (0 if pixel == 0 else 1)
            if (x + 1) % 8 == 0:
                bitmap.append(row)
                row = 0
        if width % 8 != 0:
```

```
            row <<= 8 - (width % 8)
            bitmap.append(row)

    formatted_bitmap = ", ".join(f"0x{byte:02X}" for byte in bitmap)
    c_array = f"const unsigned char bitmap[] = {{\n{formatted_bitmap}\n}};"

    return c_array


image_path = "input_image.png"
bitmap_code = image_to_bitmap(image_path)

with open("output_bitmap.h", "w") as f:
    f.write(bitmap_code)

print("Bitmap generated and saved to output_bitmap.h")
```

De exemplu, happyidle1[] este primul cadru din animația de stare IDLE. Structura fiecărei imagini constă într-o serie de valori hexazecimale care definesc punctele albe și negre ale bitmap-ului afișat pe ecranul OLED.

Utilizarea PROGMEM este esențială în microcontrolerele cu resurse limitate, deoarece permite salvarea graficii în memoria de program, eliberând astfel memoria RAM pentru alte operații. Acest mod de organizare a resurselor grafice permite o încărcare rapidă a cadrelor în timpul animațiilor, fără să încetinească funcționarea generală a sistemului.

Fișierul images.h lucrează împreună cu icons.h, unde aceste imagini sunt grupate în array-uri și temporizate

```
const unsigned char happyidle1[] PROGMEM = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
, // …. //  0xFF, 0xFF, 0xFF, 0x00, 0x3F, 0xFF, 0xFF, 0xE0, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x01, 0xFF, 0xFF, 0xFF, 0x00, 0x3F, 0xFF, 0xFF, 0xE0, 0x00, 0x00,
0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFE, 0x00, 0x1F, 0xFF, 0xFF, 0xC0, 0x00, 0x00,
0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFE, 0x00, 0x1F, 0xFF, 0xFF, 0xC0, 0x00, 0x00,
0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x7F, 0xFF, 0xFC, 0x00, 0x0F, 0xFF, 0xFF, 0x80, 0x00, 0x00,
0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x1F, 0xFF, 0xF0, 0x00, 0x03, 0xFF, 0xFE, 0x00, 0x00, 0x00,
0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
// …... //  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00};
```

Arhitectura Hardware

După prezentarea arhitecturii software și a organizării interne a codului, este important să detaliem și arhitectura hardware a proiectului, pentru a înțelege modul în care componentele fizice interacționează cu logica implementată.

Pentru a susține funcționalitatea robotului Void, am utilizat o combinație atent aleasă de componente electronice integrate într-o arhitectură hardware compactă și eficientă. În centrul proiectului se află un microcontroller ESP32 de la IdeaSpark [Fig 9]. Alegerea acestei plăcuțe a fost una intenționată, bazată pe nevoia de performanță superioară față de ceea ce poate oferi un microcontroller de tip Arduino clasic (precum Uno sau Nano).

În situația mea, este esențial ca animațiile de pe ecranul să ruleze fluent, fără întârzieri vizibile. Acest lucru presupune nu doar afișarea rapidă a bitmap-urilor, ci și rularea în paralel a altor procese, cum ar fi citirea butoanelor, actualizarea stărilor interne, controlul servo-motoarelor și prelucrarea valorilor de temperatură. Un Arduino, având o frecvență de 16 MHz și o arhitectură mai limitată, ar fi devenit rapid o problemă, mai ales când vine vorba de redarea unor secvențe compuse din mai multe cadre. ESP32, în schimb, vine cu un procesor dual-core tactat la 240 MHz, memorie RAM suficientă pentru a stoca și procesa rapid imaginile bitmap, dar și posibilitatea de a gestiona mai multe taskuri aproape simultan. De asemenea, biblioteca Adafruit_SSD1306 și metodele de desenare drawBitmap() pot fi apelate la frecvențe mari fără ca plăcuța să se blocheze sau să apară efecte vizuale deranjante. Pentru un robot care se bazează pe expresivitate și reacții rapide la input-ul utilizatorului, acest aspect e critic.

În plus, plăcuța oferă un layout accesibil pentru prototipare, dimensiuni reduse și conectivitate wireless inclusă, ceea ce îl face o alegere versatilă pentru proiecte interactive ca acesta. Chiar dacă în acest stadiu nu folosesc conexiunile BLE sau Wi-Fi, prezența lor lasă loc pentru extensii ulterioare fără să schimb hardware-ul.

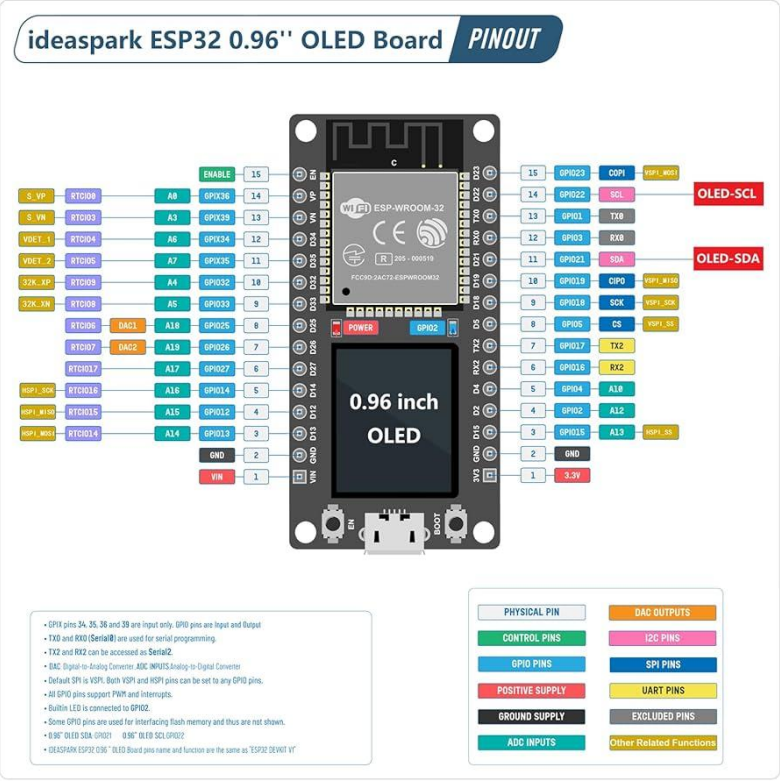

Fig 9.

Pentru monitorizarea temperaturii, am folosit un senzor DHT11, ales datorită simplității și dimensiunii reduse. Precizia sa este suficientă pentru nevoia proiectului, unde valorile exacte nu sunt esențiale, furnizând temperatura o dată la câteva secunde (de obicei 1Hz), ceea ce este mai mult decât suficient pentru declanșarea unor schimbări de stare ocazionale. Un alt avantaj al DHT11 este că necesită doar un singur pin digital pentru comunicare, lăsând astfel liberi alți pini de pe ESP32 pentru butoane, servo-uri sau alte extensii. Deși nu este cel mai precis sensor are o abatere de ±2°C și o plajă limitată (0–50°C). Este stabil, nu fluctuează inutil și e mai mult decât potrivit pentru genul de reacții pe care le vreau de la Void. În plus, consumul redus de curent și biblioteca foarte accesibilă îl fac ideal pentru prototipuri rapide și robuste.

Dacă aș fi avut nevoie de o măsurare mai frecventă, mai precisă sau cu detecție de umiditate mai fiabilă, aș fi ales DHT22 sau un senzor mai avansat precum BME280. Dar pentru scopul de a adăuga un strat „emoțional” legat de mediu , DHT11 face exact cât trebuie, fără complicații.

Deoarece senzorul funcționează la 5V și plăcuța acceptă doar 3.3V pe intrările sale, am adăugat un divizor de tensiune format din două rezistențe de 1kΩ și 2kΩ, asigurând astfel o scădere sigură a tensiunii de semnal. De asemenea, pentru stabilitatea semnalului de date, am utilizat un rezistor pull-up de 10kΩ. Întregul ansamblu compus din senzorul DHT11, rezistențele divizorului de tensiune și sursa de alimentare a fost montată pe un protoboard, lipit cu fludor, și introdus în carcasa robotului pentru protecție și organizare optimă.

Sistemul de mișcare al lui Void este realizat cu ajutorul a patru micro-servomotoare SG90 de 9g de la Tower Pro. Acestea sunt conectate la un modul de alimentare dedicat tip breadboard power supply, care furnizează curentul necesar fără a suprasolicita microcontroller-ul, având un consum de curent care variază semnificativ în funcție de starea de funcționare. În repaus, atunci când mențin o poziție fără sarcină semnificativă, consumă aproximativ 10–15 mA fiecare, deci patru astfel de servomotoare ar avea un consum total de aproximativ 40–60 mA. În timpul mișcării, consumul crește considerabil, ajungând la valori cuprinse între 100 și 250 mA per servo, în funcție de viteză și sarcina aplicată. Astfel, patru servomotoare aflate simultan în mișcare pot necesita între 400 și 1000 mA în total. Motoarele sunt responsabile de animarea membrelor robotului în funcție de starea sa actuală.

Pentru interacțiunea utilizatorului cu robotul, am montat trei butoane fizice cu capace de culori diferite, conectate la pinii de intrare. Aceste butoane sunt utilizate pentru navigarea prin meniuri, confirmarea acțiunilor și resetarea interacțiunilor din cadrul aplicației robotului. În continuare este prezentată diagrama circuitului electric utilizat pentru asamblarea componentelor hardware ale robotului Void. Aceasta ilustrează modul în care fiecare element (microcontroller, senzori, servomotoare, butoane și sursa de alimentare) este interconectat pentru a asigura funcționarea corectă a întregului sistem [Fig 10].

