King's Cup - Branzeu Mihnea 333CA

Introducere

Prin acest proiect mi-am propus sa automatizez jocul King's Cup, un drinking game cunoscut, cu ajutorul unor module ESP32.
Jocul va putea fi jucat de 3 jucatori, fiecare avand un modul ESP cu ajutorul caruia va putea juca. Ideea a pornit de la faptul ca in jocul clasic, se triseaza de multe ori, astfel ca in varianta aceasta nu se va putea trece mai departe pana ce nu toti isi vor duce la capat provocarile. Datorita automatizarii, voi putea realiza si un clasament, cu cantitatea de bautura consumata de fiecare jucator.

Regulament

Regulile jocului original de King's Cup presupun ca jucatorii sa se stranga in cerc, alaturi de bauturile lor si sa aseze un pachet de carti cu fata in jos in forma de cerc. Fiecare jucator va trage pe rand o carte, si in functie de numarul ei va face o provocare. Cartile au urmatoarele semnificatii pentru cel care este la rand:

- **2** - Alegi pe cineva care va bea o gura.
- **3** - Bei o gura.
- **4** - Fetele beau o gura.
- **5** - Thumb master: Oricand in timpul jocului poti pune degetul mare pe masa, ceilalti jucatori fiind nevoiti sa faca la fel. Ultima persoana care pune degetul pe masa bea o gura.
- **6** - Baietii beau o gura.
- **7** - Asemanator cu Thumb Master, insa trebuie ca degetul aratator sa fie indreptat in sus.
- **8** - Alegi pe cineva si beti impreuna o gura
- **9** - Spui un cuvant; Ceilalti trebuie sa spuna, pe rand, un cuvant care rimeaza. Cine nu stie sau repeta un cuvant bea o gura.
- **10** - Asemanator cu 9, insa in loc de rime jucatorii trebuie sa spuna cuvinte din aceeasi categorie cu primul.
- **J** - Alegi o regula care trebuie respectata pana la finalul jocului.
- **Q** - Pana la final, cei care iti raspund la orice intrebare trebuie sa bea o gura.
- **K** - Toti beau o gura.
- **A** - Cascada: Provocare speciala unde toata lumea bea, cu conditia ca un jucator nu se poate opri inainte jucatorului din stanga lui.

In cazul proiectului meu, pachetul de carti va fi inlocuit de un ecran LCD, care va afisa direct, cu ajutorul unui algoritm de simulare a unui pachet de carti, provocarea pentru fiecare jucator.

Descriere generală

Proiectul este alcatuit din doua componente:

1. Un modul ESP32 central, cu ajutorul caruia se va face setupul jocului si care va gestiona celelalte module
2. Mai multe module identice, alcatuite din ESP32, butoane, LEDuri si LCDuri, prin care jucatorii vor putea juca jocul

Initial, modulul master va fi configurat, fiind specificate informatii despre jucatori. Aceste informatii vor fi trimis apoi catre modulele secundare, prin Wi-Fi, cu ajutorul ESPNow.
Atunci cand jocul incepe, masterul va selecta random o provocare pe care o va afisa pe LCD-ul sau. Jucatorii care vor trebui sa execute provocarea vor fi anuntati cu ajutorul LEDurilor de pe modulele lor. Jocul nu va putea fi continuat pana ce toti executa provocarea, fapt ce poate fi determinat cu ajutorul senzorului de forta, care va sta sub paharul fiecaruia. In cazul in care se incearca trecerea la provocarea urmatoare fara ca toti sa execute provocarea, buzzerul de pe modulele celor care incearca sa triseze va incepe sa sune, oprindu-se doar atunci cand jucatorul executa provocarea.

Hardware Design

Proiectul este alcatuit din 4 componente distincte, dintre care una este modulul master, care controleaza flow-ul jocului, iar celelalte trei sunt module player, construite identic, cate unul pentru fiecare jucator.

Design Modul Master

Acest modul este alcatuit dintr-un ESP32, un LCD cu modul I2C integrat, si 4 butoane. Primele 3 butoane sunt utilizate pentru selectarea unuia dintre cei 3 jucatori in diferite situatii, pe parcursul jocului, iar cel de-al 4-lea este utilizat pentru a trece la urmatoarea provocare.

Schema electrica a modulului:

Componentele pe care le-am utilizat sunt:

- ESP32-WROOM-32D
- Display LCD 16×2 I2C
- 4xButoane
- LEDuri

Design Modul Player

Acest tip de modul este alcatuit dintr-un ESP32, 2 butoane, 2 LEDuri, un buzzer si un senzor de forta. Butoanele sunt utilizate pentru in timpul provocarilor, iar LEDurile au rolul de a semnala diferite evenimente pe parcursul jocului, precum randul unui anume jucator, sau faptul ca acesta trebuie sa bea. Senzorul de forta, plasat sub paharul jucatorului, are rolul de a detecta atunci cand acesta bea, pentru a exclude posibilele tentative de a trisa. Buzzerul anunta jucatorii care nu si-au indeplinit provocarea si incearca sa ascunda acest lucru.