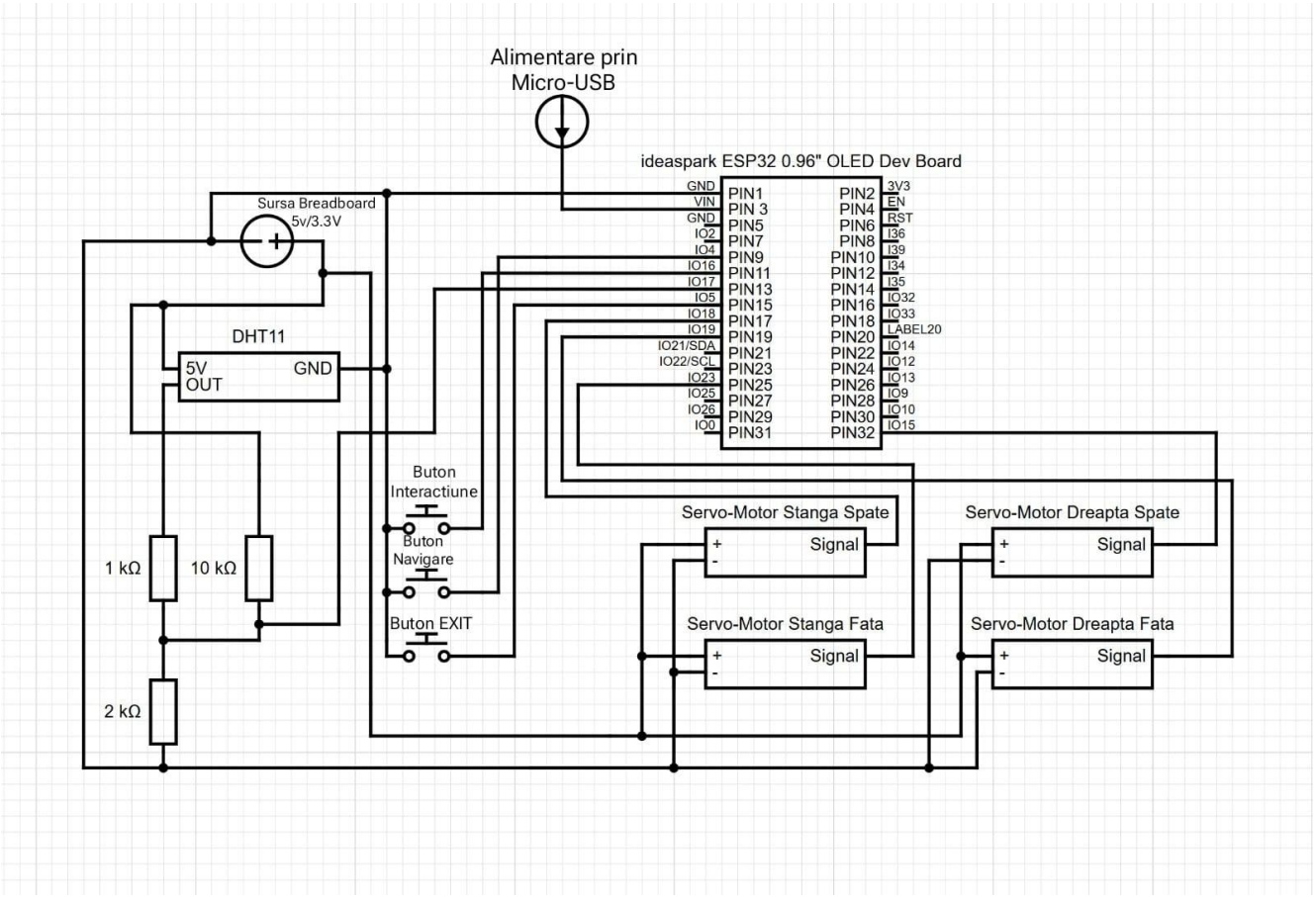

Fig 10.

Pentru a evidenția conexiunile logice dintre modulele principale ale sistemului, mai jos este prezentată o schemă bloc [Fig 11] care reflectă arhitectura hardware. ESP32-ul (model WROOM-32) acționează ca unitate centrală de control, interfațând cu senzorul de temperatură, ecranul OLED, butoanele de intrare și cele patru servomotoare

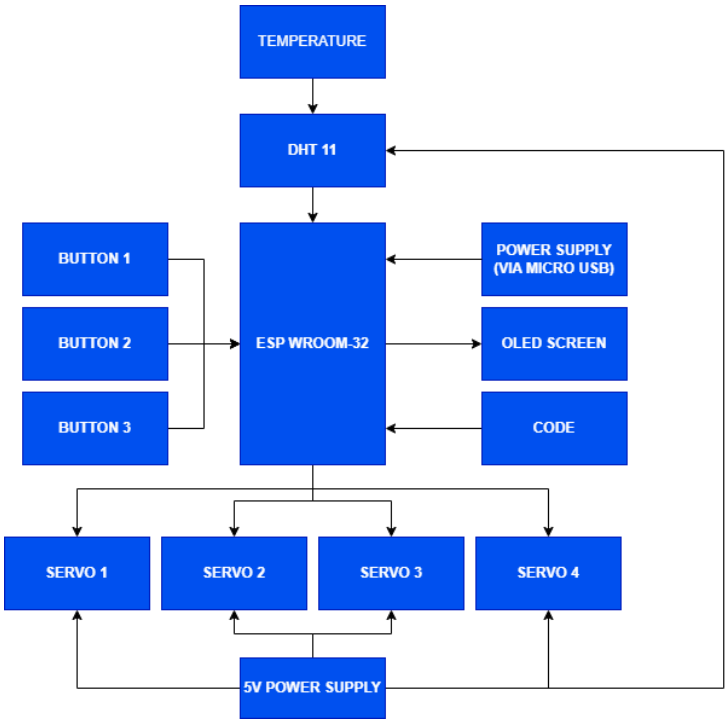

Fig. 11

Proiectarea carcasei

Pentru creerea locului unde urma să fie montat microcontrollerul, am fotografiat placa și am importat imaginile în Blender, unde am creat o plăcuță de gabarit. Aceasta a fost modificată și refăcută de mai multe ori, pe baza măsurătorilor directe, până când microcontrollerul a încăput perfect. Aceasta prezintă pe partea de sus 2 “urechi” de prindere pentru diverse accesorii de personalizare (pălării, antene, urechi).

Restul corpului robotului a fost modelat în Autodesk Inventor, folosind modelare solidă parametrică, pentru a asigura precizia dimensională. În interiorul carcasei am proiectat două brațe mici, asemănătoare unor cleme, care fixează protoboardul. Ulterior, ansamblul a fost importat în Blender pentru a realiza unificarea componentelor într-un singur corp și pentru a integra elemente estetice [Fig 12].

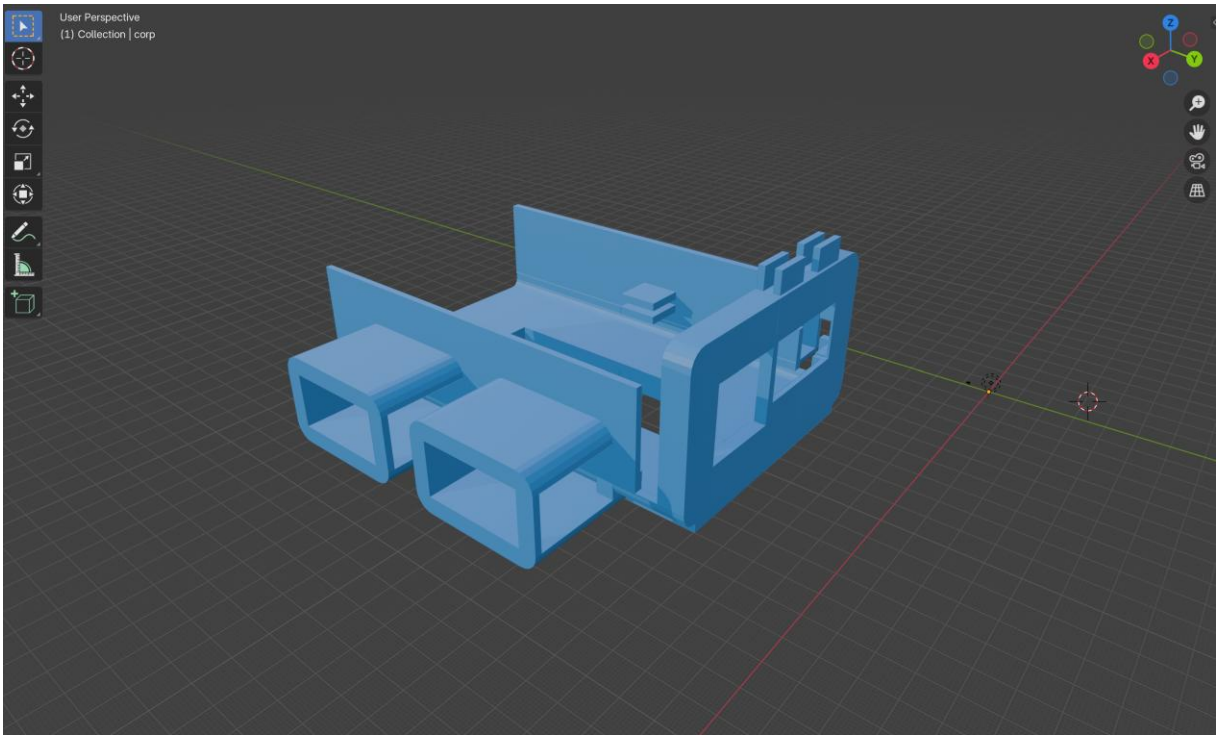

Fig 12.

În procesul de trecere din Inventor în Blender, geometria solidă a fost convertită într-o rețea de triunghiuri (mesh). Această conversie, specifică mediilor de grafică 3D, implică pierderea informațiilor exacte despre suprafețe și dimensiuni, ceea ce face dificilă realizarea unui desen tehnic tradițional detaliat. Din acest motiv, în cadrul documentației am optat pentru prezentarea dimensiunilor generale relevante ale carcasei și o descriere clară a poziționării componentelor, adaptată specificului proiectului [Fig 13].

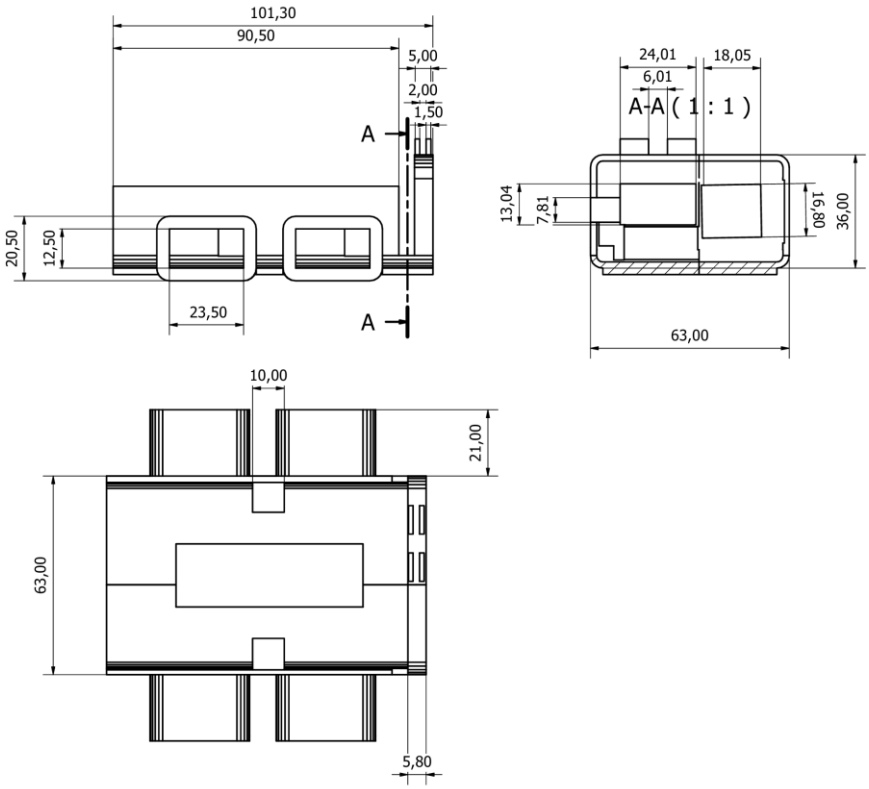

Fig 13.

Întregul ansamblu este acoperit de o carcasă exterioară, realizată de asemenea în Blender, pentru a putea vizualiza mai bine potrivirea tuturor elementelor înainte de imprimare. Pentru a accentua tema spațială a robotului Void, am adăugat decorațiuni precum stele și o planetă Saturn, modelate manual în Blender [Fig 14]. Aceasta a fost ulterior lipită de corpul principal al robotului cu ajutorul unui adeziv special pentru mase plastice, asigurând astfel o fixare durabilă și estetică.

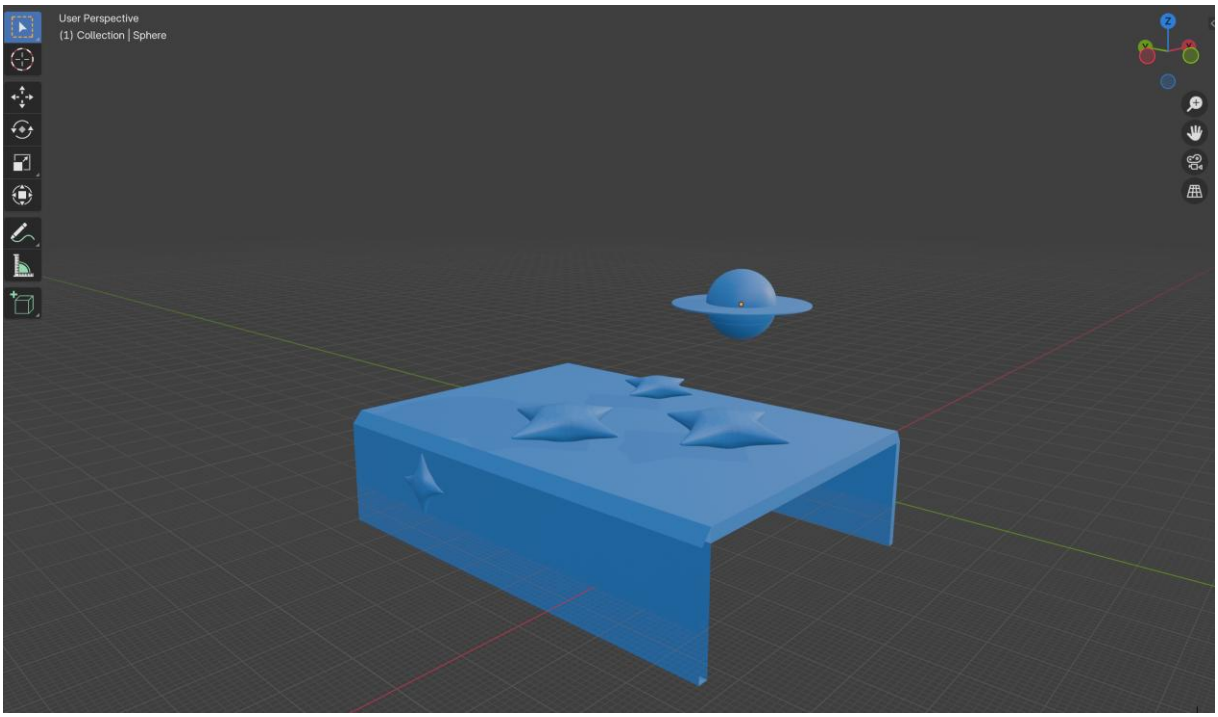

Fig 14.

Pentru picioare am ales să merg pe design-ul clasic dar perfect al brațelor servo-motoarelor. Acesta a fost modificat încât să fie mai gros si mai mare, pentru a-l face pe robotul meu sa nu atingă pământul cand este în poziția IDLE [Fig 15].

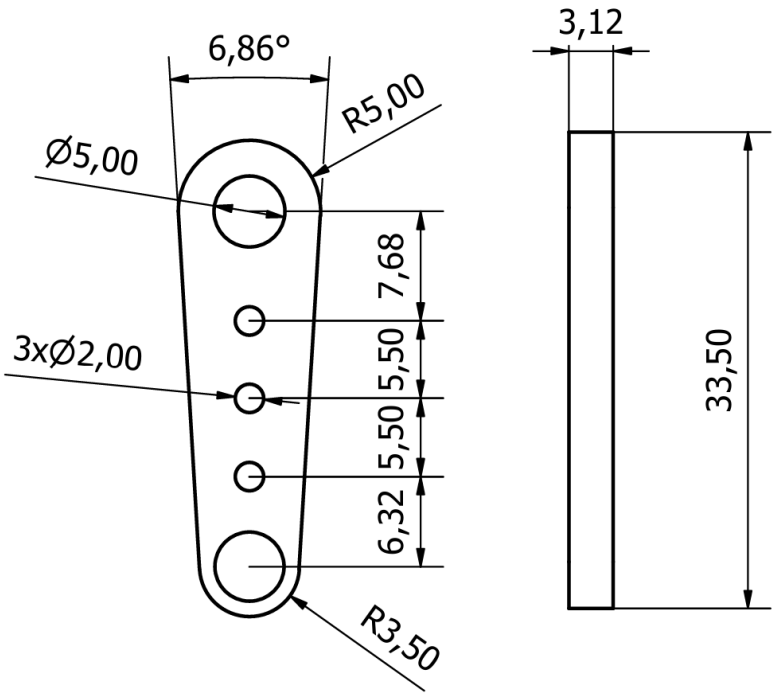

Fig 15.

Carcasa pentru controller-ul robotului a reprezentat pasul final, înlocuind breadboard-ul care fusese folosit inițial pentru fixarea butoanelor și legând astfel întregul proiect într-o formă compactă și funcțională. Este alcătuită din două componente principale: partea superioară [Fig 16], care conține alezajele pentru montarea celor trei butoane, distanțate la 14 mm între centre. Această distanță a fost aleasă astfel încât, odată cu montarea capacelelor, utilizatorul să le poată apăsa confortabil, fără ca acestea să se atingă între ele sau să genereze disconfort în utilizare.

Partea inferioară [Fig 17] acționează ca un capac care închide ansamblul și include o deschidere laterală destinată trecerii firelor, pentru a permite un transfer curat al semnalelor fără tensionarea cablurilor. Pentru a îmbunătăți rezistența mecanică a carcasei în zona de apăsare, am adăugat două nervuri interne cu secțiune triunghiulară, plasate strategic, care contribuie la rigidizarea ansamblului în cazul unor apăsări accidentale sau forțări nedorite.

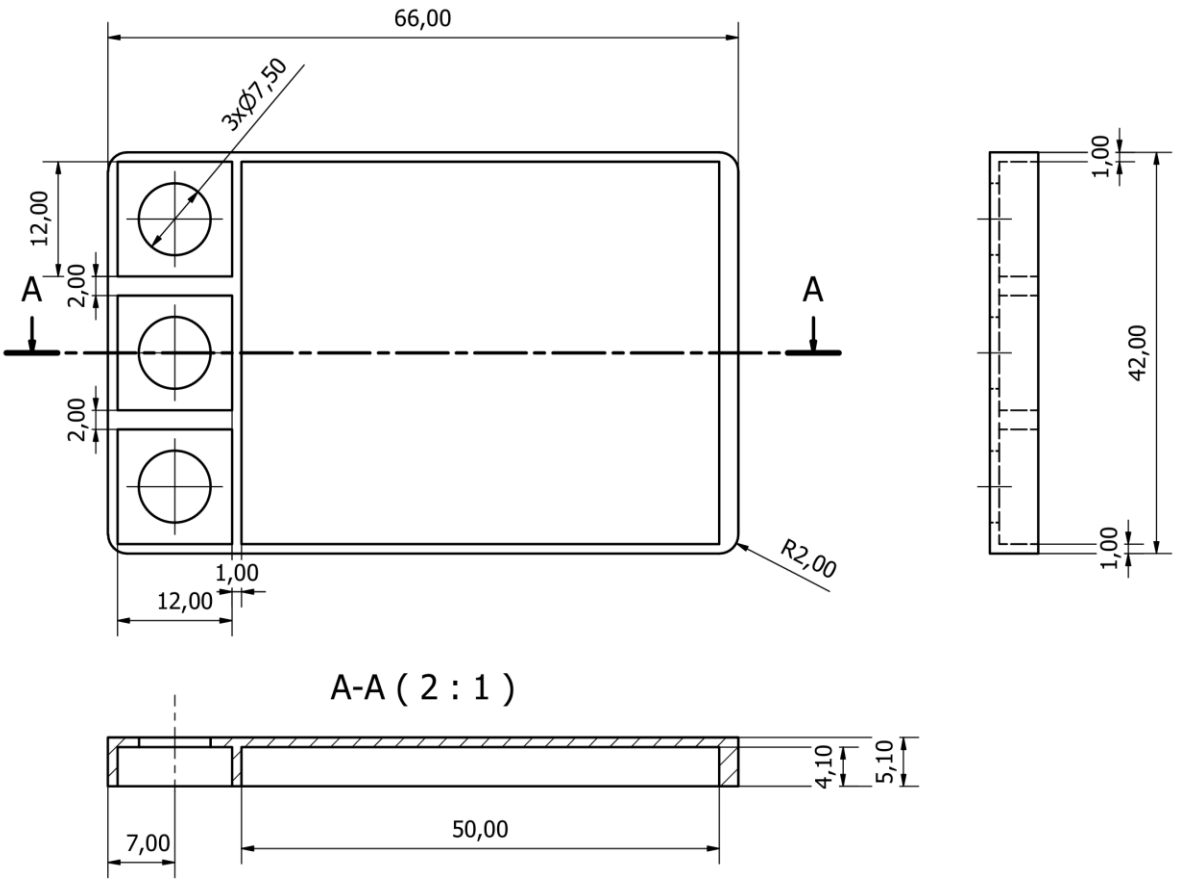
66,00
3xØ7,50
12,00
2,00
2,00
A
A
1,00
12,00
R2,00
1,00
42,00
1,00
A-A (2 : 1)
4,10
5,10
7,00
50,00

Fig 16.