Schema electrica a modulului:

Componentele pe care le-am utilizat sunt:

- ESP32-WROOM-32D
- 2xLED
- 2xButoane
- Force Resisting Sensor
- Buzzer

Software Design

Librarii folosite:

- **LiquidCrystal_I2C.h** - folosita pentru interfatarea cu LCD-ul
- **esp_now.h** si **WiFi.h** - folosite pentru comunicarea inter-modul

In cadrul proiectului am fost nevoit sa scriu doua programe separate, unul care sa ruleze pe modulul master, iar celalalt pentru a rula pe fiecare dintre modulele player.

Comunicarea inter-ESP32

Pentru a permite comunicarea intre cele 4 module ESP32 am folosit biblioteca **esp-now.h**. Aceasta foloseste adresa MAC a ESPului pentru a identifica statia in retea. Astfel, am avut nevoie de un script cu ajutorul caruia sa aflu adresa MAC a fiecarui modul:

```
#include "WiFi.h"

void setup() {
  // Setup Serial Monitor
  Serial.begin(9600);

  // Put ESP32 into Station mode
  WiFi.mode(WIFI_MODE_STA);
```

```
  // Print MAC Address to Serial monitor
  Serial.print("MAC Address: ");
  Serial.println(WiFi.macAddress());
}

void loop() {}
```

Avand adresa MAC, am trecut la a construi un standard de comunicare. Am obtinut urmatoarea configuratie:

```
#define MESSAGE_DRINK 0
#define MESSAGE_CONFIRM 1
#define MESSAGE_SCREAM 2
#define MESSAGE_SAY 3
#define MESSAGE_GO_NEXT 4
#define MESSAGE_THUMB_MASTER 5
#define MESSAGE_WATERFALL_GO 6
#define MESSAGE_WATERFALL_STOP 7
#define MESSAGE_WATERFALL_GO_INITIATOR 8

typedef struct message_t {
  int sender;
  int receiver;
  int messageType;
} message_t;
```

Cu ajutorul define-urilor, am putut sa impart mesajele in mai multe categorii, acestea completand campul **messageType** din cadrul structurii. De asemenea, structura **message_t** contine layout-ul unui mesaj intre doua module: senderul (sursa mesajului), receiverul (destinatia mesajului) si tipul mesajului. Acest segment de cod trebuie sa se regaseasca atat pe master, cat si pe player.

Mai departe, cu ajutorul functiilor **OnDataSent()**, **OnDataReceived()** si **esp_now_send()**, oferite de biblioteca, am putut realiza comunicarea intre module.

Cod Master

Modulul Master este realizat sub forma unui state-machine, avand urmatoarele stari:

```
#define NEW_GAME_STATE 0
#define NEW_CARD_STATE 1
#define WAITING_FOR_DRINKS_STATE 2
#define GAME_OVER_STATE 3
#define IDLE_STATE 4
#define CHOOSING_SOMEONE_STATE 5
#define COUNTDOWN_STATE 6
#define THUMB_MASTER_STATE 7
#define WATERFALL_STATE 8
```

Logica este alcatuita din urmatorii pasi:

1. O carte este generata random
2. Functia corespunzatoare cartii este apelata si provocarea incepe
3. Playerii care trebuie sa bea sunt anuntati, si se asteapta ca ei sa execute provocarea
4. Atunci cand se doreste trecerea la urmatoarea carte, functia de verificare este apelata
5. Daca toti jucatorii au indeplinit provocarea, se revine la pasul **1**

Un jucator este definit sub forma unei structuri astfel:

```
// The structure containing info about players
typedef struct player_t {
  uint8_t macAdress[6]; // The MAC address used for communication
  int gender; // Used for special challenges
  bool shouldDrink; // Flag for keeping track of the status for each player
  bool thumbMaster; // Flag for the thumb master challenge
} player_t;
```