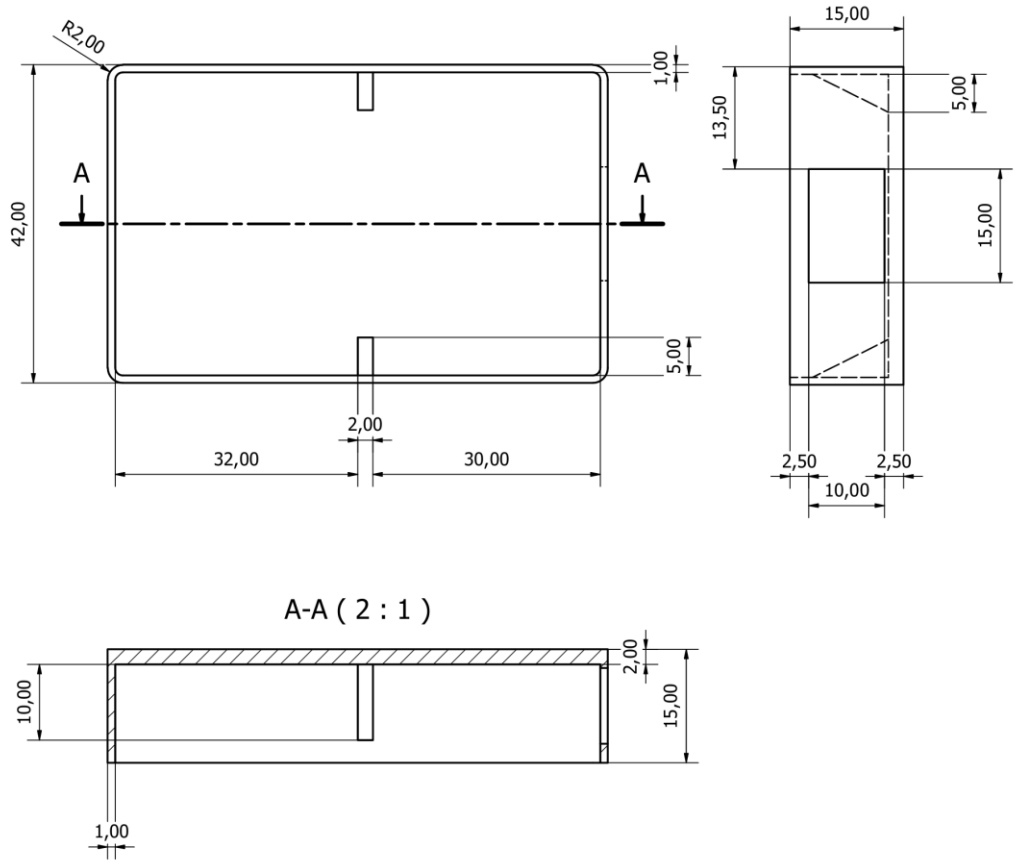
R2,00
1,00
A
A
42,00
5,00
2,00
32,00
30,00
15,00
13,50
5,00
15,00
2,50
2,50
10,00
A-A (2 : 1)
2,00
10,00
15,00
1,00

Fig 17.

Mai jos este prezentat ansamblul carcasei controlerului [Fig 18], compus din cele două părți principale: baza cu locașurile pentru butoane și capacul prevăzut cu nervuri de rigidizare și deschidere pentru trecerea firelor.

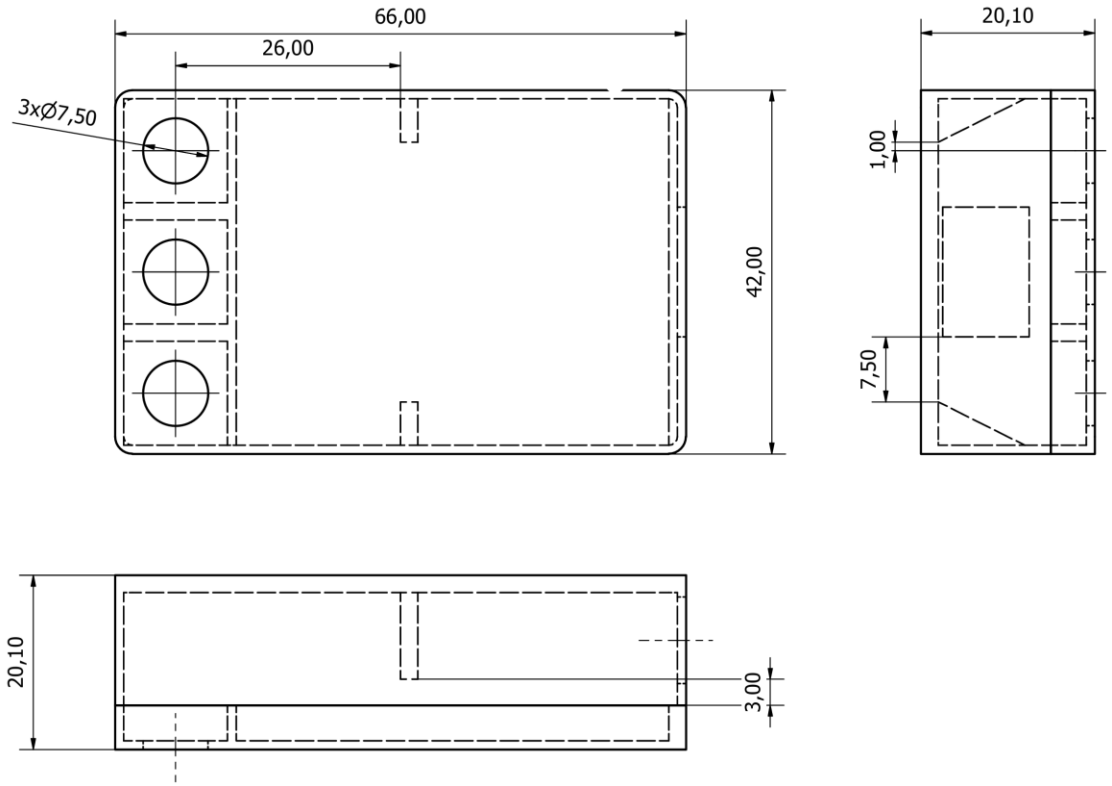

Fig 18.

Îmbunătățiri pentru viitor

Pe viitor, mi-aș dori să îmbunătățesc proiectul prin câteva modificări pe care eu le consider destul de importante:

În perspectiva îmbunătățirii sistemului hardware, am dezvoltat o nouă schemă electrică [Fig 19], care propune o sursă de alimentare unificată și mai eficientă. În locul soluției inițiale bazate pe un modul breadboard pentru servo-motoare și o alimentare separată pentru microcontroller, am proiectat un circuit care să alimenteze atât microcontrollerul ESP32, cât și servomotoarele, dintr-o singură sursă principală de 30V provenită de la un transformator conectat la rețeaua de 230V. Tensiunea de 30V este redresată printr-o punte de diode și stabilizată inițial cu un condensator electrolitic. Pentru a reduce încărcarea termică a regulatoarelor de tensiune și pentru a crește eficiența energetică, am ales o conversie în două trepte: prima etapă scade tensiunea la 24V folosind un regulator LM7824, urmată de o reducere suplimentară la 5V cu ajutorul unui LM7805. Separat, un LM317 configurat cu doi rezistori generează o tensiune de 3.3V necesară ESP32-ului.

Această soluție nu modifică funcționarea de bază a circuitului anterior, însă optimizează alimentarea componentelor prin utilizarea unei singure surse, reducând complexitatea cablajului și îmbunătățind stabilitatea sistemului. De asemenea, această configurație este gândită pentru a putea fi ulterior integrată într-un PCB dedicat, eliminând protoboardul.

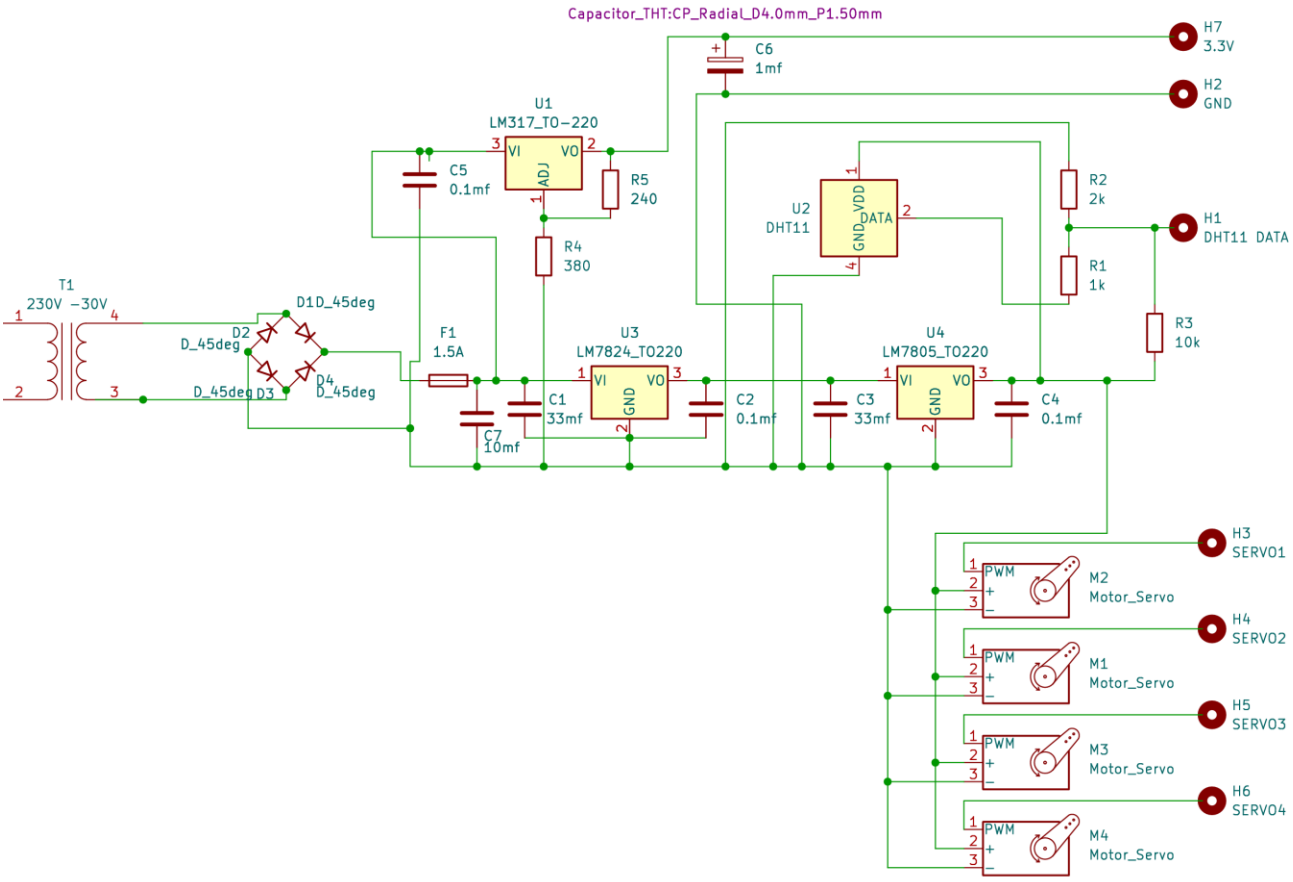

Fig 19.

O direcție de dezvoltare pe care o mai am în vedere este valorificarea conectivității de tip BLE și Wi-Fi, deja integrate în microcontroller. Deși în forma actuală robotul funcționează fără dependență de alte dispositive (exceptând alimentarea prin micro-USB), consider că integrarea cu o aplicație ar putea aduce un plus de interactivitate. Nu visez la o platformă complexă sau o mutare a interacțiunii pe telefon, ci mai degrabă o extensie simplă, prin care utilizatorul să primească alerte (de exemplu, atunci când robotul are „nevoie" de ceva) sau să poată transmite comenzi de bază, fără a renunța la caracterul local și fizic al interacțiunii. Ideea de bază rămâne ca robotul să fie prezent, pe birou, vizibil și la îndemână, iar conectivitatea să fie doar un adaos funcțional, nu o mutare a experienței în digital.

În final, doresc să refac și designul exterior al carcasei, atât din punct de vedere funcțional (poziționarea componentelor, accesibilitatea porturilor), cât și estetic, pentru a face robotul mai compact, mai robust și mai atractiv vizual.

Testare

Testarea proiectului a fost o etapă esențială, dar și îndelungată, prin care am urmărit verificarea funcționării corecte a fiecărei componente și a integrării lor în sistem. Am început prin testarea individuală a sursei de alimentare, măsurând cu un multimetru tensiunile obținute și asigurându-mă că ESP32-ul primește 3.3V, iar servo-motoarele și senzorii beneficiază de 5V, exact cum era necesar. Senzorul DHT11 a fost verificat în mod repetat în diferite condiții de temperatură pentru a confirma exactitatea citirilor, robotul fiind scos pe balcon în zile calde, dar și în serile mai răcoroase. Din păcate, nu am avut o cale bună de a îl testa și la temperaturi mai extreme (sub 10 grade sau peste 25), dar rezultatul nu ar trebui să difere. De asemenea, fiecare

servomotor a fost testat pentru a observa răspunsul la comenzile de mișcare, urmărind eventualele blocaje sau pierderi de poziție. Acestea au fost supuse unor mișcări continue, cât și la unghiuri care nu sunt naturale pentru mișcările lui Void.

Pe parcursul testării, am folosit intens afișarea în Serial Monitor pentru a urmări valorile interne ale variabilelor critice, precum pagina curentă, modul activ, starea butoanelor sau confirmările de selecție. Aceste tehnici m-au ajutat să identific mult mai rapid locurile unde apăreau erori logice sau neconcordanțe între starea așteptată și comportamentul real al sistemului.

În continuare, am testat și butoanele de navigare, confirmând că acestea funcționează corect cu logica de debounce implementată, iar apăsările sunt detectate stabil. Navigarea prin meniuri și afișarea animațiilor pe ecran au fost verificate parcurgând manual toate taburile disponibile. Am ajuns să observ un comportament neobișnuit al butonului de EXIT, acesta având un timp de răspundere mai lent decât celelalte două butoane. Din păcate, nu am reușit să găsesc rădăcina problemei, rescriind și programul destinat acelui buton, cât și înlocuind fizic butonul și jumper-ul care făcea conexiunea. Navigarea prin interfață s-a dovedit a fi rapidă și fluentă, fără întârzieri sau blocaje. Pentru a testa comportamentul general, robotul a fost trecut prin toate stările definite, de la IDLE și CUTE, până la ANGRY și SLEEP, validând tranzițiile și reacțiile programate. De asemenea, modulul coinflip a fost testat pentru a vedea dacă logica de câștig și pierdere funcționează corect, iar shop-ul a fost verificat pentru procesul de achiziționare a produselor.

În timpul testării extinse am descoperit mai multe probleme majore, dintre care cea mai grea a fost legată de modul în care interfața afișa pagina 0 (animația de bază). După un anumit număr de treceri prin meniuri, animatia tabului 0 dispărea de pe ecran, deși serial monitorul arăta că robotul era încă pe pagina corectă. După mai multe investigații, am descoperit că o variabilă static bool folosită în funcția de desenare nu se reseta cum trebuie între comutările de pagini, ceea ce bloca desenarea corectă a imaginii. Acest lucru a rezolvat și problema butonului de EXIT care acum funcționeaza perfect, variabila „currentFrame” fiind cauza problemei.

```
static bool wasOnPage0 = false;
 if (page == 0) {
   if (!wasOnPage0) {
     currentFrames = 0;
     wasOnPage0 = true;
   }
   if (currentMode == IDLE)
     idle();
   else if (currentMode == CUTE)
     cute(healthpoints);
   else if (currentMode == ANGRY)
     angry(healthpoints);
   else if (currentMode == SLEEP)
     sleepy(sleeppoints);
   else if (currentMode == PLAY)
     idleplay(playpoints);
 }
```

Problema a fost rezolvată prin resetarea acestei variabile de control ori de câte ori se schimba pagina sau modul.

O altă dificultate a fost implementarea sistemului de echipare și dezechipare a itemelor în meniul Bedroom. La început, alegerea unui item funcționa, însă dezechiparea era problematică: robotul nu revenea corect la starea IDLE, iar meniul devenea incoerent. De asemenea, selectarea anumitor iteme (precum Book) cauza dispariția textului din meniul Bedroom sau selectarea automată a unor opțiuni fără interacțiune clară din partea utilizatorului. Pentru rezolvarea acestor probleme, am restructurat complet funcția de Bedroom, separând clar logica pentru modurile IDLE, SLEEP și PLAY și rescriind tranzițiile dintre acestea. De asemenea, am implementat resetări stricte ale variabilelor infoBool, pickingItem, itemPage și infoPage imediat după alegerea sau renunțarea la un item.

În final, după multe testări și ajustări, toate aceste probleme au fost rezolvate, iar robotul funcționează stabil, cu toate animațiile, meniurile și tranzițiile între stări corect implementate.

Concluzie

Void a reprezentat pentru mine mai mult decât o simplă combinație de senzori, servo-motoare și linii de cod, a fost o ocazie să învăț, să greșesc, să refac și să construiesc ceva care să prindă viață la propriu și la figurat. De la primele încercări stângace până la momentele în care robotul a clipit pentru prima dată pe ecran, fiecare pas a adus cu el provocări tehnice și decizii de design care m-au împins să găsesc soluții cât mai creative. Fie că a fost vorba despre controlul mișcărilor, despre optimizarea consumului energetic sau despre alegerea materialelor sau a design-ului potrivit pentru carcasă, fiecare detaliu a contribuit la formarea unui întreg coerent. Pe parcurs am învățat cât de importantă este răbdarea în fața erorilor, cât de valoroasă este testarea și câtă bucurie poate aduce un proiect în care pui suflet. Void nu e doar un robot, ci și dovada că pasiunea, perseverența și dorința de a înțelege mai mult pot transforma o idee simplă într-o experiență reală și vie.

Bibliografie

1. https://en.wikipedia.org/wiki/Rubber_duck_debugging
2. https://en.wikipedia.org/wiki/Tamagotchi
3. https://blog.janbox.com/what-is-a-tamagotchi/
4. https://members.tripod.com/~Tamagotchi_Central/instructions.html
5. https://www.makeuseof.com/eilik-cozmo-vector-best-emotional-robots/
6. https://www.wired.com/2016/06/anki-cozmo-ai-robot-toy/
7. https://en.wikipedia.org/wiki/Scratch_(programming_language)

8. https://ankicozmorobot.com/
9. https://time.com/4531323/anki-cozmo-robot-review-2016/
10. https://www.techradar.com/reviews/anki-vector
11. https://anki.bot/products/vector-robot
12. https://energizelab.com/consumerview/eilik
13. https://energizelab.com/blogview/update055
14. https://en.wikipedia.org/wiki/Pou_(video_game)
15. https://pou.fandom.com/wiki/Game_Room
16. https://pou.fandom.com/wiki/Shop
17. https://pou.fandom.com/wiki/Status
18. https://www.questgamefi.com/game/Pou.html
19. pixilart.com/draw
20. https://en.radzio.dxp.pl/bitmap_converter/
21. https://en.wikipedia.org/wiki/Microsoft_Paint

Anexe

Fișierul images.h nu este atașat deoarece acesta este foarte mare datorită dimensiunilor vectorilor de bitmapuri. Poate fii vizualizat în repertoriul meu pe github, alături de alte poze și detalii.

https://github.com/tiberiutudorascu/Void

Fișierul void.ino

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <ESP32Servo.h>
#include "images.h"
#include "DHT.h"
#include "icons.h"

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
#define SCREEN_ADDRESS 0x3C
#define DHTPIN 17
#define DHTTYPE DHT11

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RESET);
DHT dht(DHTPIN, DHTTYPE);

int page = 0;
int infoPage = 0;

unsigned long lastButtonPress1 = 0;
unsigned long lastButtonPress2 = 0;
unsigned long lastButtonPress3 = 0;
const unsigned long debounceDelay = 200;
unsigned long lastTempCheck = 0;
int buttonPressCount = 0;
int buttonPressCount2 = 0;

bool apasat = false;
bool apasat2 = false;
bool apasat3 = false;
enum Mode
{
   IDLE,
   CUTE,
   ANGRY,
   SLEEP,
   PLAY,
   HUNGRY,
};
Mode currentMode = IDLE; // idle e activ initial
```

```
Mode lastMode = IDLE;

bool infoBool = false;
bool infoConfirm = false;

int totalPages = 5;
int totalInfoPages = 5;

float temperature;
int healthpoints = 0;
float angrypoints = 0;
int sleeppoints = 0;
int playpoints = 0;
int hungerpoints = 0;

int AngryTarget;

unsigned long sleepTime = 0;

unsigned long HPtime = 0;

Servo stangaS;
int PositionSS = 90;
int currentAngleSS;
Servo dreaptaF;
int PositionDF = 0;
int currentAngleDF;
Servo dreaptaS;
int PositionDS = 0;
int currentAngleDS;
Servo stangaF;
int PositionSF = 90;
int currentAngleSF;
int lastMoveTimeSS = 0;
int lastMoveTimeDF = 0;
int lastMoveTimeDS = 0;
int lastMoveTimeSF = 0;
int targetAngles1[] = {45, 0, 45, 0, 45, 0, 45, 0};

int money = 2000;
enum CoinflipState
{
  COINFLIP_IDLE,
  COINFLIP_FLIPPING,
  COINFLIP_SHOW_RESULT
};

CoinflipState coinState = COINFLIP_IDLE;
unsigned long coinStartTime = 0;
int winlose = 0;
bool coinInProgress = false;
```

```
int food = 4, drinks = 40;
bool haveTV = false, tvON = false;
bool haveCards = false, cardsON = false;
bool haveBook = false, bookON = false;
bool haveRadio = false, radioON = false;
int itemPage = 0;
bool pickingItem = false;

void setup()
{
   Serial.begin(9600);
   dht.begin();
   if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS))
   {
      Serial.println(F("Eroare la initializarea OLED!"));
      for (;;)
         ;
   }

   stangaS.attach(18);
   dreaptaF.attach(19);
   dreaptaS.attach(15);
   stangaF.attach(23);

   stangaS.write(PositionSS);
   currentAngleSS = PositionSS;
   dreaptaF.write(PositionDF);
   currentAngleDF = PositionDF;
   dreaptaS.write(PositionDS);
   currentAngleDS = PositionDS;
   stangaF.write(PositionSF);
   currentAngleSF = PositionSF;

   pinMode(4, INPUT_PULLUP);
   pinMode(16, INPUT_PULLUP);
   pinMode(5, INPUT_PULLUP);
   display.clearDisplay();
   display.display();
}
void servoMovement(Servo &s, int targetAngle, int &currentAngle, int &lastMoveTime, int
step = 5, unsigned long stepDelay = 10)
{
   unsigned long currentTime7 = millis();
   if (currentTime7 - lastMoveTime >= stepDelay)
   {
      lastMoveTime = currentTime7;
      if (currentAngle < targetAngle)
      {
         currentAngle += step;
         if (currentAngle > targetAngle)
            currentAngle = targetAngle;
         s.write(currentAngle);
```

```
        }
        else if (currentAngle > targetAngle)
        {
            currentAngle -= step;
            if (currentAngle < targetAngle)
                currentAngle = targetAngle;
            s.write(currentAngle);
        }
    }
}
void servoMultipleMovement(Servo &s, int targetAngles[], int size, int &currentAngle, int
&lastMoveTime, int step = 5, unsigned long stepDelay = 15)
{
    static int i = 0;
    unsigned long currentTime = millis();