Codul integral al modulului master:

```
#include <LiquidCrystal_I2C.h>
#include <esp_now.h>
#include <WiFi.h>

// Pin Setup
#define BUTTON_1_PIN 33
#define BUTTON_2_PIN 25
#define BUTTON_3_PIN 26
#define BUTTON_NEXT_PIN 32

// States definition
#define NEW_GAME_STATE 0
#define NEW_CARD_STATE 1
#define WAITING_FOR_DRINKS_STATE 2
#define GAME_OVER_STATE 3
#define IDLE_STATE 4
#define CHOOSING_SOMEONE_STATE 5
#define COUNTDOWN_STATE 6
#define THUMB_MASTER_STATE 7
#define WATERFALL_STATE 8

#define CARDS_PER_NUMBER 4
#define NUMBER_OF_PLAYERS 3

// Message types
#define MESSAGE_DRINK 0
#define MESSAGE_CONFIRM 1
#define MESSAGE_SCREAM 2
#define MESSAGE_SAY 3
#define MESSAGE_GO_NEXT 4
#define MESSAGE_THUMB_MASTER 5
```

```
#define MESSAGE_WATERFALL_GO 6
#define MESSAGE_WATERFALL_STOP 7
#define MESSAGE_WATERFALL_GO_INITIATOR 8

#define GIRL 0
#define BOY 1

#define PLAYER_1 0
#define PLAYER_2 1
#define PLAYER_3 2
#define MASTER 3


// The structure containing info about players
typedef struct player_t {
  uint8_t macAdress[6]; // The MAC address used for communication
  int gender; // Used for special challenges
  bool shouldDrink; // Flag for keeping track of the status for each player
  bool thumbMaster; // Flag for the thumb master challenge
} player_t;

// The structure containing the layout of a message
typedef struct message_t {
  int sender;
  int receiver;
  int messageType;
} message_t;

// The players array
player_t players[3];

// The left amount of each card
int cardFrequencies[15];
int cardsLeft = (CARDS_PER_NUMBER) * 12;

// The current state of the game
int currentState;
int currentTurn;

// Variables used for countdown
double a = millis();
double i = 0;
double c;
bool shouldRun = false;
bool shouldReset = false;
int currentToAnswer;

// Variable used for thumb master
int thumbMasterCount;

// Peer info structure
```

```
esp_now_peer_info_t peerInfo;

LiquidCrystal_I2C lcd(0x27,16,2);  // set the LCD address to 0x3F for a 16
chars and 2 line display

// Callback when data is sent
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  char macStr[18];
  Serial.print("Packet to: ");
  // Copies the sender mac address to a string
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
           mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4],
mac_addr[5]);
  Serial.print(macStr);
  Serial.print(" send status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" :
"Delivery Fail");
}

// Callback function executed when data is received
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len) {
  message_t message;

  memcpy(&message, incomingData, sizeof(message_t));
  Serial.print("Data received: ");
  Serial.println(len);
  Serial.print("Sender: ");
  Serial.println(message.sender);
  Serial.print("Message type: ");
  Serial.println(message.messageType == 0 ? "DRINK" : "CONFIRM");

  // Check the type of message
  if (message.messageType == MESSAGE_CONFIRM) {
    // Mark the drink flag for the player
    players[message.sender].shouldDrink = false;
  }

  if (message.messageType == MESSAGE_GO_NEXT) {
    // Send the say message to the next player
    shouldReset = true;
    player_t nextPlayer = players[(message.sender + 1) % NUMBER_OF_PLAYERS];
    currentToAnswer = (message.sender + 1) % NUMBER_OF_PLAYERS;
    sendMessage((message.sender + 1) % NUMBER_OF_PLAYERS, MESSAGE_SAY,
nextPlayer.macAdress);
  }

  if (message.messageType == MESSAGE_THUMB_MASTER) {
    // Mark the player
    players[message.sender].thumbMaster = true;
    thumbMasterCount++;
  }
```

```
  if (message.messageType == MESSAGE_WATERFALL_STOP) {
    if ((message.sender  + 1) % NUMBER_OF_PLAYERS != currentTurn) {
      player_t nextPlayer = players[(message.sender + 1) % NUMBER_OF_PLAYERS];
      sendMessage((message.sender + 1) % NUMBER_OF_PLAYERS,
MESSAGE_WATERFALL_STOP, nextPlayer.macAdress);
    }
    else {
      currentState = WAITING_FOR_DRINKS_STATE;
    }
  }
}

void sendMessage(int receiver, int messageType, uint8_t *macAdress) {
  // Create the message
  message_t message = {MASTER, receiver, messageType};

  // Send the message
  esp_err_t result = esp_now_send(macAdress, (uint8_t *) &message, sizeof(
message_t));

  // Check the result
  if (result == ESP_OK) {
    Serial.println("Sent with success");
  }
  else {
    Serial.println("Error sending the data");
  }
}

void lcdPrintAnimation() {
  lcd.clear();
  lcd.setCursor(2, 0);
  lcd.print(" PLAYER ");
  lcd.setCursor(10, 0);
  lcd.print(currentTurn + 1);
  delay(2000);
  lcd.clear();
  int i, j;
  lcd.setCursor(0, 0);
  for (int i = 0; i < 16; i++) {
    for (int j = 0; j < i; j++) {
      lcd.setCursor(j, 0);
      lcd.print("=");
      lcd.setCursor(15 - j, 1);
      lcd.print("=");
    }
    delay(100);
  }
}
```

```
void lcdPrint(char *message, int row, int shouldClear) {
  if (shouldClear) {
    lcd.clear();
  }
  lcd.setCursor(2, row);
  lcd.print(message);
}

void twoChallenge() {
  // Print the message on the LCD
  lcdPrint("2 IS YOU", 0, 1);
  lcdPrint("Choose Someone", 1, 0);

  // Change the state to choosing someone
  currentState = CHOOSING_SOMEONE_STATE;
}

void threeChallenge() {
  // Print the message on the LCD
  lcdPrint("3 IS ME", 0, 1);
  lcdPrint(" Drink!!", 1, 0);

  // Mark the player as drinker
  players[currentTurn].shouldDrink = true;

  // Send the message
  sendMessage(currentTurn, MESSAGE_DRINK, players[currentTurn].macAdress);
}

void fourChallenge() {
  lcdPrint("4 IS GIRLS", 0, 1);
  lcdPrint("Girls Drink", 1, 0);

  // Search for the girls
  for (int i = 0; i < NUMBER_OF_PLAYERS; i++) {
    if (players[i].gender == GIRL) {
      // Mark the player as drinker
      players[i].shouldDrink = true;

      // Send the message
      sendMessage(i, MESSAGE_DRINK, players[i].macAdress);
    }
  }

}

void fiveChallenge() {
  lcdPrint("  THUMB MASTER", 0, 1);
  lcdPrint("     GO", 1, 0);

  // Reset the thumb master flag for players
```

```
  for (int i = 0; i < NUMBER_OF_PLAYERS; i++) {
    players[i].thumbMaster = false;
  }

  currentState = THUMB_MASTER_STATE;

  // Send the message to the players
  for (int i = 0; i < NUMBER_OF_PLAYERS; i++) {
    sendMessage(i, MESSAGE_THUMB_MASTER, players[i].macAdress);
  }

  // Initialize the counter
  thumbMasterCount = 0;
}

void sixChallenge() {
  lcdPrint("6 IS GUYS", 0, 1);
  lcdPrint("Boys Drink", 1, 0);

  // Search for the boys
  for (int i = 0; i < NUMBER_OF_PLAYERS; i++) {
    if (players[i].gender == BOY) {
      // Mark the player as drinker
      players[i].shouldDrink = true;

      // Send the message
      sendMessage(i, MESSAGE_DRINK, players[i].macAdress);
    }
  }
}

void eightChallenge() {
  lcdPrint("8 IS MATE", 0, 1);
  lcdPrint("Choose Someone", 1, 0);

  // Mark the player as drinker
  players[currentTurn].shouldDrink = true;
  // Send the message
  sendMessage(currentTurn, MESSAGE_DRINK, players[currentTurn].macAdress);

  // Change the state to choosing someone
  currentState = CHOOSING_SOMEONE_STATE;
}

void nineChallenge() {
  lcdPrint("9 IS RHYME", 0, 1);
  lcdPrint("BEGIN", 1, 0);

  delay(2000);

  // Update the current state
```

```
  currentState = COUNTDOWN_STATE;
  currentToAnswer = currentTurn;
  sendMessage(currentTurn, MESSAGE_SAY, players[currentTurn].macAdress);

}

void tenChallenge() {
  lcdPrint("10 IS CATEGORY", 0, 1);
  lcdPrint("BEGIN", 1, 0);

  delay(2000);

  // Update the current state
  currentState = COUNTDOWN_STATE;
  currentToAnswer = currentTurn;
  sendMessage(currentTurn, MESSAGE_SAY, players[currentTurn].macAdress);
}

void JChallenge() {
  lcdPrint("J IS RULE", 0, 1);
  lcdPrint("Choose a Rule", 1, 0);

}

void QChallenge() {
  lcdPrint("Q IS QUESTION", 0, 1);
}

void KChallenge() {
  lcdPrint("K IS EVERYONE", 0, 1);
  lcdPrint("Everyone Drinks", 1, 0);

  // Mark everyone as drinker
  for (int i = 0; i < NUMBER_OF_PLAYERS; i++) {
    // Mark the player as drinker
    players[i].shouldDrink = true;

    // Send the message
    sendMessage(i, MESSAGE_DRINK, players[i].macAdress);
  }
}

void AChallenge() {
  lcdPrint("A IS WATERFALL", 0, 1);

  currentState = WATERFALL_STATE;

  // Send the go message to the initiator
  sendMessage(currentTurn, MESSAGE_WATERFALL_GO_INITIATOR, players[
currentTurn].macAdress);
  // Send the go message to all the players
```

```
  for (int i = 0; i < NUMBER_OF_PLAYERS; i++) {
    if (i == currentTurn) {
      continue;
    }
    sendMessage(i, MESSAGE_WATERFALL_GO, players[i].macAdress);
  }
}

void startRound(int card) {
  // Update the game state
  currentState = WAITING_FOR_DRINKS_STATE;

  // Check the value of the card and proceed accordingly
  switch (card) {
    case 2:
      twoChallenge();
      break;
    case 3:
      threeChallenge();
      break;
    case 4:
      fourChallenge();
      break;
    case 5:
      fiveChallenge();
      break;
    case 6:
      sixChallenge();
      break;
    case 8:
      eightChallenge();
      break;
    case 9:
      nineChallenge();
      break;
    case 10:
      tenChallenge();
      break;
    case 11:
      JChallenge();
      break;
    case 12:
      QChallenge();
      break;
    case 13:
      KChallenge();
      break;
    case 14:
      AChallenge();
      break;
  }
```

```
}

int canMoveToNext() {
  int ret = 1;
  // Check the flag for each player
  for (int i = 0; i < NUMBER_OF_PLAYERS; i++) {
    // Check if the player drank
    if (players[i].shouldDrink) {
      // Send the alert message
      sendMessage(i, MESSAGE_SCREAM, players[i].macAdress);
      ret = 0;
    }
  }

  // Everybody drank
  return ret;
}

void setup() {
  Serial.begin(9600);

  // Initialize the players
  players[0].macAdress[0] = 0xD4;
  players[0].macAdress[1] = 0xD4;
  players[0].macAdress[2] = 0xDA;
  players[0].macAdress[3] = 0x5B;
  players[0].macAdress[4] = 0x41;
  players[0].macAdress[5] = 0x88;
  players[0].gender = GIRL;

  players[1].macAdress[0] = 0xD4;
  players[1].macAdress[1] = 0xD4;
  players[1].macAdress[2] = 0xDA;