    if (i >= size)
        i = 0;

    if (currentTime - lastMoveTime >= stepDelay)
    {
        lastMoveTime = currentTime;
        int target = targetAngles[i];

        if (currentAngle < target)
        {
            currentAngle += step;
            if (currentAngle > target)
                currentAngle = target;
            s.write(currentAngle);
        }
        else if (currentAngle > target)
        {
            currentAngle -= step;
            if (currentAngle < target)
                currentAngle = target;
            s.write(currentAngle);
        }

        if (currentAngle == target)
        {
            i++;
        }
    }
}

void loop()
{
    unsigned long currentMillis = millis();
    temperature = dht.readTemperature();
    if (currentMillis - HPtime >= 300000)
    {
```

```
      healthpoints--;
      sleeppoints--;
      HPtime = currentMillis;
   }

   if (digitalRead(4) == LOW && apasat && (currentMillis - lastButtonPress1 >
debounceDelay))
   {
      apasat = false;
      lastButtonPress1 = currentMillis;
      if (infoBool)
         infoPage++;
      else
         page++;
      if (page > totalPages)
         page = 0;
      if (page == 1 && infoPage > totalInfoPages)
         infoPage = 0;
      if (pickingItem)
         itemPage++;
      if (itemPage > totalItems)
         itemPage = 0;
   }
   if (digitalRead(4) == HIGH)
      apasat = true;

   if (digitalRead(5) == LOW && apasat3 && (currentMillis - lastButtonPress3 >
debounceDelay))
   {
      apasat3 = false;
      lastButtonPress3 = currentMillis;
      if (infoBool)
      {
         infoBool = false;
         infoConfirm = false;
         pickingItem = false;
         itemPage = 0;
      }
      else
      {
         page = 0;
      }
   }
   if (digitalRead(5) == HIGH)
      apasat3 = true;

   if (digitalRead(16) == LOW && apasat2 && (currentMillis - lastButtonPress2 >
debounceDelay))
   {
      apasat2 = false;
      lastButtonPress2 = currentMillis;
```

```
      if (page != 0)
      {
         if (!infoBool)
         {
            infoBool = true;
            infoConfirm = false;
         }
         else
         {
            infoConfirm = true;
            Serial.println("Confirmare in meniu activata.");
         }
         if (!infoBool)
            previousTime2 = currentMillis;
      }

      else if (page == 0)
      {
         buttonPressCount++;
         Serial.println(buttonPressCount);
         Serial.print("AngryPoints: ");
         Serial.println(angrypoints);
         AngryTarget = temperature >= 19 && temperature <= 27 ? 15 : temperature >= 15
&& temperature <= 18 ? 10
                                                                  : 5;
         if (buttonPressCount == AngryTarget)
         {
            if (currentMode == SLEEP)
            {
               currentMode = ANGRY;
               if (lastMode != currentMode)
                  lastMode = currentMode;
               buttonPressCount = 0;
            }
            else if (angrypoints == 3)
            {
               currentMode = ANGRY;
               if (lastMode != currentMode)
                  lastMode = currentMode;
               buttonPressCount = 0;
            }
            else if (angrypoints < 3)
            {
               currentMode = CUTE;
               if (lastMode != currentMode)
                  lastMode = currentMode;
               angrypoints = angrypoints + 0.5;
               Serial.print("Health: ");
               Serial.println(healthpoints);
               buttonPressCount = 0;
            }
         }
```

```
    }
  }
  if (digitalRead(16) == HIGH)
    apasat2 = true;

  static int lastPage = -1;
  if (page != lastPage)
  {
    Serial.print("Pagina curenta: ");
    Serial.println(page);
    lastPage = page;
  }
  static bool wasOnPage0 = false;
  if (page == 0)
  {
    if (!wasOnPage0)
    {
      currentFrames = 0;
      wasOnPage0 = true;
    }
    if (currentMode == IDLE)
      idle();
    else if (currentMode == CUTE)
      cute(healthpoints);
    else if (currentMode == ANGRY)
      angry(healthpoints);
    else if (currentMode == SLEEP)
      sleepy(sleeppoints);
    else if (currentMode == PLAY)
      idleplay(playpoints);
  }
  else
  {
    wasOnPage0 = false;
  }
  if (page == 1)
  {
    if (infoBool)
      stats();
    else
      progress(healthpoints, sleeppoints, hungerpoints, playpoints);
  }
  else if (page == 2)
  {
    bedroom();
  }
  else if (page == 3)
  {
    kitchen(hungerpoints, drinks, food);
  }
  else if (page == 4)
  {
```

```
      coinflip(money);
   }
   else if (page == 5)
   {
      shop(money, drinks, food, haveCards, haveBook, haveRadio, haveTV);
   }
   if (currentMode == SLEEP)
   {
      servoMovement(stangaS, 0, currentAngleSS, lastMoveTimeSS);
      servoMovement(dreaptaF, 90, currentAngleDF, lastMoveTimeDF);
      servoMovement(dreaptaS, 90, currentAngleDS, lastMoveTimeDS);
      servoMovement(stangaF, 0, currentAngleSF, lastMoveTimeSF);
   }
   if (currentMode == IDLE)
   {
      servoMovement(stangaS, 90, currentAngleSS, lastMoveTimeSS);
      servoMovement(dreaptaF, 0, currentAngleDF, lastMoveTimeDF);
      servoMovement(dreaptaS, 0, currentAngleDS, lastMoveTimeDS);
      servoMovement(stangaF, 90, currentAngleSF, lastMoveTimeSF);
   }
   if (currentMode == CUTE)
   {
      servoMovement(stangaS, 45, currentAngleSS, lastMoveTimeSS);
      servoMovement(dreaptaS, 45, currentAngleDS, lastMoveTimeDS);
      servoMovement(dreaptaF, 0, currentAngleDF, lastMoveTimeDF);
      servoMultipleMovement(stangaF, targetAngles1, 8, currentAngleSF,
lastMoveTimeSF);
   }

   if (currentMillis - lastTempCheck >= 10000)
   {
      Serial.println(temperature);
      lastTempCheck = currentMillis;
   }
}
void idle()
{
   unsigned long currentTime = millis();
   if (currentTime - previousTime >= framestime[currentFrames])
   {
      previousTime = currentTime;
      display.clearDisplay();
      display.drawBitmap(0, 0, frames[currentFrames], 128, 64, SSD1306_WHITE);
      display.display();
      currentFrames++;
      if (currentFrames >= totalFrames)
         currentFrames = 0;
   }
}
void idleplay(int &playpoints)
{
   unsigned long currentTime = millis();
```

```
    if (currentTime - previousTime >= framestime[currentFrames])
    {
      previousTime = currentTime;
      display.clearDisplay();

      display.drawBitmap(0, 0, frames[currentFrames], 128, 64, SSD1306_WHITE);

      if (currentMode == PLAY)
      {
        if (cardsON)
        {
          display.drawBitmap(0, 0, items[0], 128, 64, SSD1306_WHITE);
          playpoints += 2;
        }
        else if (bookON)
        {
          display.drawBitmap(0, 0, items[1], 128, 64, SSD1306_WHITE);
          playpoints += 4;
        }
        else if (radioON)
        {
          display.drawBitmap(0, 0, items[2], 128, 64, SSD1306_WHITE);
          playpoints += 5;
        }
        else if (tvON)
        {
          display.drawBitmap(0, 0, items[3], 128, 64, SSD1306_WHITE);
          playpoints += 8;
        }
      }

      currentFrames++;
      if (currentFrames >= totalFrames)
        currentFrames = 0;

      display.display();
    }
}

void cute(int &health)
{
    unsigned long currentTime3 = millis();

    Serial.print("Frame: ");
    Serial.print(currentFrames3);
    Serial.print(" / ");
    Serial.println(totalFrames3);

    if (currentTime3 - previousTime3 >= framestime3[currentFrames3])
    {
      previousTime3 = currentTime3;
      display.clearDisplay();
```

```
      display.drawBitmap(0, 0, frames3[currentFrames3], 128, 64, SSD1306_WHITE);
      display.display();
      currentFrames3++;
   }

   if (currentFrames3 >= totalFrames3)
   {
      Serial.println("ANIMATIE TERMINATA - +1 HEALTH");
      health += 10;
      Serial.print("HEALTH: ");
      Serial.println(health);
      currentMode = IDLE;
      lastMode = CUTE;
      currentFrames3 = 0;
   }
}

void angry(int &health)
{
   unsigned long currentTime4 = millis();
   if (currentTime4 - previousTime4 >= framestime4[currentFrames4])
   {
      previousTime4 = currentTime4;
      display.clearDisplay();
      display.drawBitmap(0, 0, frames4[currentFrames4], 128, 64, SSD1306_WHITE);
      display.display();
      currentFrames4++;
      if (currentFrames4 >= totalFrames4)
      {
         currentFrames4 = 0;
         angrypoints = 0;
         health -= 2;
         currentMode = IDLE;
         lastMode = ANGRY;
      }
   }
}
void bedroom()
{
   unsigned long currentTime6 = millis();

   if (!pickingItem)
   {
      if (!infoBool)
      {
         if (currentTime6 - previousTime6 >= 300)
         {
            previousTime6 = currentTime6;
            display.clearDisplay();
            display.drawBitmap(0, 0, bedrooms[currentFrames6], 128, 64,
SSD1306_WHITE);
            display.setTextColor(SSD1306_WHITE);
```

```
            display.setTextSize(1);
            display.setCursor(0, 0);
            display.println("Bedroom");

            display.display();
            currentFrames6++;
            if (currentFrames6 >= totalFrames6)
                currentFrames6 = 0;
        }
    }
    else
    {
        display.clearDisplay();
        display.drawBitmap(0, 0, roomStatic[0], 128, 64, SSD1306_WHITE);
        display.setCursor(0, 0);
        display.setTextColor(SSD1306_WHITE);

        if (currentMode == SLEEP)
        {
            if (infoPage == 0)
            {
                display.setTextSize(1.5);
                display.println(">Wake Up");
                display.setTextSize(1);
                display.println("Items");

                if (infoConfirm)
                {
                    currentMode = IDLE;
                    infoConfirm = false;
                }
            }
            else if (infoPage == 1)
            {
                display.setTextSize(1);
                display.println("Wake Up");
                display.setTextSize(1.5);
                display.println(">Pick Item");

                if (infoConfirm)
                {
                    pickingItem = true;
                    itemPage = 0;
                    infoConfirm = false;
                }
            }
        }
        else
        {
            if (infoPage == 0)
            {
                display.setTextSize(1.5);
```

```
            display.println(">Sleep");
            display.setTextSize(1);
            display.println("Items");

            if (infoConfirm)
            {
               currentMode = SLEEP;
               infoConfirm = false;
            }
         }
         else if (infoPage == 1)
         {
            display.setTextSize(1);
            display.println("Sleep");
            display.setTextSize(1.5);
            display.println(">Pick Item");