  players[1].macAdress[3] = 0x5A;
  players[1].macAdress[4] = 0x5F;
  players[1].macAdress[5] = 0x3C;
  players[1].gender = BOY;

  players[2].macAdress[0] = 0xD4;
  players[2].macAdress[1] = 0xD4;
  players[2].macAdress[2] = 0xDA;
  players[2].macAdress[3] = 0x5A;
  players[2].macAdress[4] = 0x66;
  players[2].macAdress[5] = 0xEC;
  players[2].gender = BOY;

  // Initialize ESP NOW stuff
  WiFi.mode(WIFI_STA);
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP NOW");
```

```
    return;
  }

  esp_now_register_send_cb(OnDataSent);
  esp_now_register_recv_cb(OnDataRecv);


  // Register peers
  peerInfo.channel = 0;
  peerInfo.encrypt = false;
  for (int i = 0; i < NUMBER_OF_PLAYERS; i++) {
    memcpy(peerInfo.peer_addr, players[i].macAdress, 6);
    if (esp_now_add_peer(&peerInfo) != ESP_OK){
      Serial.println("Failed to add peer");
      return;
    }
  }

  // LCD CODE
  lcd.init();
  lcd.clear();
  lcd.backlight();

  // Print a message on both lines of the LCD.
  lcd.setCursor(2,0);   //Set cursor to character 2 on line 0
  lcd.print("King's Cup!");

  lcd.setCursor(2,1);   //Move cursor to character 2 on line 1
  lcd.print("Press NEXT");
  //--------------------------

  //BUTTON SETUP
  pinMode(BUTTON_1_PIN, INPUT_PULLUP);
  pinMode(BUTTON_2_PIN, INPUT_PULLUP);
  pinMode(BUTTON_3_PIN, INPUT_PULLUP);
  pinMode(BUTTON_NEXT_PIN, INPUT_PULLUP);

  //--------------------------

  // Initialize the card frequencies
  for (int i = 2; i < 15; i++) {
    cardFrequencies[i] = CARDS_PER_NUMBER;
  }
  // 7 card won't be played
  cardFrequencies[7] = 0;

  // Initialize the current state of the game
  currentState = NEW_GAME_STATE;
  currentTurn = 0;
}
```

```
void loop() {
  if (currentState == NEW_GAME_STATE) {
    // Check if the button was pressed
    int nextButtonValue = digitalRead(BUTTON_NEXT_PIN);
    if (!nextButtonValue) {
      // Switch into the new card state
      currentState = NEW_CARD_STATE;
    }
  }

  if (currentState == NEW_CARD_STATE) {
    // Check if game should be over
    if (cardsLeft == 0) {
      currentState = GAME_OVER_STATE;
    } else {
      lcdPrintAnimation();
      // Generate a new card
      int cardValue;
      while (1) {
        cardValue = random(2, 15);
        if (cardFrequencies[cardValue] > 0) {
          cardFrequencies[cardValue]--;
          cardsLeft--;
          break;
        }
      }
      Serial.print("Card Value: ");
      Serial.println(cardValue);

      // Start the new round depening on the card
      startRound(cardValue);
    }
  }

  if (currentState == WAITING_FOR_DRINKS_STATE) {
    // Check the next button state
    int nextButtonValue = digitalRead(BUTTON_NEXT_PIN);
    if (!nextButtonValue) {
      // Check if everybody is ready
      if (canMoveToNext()) {
        currentState = NEW_CARD_STATE;
        currentTurn = (currentTurn + 1) % NUMBER_OF_PLAYERS;
      }
    }
  }

  if (currentState == CHOOSING_SOMEONE_STATE) {
    // Check the input on the buttons
    int button1Value = digitalRead(BUTTON_1_PIN);
    int button2Value = digitalRead(BUTTON_2_PIN);
    int button3Value = digitalRead(BUTTON_3_PIN);
```

```
    if (!button1Value) {
      players[PLAYER_1].shouldDrink = true;
      sendMessage(PLAYER_1, MESSAGE_DRINK, players[PLAYER_1].macAdress);
      // Update the current state
      currentState = WAITING_FOR_DRINKS_STATE;
    } else if (!button2Value) {
      players[PLAYER_2].shouldDrink = true;
      sendMessage(PLAYER_2, MESSAGE_DRINK, players[PLAYER_2].macAdress);
      // Update the current state
      currentState = WAITING_FOR_DRINKS_STATE;
    } else if (!button3Value) {
      players[PLAYER_3].shouldDrink = true;
      sendMessage(PLAYER_3, MESSAGE_DRINK, players[PLAYER_3].macAdress);
      // Update the current state
      currentState = WAITING_FOR_DRINKS_STATE;
    }
  }

  if (currentState == COUNTDOWN_STATE) {
    lcd.clear();
    a = millis();
    shouldRun = true;
    while (shouldRun) {
      c = millis();
      i = (c - a) / 1000;
      if (shouldReset) {
        a = millis();
        shouldReset = false;
      }
      if (i >= 5) {
        shouldRun = false;
      }
      lcd.setCursor(5, 0);
      lcd.print(i);
      delay(100);
    }
    players[currentToAnswer].shouldDrink = true;
    sendMessage(currentToAnswer, MESSAGE_DRINK, players[currentToAnswer].
macAdress);
    currentState = WAITING_FOR_DRINKS_STATE;
  }

  if (currentState == THUMB_MASTER_STATE) {
    // Check if only one remains
    if (thumbMasterCount == NUMBER_OF_PLAYERS - 1) {
      // Check who it is
      for (int i = 0; i < NUMBER_OF_PLAYERS; i++) {
        if (!players[i].thumbMaster) {
          players[i].shouldDrink = true;
          sendMessage(i, MESSAGE_DRINK, players[i].macAdress);
```

```
          break;
        }
      }
      currentState = WAITING_FOR_DRINKS_STATE;
    }

  }

  if (currentState == GAME_OVER_STATE) {
    lcdPrint(" GAME OVER", 0, 1);
    currentState = IDLE_STATE;
  }
}
```

Cod Player

Asemanator cu Masterul, Playerul este construi tot sub forma unui state-machine:

```
#include <esp_now.h>
#include <WiFi.h>

#define THUMB_MASTER_BUTTON_PIN 18
#define GENERAL_PURPOSE_BUTTON_PIN 19
#define FORCE_SENSOR_PIN 36
#define LED_PIN 22
#define LED2_PIN 17
#define BUZZER_PIN 23

#define IDLE_STATE 0
#define SHOULD_DRINK_STATE 1
#define SHOULD_SAY_STATE 2
#define THUMB_MASTER_STATE 3
#define WATERFALL_STATE 4

#define MESSAGE_DRINK 0
#define MESSAGE_CONFIRM 1
#define MESSAGE_SCREAM 2
#define MESSAGE_SAY 3
#define MESSAGE_GO_NEXT 4
#define MESSAGE_THUMB_MASTER 5
#define MESSAGE_WATERFALL_GO 6
#define MESSAGE_WATERFALL_STOP 7
#define MESSAGE_WATERFALL_GO_INITIATOR 8


#define GLASS_NOT_LIFTED 0
#define GLASS_LIFTED 1
#define GLASS_BACK 2
```

```
typedef struct message_t {
  int sender;
  int receiver;
  int messageType;
} message_t;

// MAC Address of the master ESP
uint8_t broadcastAddress[] = {0xE8, 0x31, 0xCD, 0x14, 0x21, 0x1C};

// The player ID
int myId = -1;

// Peer info structure
esp_now_peer_info_t peerInfo;

// The drinking flag
int currentState;

// The state of the buzzer
bool buzzerOn = false;

// The state of the glass
int glassState;

// Flag for the waterfall challenge
bool canPutBack;
bool isInitiator;

const int movingAvgLength = 10;
int movingAvgBuffer[movingAvgLength];
int movingAvgIndex = 0;

// Callback function executed when data is received
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len) {
  message_t message;

  memcpy(&message, incomingData, sizeof(message_t));
  Serial.print("Data received: ");
  Serial.println(len);
  Serial.print("Sender: ");
  Serial.println(message.sender);
  Serial.print("Message type: ");
  Serial.println(message.messageType == 0 ? "DRINK" : "CONFIRM");

  if (message.messageType == MESSAGE_DRINK) {
    currentState = SHOULD_DRINK_STATE;
    glassState = GLASS_NOT_LIFTED;
  }

  if (message.messageType == MESSAGE_SCREAM) {
```

```
    buzzerOn = true;
  }

  if (message.messageType == MESSAGE_SAY) {
    currentState = SHOULD_SAY_STATE;
  }

  if (message.messageType == MESSAGE_THUMB_MASTER) {
    currentState = THUMB_MASTER_STATE;
  }

  if (message.messageType == MESSAGE_WATERFALL_GO) {
    currentState = WATERFALL_STATE;
    glassState = GLASS_NOT_LIFTED;
    canPutBack = false;
    isInitiator = false;
  }

  if (message.messageType == MESSAGE_WATERFALL_GO_INITIATOR) {
    currentState = WATERFALL_STATE;
    glassState = GLASS_NOT_LIFTED;
    canPutBack = false;
    isInitiator = true;
  }

  if (message.messageType == MESSAGE_WATERFALL_STOP) {
    canPutBack = true;
  }

   // Check if it's the first message and update ID
  if (myId == -1) {
    myId = message.receiver;
  }


}

// Callback when data is sent
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  char macStr[18];
  Serial.print("Packet to: ");
  // Copies the sender mac address to a string
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
           mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4],
mac_addr[5]);
  Serial.print(macStr);
  Serial.print(" send status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" :
"Delivery Fail");
}
```

```
int calculateMovingAverage(int sensorValue) {
  movingAvgBuffer[movingAvgIndex] = sensorValue;
  movingAvgIndex = (movingAvgIndex + 1) % movingAvgLength;

  int sum = 0;
  for (int i = 0; i < movingAvgLength; i++) {
    sum += movingAvgBuffer[i];
  }

  return sum / movingAvgLength;
}



void setup() {
  // Initialize serial communication at 9600 bits per second:
  Serial.begin(9600);

  // Initialize ESP NOW stuff
  // Set ESP32 as a Wi-Fi Station
  WiFi.mode(WIFI_STA);

  // Initilize ESP-NOW stuff
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }

  // Register callback function
  esp_now_register_recv_cb(OnDataRecv);
  esp_now_register_send_cb(OnDataSent);

  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }


  // Initialize the thumb master button
  pinMode(THUMB_MASTER_BUTTON_PIN, INPUT_PULLUP);
  pinMode(GENERAL_PURPOSE_BUTTON_PIN, INPUT_PULLUP);
  // Initialize the led
  pinMode(LED_PIN, OUTPUT);
  pinMode(LED2_PIN, OUTPUT);
  // Initialize the buzzer
  pinMode(BUZZER_PIN, OUTPUT);

  currentState = IDLE_STATE;
}
```