            if (infoConfirm)
            {
               pickingItem = true;
               itemPage = 0;
               infoConfirm = false;
            }
         }
      }

      if (infoPage > 1)
         infoPage = 0;

      display.display();
   }
}
else
{
   display.clearDisplay();
   display.drawBitmap(0, 0, roomStatic[0], 128, 64, SSD1306_WHITE);
   display.setCursor(0, 0);
   display.setTextColor(SSD1306_WHITE);

   if (itemPage == 0)
      display.println(">Cards");
   else if (itemPage == 1)
      display.println(">Book");
   else if (itemPage == 2)
      display.println(">Radio");
   else if (itemPage == 3)
      display.println(">TV");

   if (infoConfirm)
   {
      if (itemPage == 0 && haveCards)
      {
```

```
            if (cardsON)
            {
               cardsON = false;
               currentMode = IDLE;
            }
            else
            {
               cardsON = true;
               bookON = false;
               radioON = false;
               tvON = false;
               currentMode = PLAY;
            }
         }
         else if (itemPage == 1 && haveBook)
         {
            if (bookON)
            {
               bookON = false;
               currentMode = IDLE;
            }
            else
            {
               cardsON = false;
               bookON = true;
               radioON = false;
               tvON = false;
               currentMode = PLAY;
            }
         }
         else if (itemPage == 2 && haveRadio)
         {
            if (radioON)
            {
               radioON = false;
               currentMode = IDLE;
            }
            else
            {
               cardsON = false;
               bookON = false;
               radioON = true;
               tvON = false;
               currentMode = PLAY;
            }
         }
         else if (itemPage == 3 && haveTV)
         {
            if (tvON)
            {
               tvON = false;
               currentMode = IDLE;
```

```
            }
            else
            {
               cardsON = false;
               bookON = false;
               radioON = false;
               tvON = true;
               currentMode = PLAY;
            }
         }
         else
         {
            display.clearDisplay();
            display.setCursor(0, 0);
            display.setTextSize(1);
            display.println("LOCKED ITEM");
            display.display();
            delay(1000);
         }

         infoConfirm = false;
         pickingItem = false;
         infoBool = false;
         infoPage = 0;
         itemPage = 0;
      }

      if (itemPage > 3)
         itemPage = 0;

      display.display();
   }
}

void sleepy(int &sleeppoints)
{
   unsigned long currentTime5 = millis();
   if (currentTime5 - previousTime5 >= framestime5[currentFrames5])
   {
      previousTime5 = currentTime5;
      display.clearDisplay();
      display.drawBitmap(0, 0, frames5[currentFrames5], 128, 64, SSD1306_WHITE);
      display.display();
      currentFrames5++;
      Serial.println("Sleep points: ");
      Serial.println(sleeppoints);

      if (currentFrames5 >= totalFrames5)
      {
         currentFrames5 = 0;
         playpoints = playpoints + 1;
      }
```

```
    }
    if (currentTime5 - sleepTime > 60000)
    {
        sleepTime = currentTime5;
        sleeppoints = sleeppoints + 5;
    }

    if (sleeppoints > 100)
    {
        currentMode = IDLE;
        lastMode = SLEEP;
    }
    if (playpoints > 100)
        playpoints = 100;
    if (buttonPressCount == 15)
    {
        currentMode = ANGRY;
        lastMode = SLEEP;
        buttonPressCount = 0;
    }
}
void kitchen(int &hungrypoints, int &drinks, int &food)
{
    unsigned long currentTime8 = millis();

    if (!infoBool)
    {
        if (currentTime8 - previousTime8 >= 300)
        {
            previousTime8 = currentTime8;
            display.clearDisplay();
            display.drawBitmap(0, 0, kitchens[currentFrames8], 128, 64, SSD1306_WHITE);
            display.setTextColor(SSD1306_WHITE);
            display.setTextSize(1);
            display.setCursor(0, 0);
            display.println("Kitchen");

            display.display();
            currentFrames8++;
            if (currentFrames8 >= totalFrames8)
                currentFrames8 = 0;
        }
    }
    else
    {

        display.clearDisplay();
        display.drawBitmap(0, 0, roomStatic[2], 128, 64, SSD1306_WHITE);
        display.setCursor(0, 0);
        display.setTextColor(SSD1306_WHITE);
        if (currentMode != SLEEP)
        {
```

```
        if (infoPage == 0)
        {
          display.setTextSize(1.5);
          display.print(">Drink");
          display.println(drinks);
          display.setTextSize(1);
          display.println("Burger");
          if (infoConfirm && drinks >= 1)
          {
            infoConfirm = false;
            drinks--;
            Serial.println("Drank water.");
            Serial.print("Water bottles left: ");
            Serial.println(drinks);
            hungrypoints += 5;
          }
        }
        else if (infoPage == 1 && currentMode != SLEEP)
        {
          display.setTextSize(1);
          display.println("Water");
          display.setTextSize(1.5);
          display.print(">Eat");
          display.println(food);

          if (infoConfirm && food >= 1)
          {
            infoConfirm = false;
            food--;
            Serial.println("Ate burger.");
            Serial.print("Food left: ");
            Serial.println(food);
            hungrypoints += 10;
          }
        }
        if (hungerpoints > 100)
          hungerpoints = 100;
        if (infoPage > 1)
          infoPage = 0;

        display.display();
      }
      else
      {
        display.clearDisplay();
        display.drawBitmap(0, 0, kitchens[currentFrames8], 128, 64, SSD1306_WHITE);
        display.setTextColor(SSD1306_WHITE);
        display.setTextSize(1);
        display.setCursor(0, 0);
        display.println("Kitchen");
        display.println("Void is asleep");
```

```
            display.display();
        }
    }
}

void coinflip(int &money)
{
    unsigned long currentTime7 = millis();
    display.clearDisplay();
    display.setTextColor(SSD1306_WHITE);
    display.setTextSize(1);
    display.setCursor(0, 0);
    display.print("$");
    display.println(money);

    if (!infoBool)
    {
        display.drawBitmap(0, 0, roomStatic[1], 128, 64, SSD1306_WHITE);
    }
    else
    {
        switch (coinState)
        {
        case COINFLIP_IDLE:
            coinInProgress = true;
            coinStartTime = currentTime7;
            currentFrames7 = 0;
            winlose = random(1, 3); // 1 win, 2 lose
            coinState = COINFLIP_FLIPPING;
            break;

        case COINFLIP_FLIPPING:
            if (currentTime7 - previousTime7 >= framestime7[currentFrames7] &&
currentFrames7 < totalFrames7)
            {
                previousTime7 = currentTime7;
                display.drawBitmap(0, 0, coinflipanimation[currentFrames7], 128, 64,
SSD1306_WHITE);
                currentFrames7++;
            }
            else if (currentFrames7 >= totalFrames7)
            {
                coinStartTime = currentTime7;
                coinState = COINFLIP_SHOW_RESULT;
            }
            break;

        case COINFLIP_SHOW_RESULT:
            if (winlose == 1)
            {
                display.drawBitmap(0, 0, winloseframes[0], 128, 64, SSD1306_WHITE);
            }
```

```
            else
            {
               display.drawBitmap(0, 0, winloseframes[1], 128, 64, SSD1306_WHITE);
            }

            if (currentTime7 - coinStartTime >= 2300)
            {
               if (winlose == 1)
               {
                  money++;
               }
               infoBool = false;
               coinInProgress = false;
               coinState = COINFLIP_IDLE;
            }
            break;
         }
      }

      display.display();
}

void shop(int &money, int &drinks, int &food, bool &haveCards, bool &haveBook, bool
&haveRadio, bool &haveTV)
{
   unsigned long currentTime9 = millis();

   if (!infoBool)
   {
      if (currentTime9 - previousTime9 >= 300)
      {
         previousTime9 = currentTime9;
         display.clearDisplay();
         display.drawBitmap(0, 0, shops[currentFrames9], 128, 64, SSD1306_WHITE);
         display.setTextColor(SSD1306_WHITE);
         display.setTextSize(1);
         display.setCursor(0, 0);
         display.println("Shop");

         display.display();
         currentFrames9++;
         if (currentFrames9 >= totalFrames9)
            currentFrames9 = 0;
      }
   }
   else
   {
      display.clearDisplay();
      if (currentMode != SLEEP)
      {
         display.setCursor(0, 0);
         display.setTextColor(SSD1306_WHITE);
```

```
        display.setTextSize(1);

        if (infoPage == 0)
        {
           display.print(">Buy Water $5\n");
           display.println("Buy Burger");
           display.println("Buy Cards");
           display.println("Buy Book");
           display.println("Buy Radio");
           display.println("Buy TV");

           if (infoConfirm && money >= 5)
           {
              drinks++;
              money -= 5;
              Serial.println("Bought Water");
              infoConfirm = false;
              infoPage = 0;
           }
        }
        else if (infoPage == 1)
        {
           display.println("Buy Water");
           display.print(">Buy Burger $10\n");
           display.println("Buy Cards");
           display.println("Buy Book");
           display.println("Buy Radio");
           display.println("Buy TV");

           if (infoConfirm && money >= 10)
           {
              food++;
              money -= 10;
              Serial.println("Bought Burger");
              infoConfirm = false;
              infoPage = 0;
           }
        }
        else if (infoPage == 2)
        {
           display.println("Buy Water");
           display.println("Buy Burger");
           display.print(">Buy Cards $20\n");
           display.println("Buy Book");
           display.println("Buy Radio");
           display.println("Buy TV");

           if (infoConfirm && money >= 20 && !haveCards)
           {
              haveCards = true;
              money -= 20;
              Serial.println("Bought Cards");
```

```
            infoConfirm = false;
            infoPage = 0;
        }
        else if (infoConfirm && haveCards)
        {
            Serial.println("Already own Cards!");
            infoConfirm = false;
        }
    }
    else if (infoPage == 3)
    {
        display.println("Buy Water");