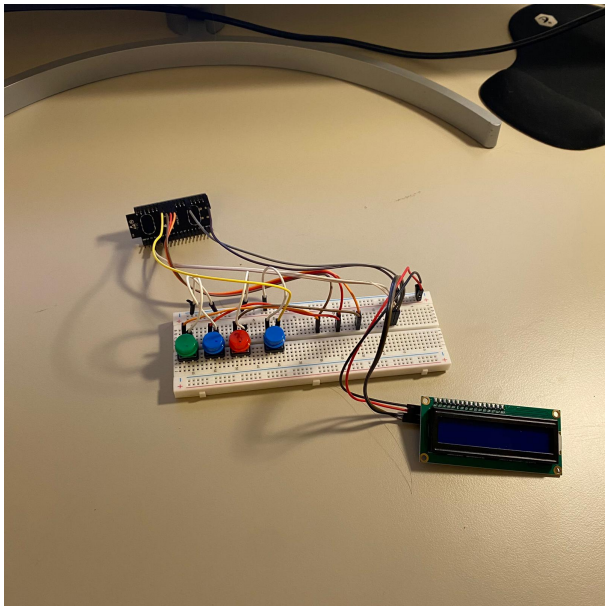
```
void loop() {
  if (currentState == SHOULD_DRINK_STATE) {
    // Turn on the LED
    digitalWrite(LED_PIN, HIGH);
    digitalWrite(LED2_PIN, LOW);

    if (buzzerOn) {
      digitalWrite(BUZZER_PIN, HIGH);
    }

    // Read the FSR value
    int forceSensorValue = calculateMovingAverage(analogRead(
FORCE_SENSOR_PIN));


    if ((forceSensorValue <5) && glassState == GLASS_NOT_LIFTED) {
      glassState = GLASS_LIFTED;
    }

    if ((forceSensorValue > 100) && glassState == GLASS_LIFTED) {
      glassState = GLASS_BACK;
    }

    if (glassState == GLASS_BACK) {
      // Send the confirmation message
      // Create the message
      message_t message = {myId, 3, MESSAGE_CONFIRM};

      // Send the message
      esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &message,
sizeof(message_t));

      // Check the result
      if (result == ESP_OK) {
        Serial.println("Sent with success");
        currentState = IDLE_STATE;
        buzzerOn = false;
      }
      else {
        Serial.println("Error sending the data");
      }
    }
  }

  if (currentState == SHOULD_SAY_STATE) {
    // Turn on the LED
    digitalWrite(LED2_PIN, HIGH);

    // Wait for the button press
    int generalPurposeButtonState = digitalRead(GENERAL_PURPOSE_BUTTON_PIN);
```

```
    if (!generalPurposeButtonState) {
      // Send the message to master to go to the next player
      message_t message = {myId, 3, MESSAGE_GO_NEXT};
      esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &message,
sizeof(message_t));
      if (result == ESP_OK) {
        Serial.println("Sent with success");
        currentState = IDLE_STATE;
        buzzerOn = false;
      }
      else {
        Serial.println("Error sending the data");
      }
      digitalWrite(LED2_PIN, LOW);
    }
  }

  if (currentState == THUMB_MASTER_STATE) {
    int thumbMasterButtonState = digitalRead(THUMB_MASTER_BUTTON_PIN);
    if (!thumbMasterButtonState) {
      message_t message = {myId, 3, MESSAGE_THUMB_MASTER};
      esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &message,
sizeof(message_t));
      if (result == ESP_OK) {
        Serial.println("Sent with success");
        currentState = IDLE_STATE;
        buzzerOn = false;
      }
      else {
        Serial.println("Error sending the data");
      }
    }
  }

  if (currentState == WATERFALL_STATE) {
    int forceSensorValue = calculateMovingAverage(analogRead(
FORCE_SENSOR_PIN));
    if (forceSensorValue <5 && (glassState == GLASS_NOT_LIFTED || glassState
== GLASS_BACK)) {
      glassState = GLASS_LIFTED;
    }

    if (forceSensorValue > 5 && glassState == GLASS_LIFTED) {
      glassState = GLASS_BACK;
      if (isInitiator) {
        canPutBack = true;
      }
    }

    if ((glassState == GLASS_NOT_LIFTED || glassState == GLASS_BACK) && !
canPutBack) {
```