        display.println("Buy Burger");
        display.println("Buy Cards");
        display.print(">Buy Book $30\n");
        display.println("Buy Radio");
        display.println("Buy TV");

        if (infoConfirm && money >= 30 && !haveBook)
        {
            haveBook = true;
            money -= 30;
            Serial.println("Bought Book");
            infoConfirm = false;
            infoPage = 0;
        }
        else if (infoConfirm && haveBook)
        {
            Serial.println("Already own Book!");
            infoConfirm = false;
        }
    }
    else if (infoPage == 4)
    {
        display.println("Buy Water");
        display.println("Buy Burger");
        display.println("Buy Cards");
        display.println("Buy Book");
        display.print(">Buy Radio $50\n");
        display.println("Buy TV");

        if (infoConfirm && money >= 50 && !haveRadio)
        {
            haveRadio = true;
            money -= 50;
            Serial.println("Bought Radio");
            infoConfirm = false;
            infoPage = 0;
        }
        else if (infoConfirm && haveRadio)
        {
            Serial.println("Already own Radio!");
```

```
                infoConfirm = false;
            }
        }
        else if (infoPage == 5)
        {
            display.println("Buy Water");
            display.println("Buy Burger");
            display.println("Buy Cards");
            display.println("Buy Book");
            display.println("Buy Radio");
            display.print(">Buy TV $200\n");

            if (infoConfirm && money >= 200 && !haveTV)
            {
                haveTV = true;
                money -= 200;
                Serial.println("Bought TV");
                infoConfirm = false;
                infoPage = 0;
            }
            else if (infoConfirm && haveTV)
            {
                Serial.println("Already own TV!");
                infoConfirm = false;
            }
        }

        if (infoPage > 5)
            infoPage = 0;
    }
    else
    {
        display.drawBitmap(0, 0, shops[currentFrames8], 128, 64, SSD1306_WHITE);
        display.setCursor(0, 0);
        display.setTextColor(SSD1306_WHITE);
        display.setTextSize(1);
        display.println("Shop");
        display.println("Void is asleep");
    }
    display.display();
  }
}

void progress(int &healthpoints, int &sleeppoints, int &hungrypoints, int &playpoints)
{
  unsigned long currentTime2 = millis();
  display.clearDisplay();

  if (currentTime2 - previousTime2 >= 300)
  {
    previousTime2 = currentTime2;
    currentFrames2++;
```

```
      if (currentFrames2 >= 3)
         currentFrames2 = 0;
   }

   if (temperature >= 19 && temperature <= 27)
   {
      display.drawBitmap(0, 0, progiconsSun[currentFrames2], 128, 64,
SSD1306_WHITE);
      if (currentTime2 - HPtime >= 300000)
      {
         healthpoints -= 5;
         sleeppoints -= 5;
         hungrypoints -= 5;
         playpoints -= 5;
         HPtime = currentTime2;
      }
   }
   else if (temperature >= 15 && temperature <= 18)
   {
      display.drawBitmap(0, 0, progiconsCold[currentFrames2], 128, 64,
SSD1306_WHITE);
   }
   else if (temperature >= 0 && temperature <= 14)
   {
      display.drawBitmap(0, 0, progiconsFreeze[currentFrames2], 128, 64,
SSD1306_WHITE);
      if (currentTime2 - HPtime >= 300000)
      {
         healthpoints -= 5;
         sleeppoints -= 5;
         hungrypoints -= 7;
         playpoints += 2;
         HPtime = currentTime2;
      }
   }
   else if (temperature >= 28 && temperature <= 40)
   {
      display.drawBitmap(0, 0, progiconsHot[currentFrames2], 128, 64, SSD1306_WHITE);
      if (currentTime2 - HPtime >= 300000)
      {
         healthpoints -= 10;
         sleeppoints -= 10;
         hungrypoints -= 5;
         playpoints -= 5;
         HPtime = currentTime2;
      }
   }

   display.drawRoundRect(5, 16, 10, 40, 5, SSD1306_WHITE);  // Health
   display.drawRoundRect(25, 16, 10, 40, 5, SSD1306_WHITE); // Hunger
   display.drawRoundRect(45, 16, 10, 40, 5, SSD1306_WHITE); // Joy
   display.drawRoundRect(65, 16, 10, 40, 5, SSD1306_WHITE); // Sleep
```

```
  int barHeightHealth = map(healthpoints, 0, 100, 0, 40);
  int barYHealth = 16 + (40 - barHeightHealth);
  display.fillRoundRect(5, barYHealth, 10, barHeightHealth, 5, SSD1306_WHITE);

  // HUNGER BAR
  int barHeightHunger = map(hungrypoints, 0, 100, 0, 40);
  int barYHunger = 16 + (40 - barHeightHunger);
  display.fillRoundRect(25, barYHunger, 10, barHeightHunger, 5, SSD1306_WHITE);

  // JOY BAR
  int barHeightJoy = map(playpoints, 0, 100, 0, 40);
  int barYJoy = 16 + (40 - barHeightJoy);
  display.fillRoundRect(45, barYJoy, 10, barHeightJoy, 5, SSD1306_WHITE);

  // SLEEP BAR
  int barHeightSleep = map(sleeppoints, 0, 100, 0, 40);
  int barYSleep = 16 + (40 - barHeightSleep);
  display.fillRoundRect(65, barYSleep, 10, barHeightSleep, 5, SSD1306_WHITE);

  display.display();
}

void stats()
{
  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  display.setCursor(0, 0);

  if (infoPage == 0)
  {
    display.println("INFORMATION - Stats");
    display.setCursor(0, 16);
    display.println("On the next pages you will find general informations and how to keep
your Void pet healthy, safe and active.");
    display.setCursor(0, 0);
  }
  else if (infoPage == 1)
  {
    display.println("How to: Health");
    display.setCursor(0, 16);
    display.println("You can fill up Void's Health bar by petting him and keeping him well
fed.");
    display.setCursor(0, 0);
  }
  else if (infoPage == 2)
  {
    display.println("How to: Hunger");
    display.setCursor(0, 16);
    display.println("You can fill up Void's Hunger bar by feeding him in the kitchen.");
    display.setCursor(0, 0);
```

```
    }
    else if (infoPage == 3)
    {
        display.println("How to: Joy");
        display.setCursor(0, 16);
        display.println("You can fill up Void's Joy bar by playing the different minigames.
Don't try too hard, he doesn't mind losing.");
        display.setCursor(0, 0);
    }
    else if (infoPage == 4)
    {
        display.println("How to: Sleep");
        display.setCursor(0, 16);
        display.println("After you're done with everything be sure to lay him to bed. He loves
sleeping..Maybe too much.");
        display.setCursor(0, 0);
    }
    else if (infoPage == 5)
    {
        display.println("How to: Climate");
        display.setCursor(0, 16);
        display.println("Be sure the temperature isn't too high or too low. He may be an alien,
but he still likes comfort");
        display.setCursor(0, 0);
    }

    display.display();
}
```

Fișierul icons.h

```
const unsigned char *frames[] = {happyidle1, happyidle2, happyidle3, happyidle4,
happyidle5, happyidle6,
                        happyidle7, happyidle8, happyidle9, happyidle8, happyidle7,
happyidle6, happyidle5, happyidle4, happyidle3, happyidle2,
                        happyidle1, happyidle1, happyidle2, happyidle3, happyidle4,
happyidle5, happyidle6, happyidle7, happyidle8, happyidle9,
                        happyidle8, happyidle7, happyidle6, happyidle5, happyidle4,
happyidle3, happyidle2, happyidle1, happyidle1, happyidle2,
                        happyidle3, happyidle4, happyidle5, happyidle6, happyidle7,
happyidle8, happyidle9, happyidle8, happyidle7, happyidle6,
                        happyidle5, happyidle4, happyidle3, happyidle2, happyidle1,
happyidle10, happyidle11, happyidle12, happyidle13, happyidle14,
                        happyidle13, happyidle12, happyidle11, happyidle10, happyidle1};
const unsigned char *frames3[] = {cute1, cute2, cute3, cute4, cute5, cute6, cute7, cute6,
cute5, cute4, cute3, cute2, cute1};
const unsigned char *frames4[] = {angry1, angry2, angry3, angry4, angry5, angry6, angry7,
angry8, angry7, angry8, angry7, angry8, angry7, angry8, angry7, angry8, angry7, angry8,
angry6, angry5, angry4, angry3, angry2, angry1};
const unsigned char *frames5[] = {sleep1, sleep2, sleep3, sleep4};
```

```
const unsigned char *bedrooms[] = {bedroompic2, bedroompic3, bedroompic4};
const unsigned char *kitchens[] = {kitchenpic2, kitchenpic3, kitchenpic4};
const unsigned char *shops[] = {shoppic2, shoppic3, shoppic4};
const unsigned char *items[] = {cards, book, radio, tv};

const unsigned char *progiconsSun[] = {infosun1, infosun2, infosun3};
const unsigned char *progiconsCold[] = {infocold1, infocold2, infocold3};
const unsigned char *progiconsHot[] = {infohot1, infohot2, infohot3};
const unsigned char *progiconsFreeze[] = {infofreeze1, infofreeze2, infofreeze3};

const unsigned char *roomStatic[] = {bedroompic, coin, kitchenpic, shoppic};
const unsigned char *winloseframes[] = {winner, loser};
const unsigned char *coinflipanimation[] = {coin1, coin2, coin3, coin4, coin5, coin6, coin7,
coin8, coin9, coin10, coin11, coin12, coin13, coin14, coin15, coin17, coin18};
const unsigned framestime[] = {10, 10, 10, 10, 10, 10, 10, 10, 30, 10, 10, 10, 10, 10, 10, 10,
5000,
                    10, 10, 10, 10, 10, 10, 10, 10, 30, 10, 10, 10, 10, 10, 10, 10, 5000,
                    10, 10, 10, 10, 10, 10, 10, 10, 30, 10, 10, 10, 10, 10, 10, 10, 5000,
                    15, 15, 15, 15, 3000, 15, 15, 15, 15};
const unsigned framestime3[] = {10, 10, 10, 10, 10, 10, 2000, 5000, 10, 10, 10, 10, 10, 10};
const unsigned int framestime4[] = {20, 20, 20, 20, 50, 200, 300, 300, 300, 300, 300, 300,
300, 300, 300, 300, 200, 100, 80, 50, 40, 30, 20, 20};
const unsigned int framestime5[] = {4000, 200, 200, 200};
const unsigned int framestime7[] = {50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50,
50, 50, 50, 50, 500};

const int totalFrames = sizeof(frames) / sizeof(frames[0]);
const int totalFrames3 = sizeof(frames3) / sizeof(frames3[0]);
const int totalFrames4 = sizeof(frames4) / sizeof(frames4[0]);
const int totalFrames5 = sizeof(frames5) / sizeof(frames5[0]);
const int totalFrames6 = sizeof(bedrooms) / sizeof(bedrooms[0]);
const int totalFrames7 = sizeof(coinflipanimation) / sizeof(coinflipanimation[0]);
const int totalFrames8 = sizeof(kitchens) / sizeof(kitchens[0]);
const int totalFrames9 = sizeof(shops) / sizeof(shops[0]);
const int totalItems = sizeof(items) / sizeof(items[0]);

unsigned long previousTime = 0;
unsigned long previousTime2 = 0;
unsigned long previousTime3 = 0;
unsigned long previousTime4 = 0;
unsigned long previousTime5 = 0;
unsigned long previousTime6 = 0;
unsigned long previousTime7 = 0;
unsigned long previousTime8 = 0;
unsigned long previousTime9 = 0;

int currentFrames = 0;
int currentFrames2 = 0;
int currentFrames3 = 0;
int currentFrames4 = 0;
int currentFrames5 = 0;
int currentFrames6 = 0;
```

```
int currentFrames7 = 0;
int currentFrames8 = 0;
int currentFrames9 = 0;
```