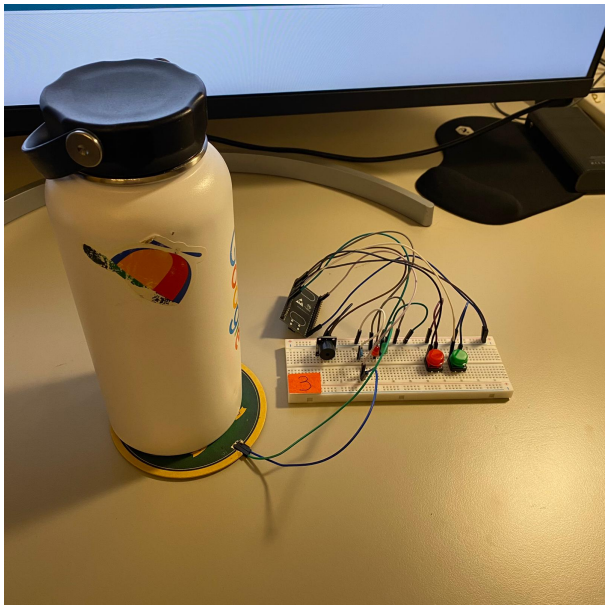
```
      // Turn on the alert
      digitalWrite(LED_PIN, HIGH);
      digitalWrite(BUZZER_PIN, HIGH);
    }

    if (glassState == GLASS_LIFTED && !canPutBack) {
      digitalWrite(LED_PIN, LOW);
      digitalWrite(BUZZER_PIN, LOW);
    }

    if (glassState == GLASS_BACK && canPutBack) {
      message_t message = {myId, 3, MESSAGE_WATERFALL_STOP};
      esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &message,
sizeof(message_t));
      if (result == ESP_OK) {
        Serial.println("Sent with success");
        currentState = IDLE_STATE;
        buzzerOn = false;
      }
      else {
        Serial.println("Error sending the data");
      }
    }
  }

  if (currentState == IDLE_STATE) {
    // Turn off the LED
    digitalWrite(LED_PIN, LOW);
    digitalWrite(BUZZER_PIN, LOW);
  }
}
```

Rezultate Obţinute

Modulul Master:



Modul Player:



Video cu flow-ul jocului:

Concluzii

Obiectivul meu in cadrul proiectului a fost sa gasesc o idee noua, si sa o implementez intr-un mod care sa ma invete ceva nou. Consider ca am atins ambele obiective. Nu am putut gasi pe internet ceva asemanator, ceea ce a fost in acelasi timp satisfacator si challenging, pentru ca nu am avut nicio varianta de fallback. Cel mai important lucru pe care l-am invatat este functionalitatea bibliotecii ESP-NOW.

Cel mai mare challenge are legatura cu faptul ca proiectul este alcatuit din multe module, ceea ce a facut testarea destul de dificila. Pe partea de cod, implementarea initiala a comunicarii si a unei provocari de test a fost cea mai interesanta, urmand ca restul de functionalitati sa fie usor de

implementat, fiind foarte asemanatoare cu prima.

Download

Link Github: https://github.com/mihneabranzeu/King-s-Cup-Arduino

Jurnal

- 5 Mai - Am creat pagina de documentatie
- 6 Mai - Am comandat piesele
- 10 Mai - Au ajuns piesele
- 20 - 23 Mai - Am lucrat la proiect
- 24 Mai - Am prezentat proiectul la laborator
- 30 Mai - PM Fair

Bibliografie/Resurse

- https://www.youtube.com/watch?v=bEKjCDDUPaU
- https://dronebotworkshop.com/esp-now/