Dokumentacija projekta iz ugradbenih sistema

Connect4

Vedad Hajrić, Ilhan Hasičić, Dževad Madžak

1

UVOD

Za ovaj projekat smo koristili LVGL za MicroPython za prikaz korisničkog interfejsa. Za rad sa ovom bibliotekom je potrebno instalirati poseban firmware na PicoETF sistem, kao i uključiti datoteke ili9xxx.py, lv_utils.py i st77xx.py (sve ovo će biti priloženo uz ovu dokumentaciju). Na ovom mjestu je također vrijedno spomenuti da nigdje u kodu nije korišten sistem prekida iako bi znatno olakšao neke dijelove implementacije i učinio kod čitljivijim. Naime kod korištenja sistema prekida smo naišli određene probleme sa prikazom interfejsa (preciznije elementi LVGL biblioteke se nisu prikazivali sve dok se nije izvršila akcija (pomjeranje analognog sticka).

IMPORTI I INICIJALIZACIJA ELEMENATA

Što se tiče korištenih biblioteka, uključili smo *ili9xxx* biblioteku koju smo spomenuli u uvodu, kao i samu biblioteku *lvgl*. Dalje iz *machine* smo importovali SPI (za inicijalizaciju ekrana), Pin i ADC. Dalje smo importovali *time* zbog korištenja time.sleep() metode, biblioteku *gc* jer je LVGL biblioteka poprilično memorijski skupa pa je potrebno imati neku vrstu “prikupljanja smeća“, i zadnje je potrebna biblioteka *random* koja će nam služiti pri donošenju odluka AI-a u singleplayer modu.

```
import ili9xxx
from machine import SPI, Pin, ADC
import lvgl as lv
import time
import gc
import random

# Initialize SPI and display driver
spi = SPI(0, baudrate=62500000, miso=Pin(16), mosi=Pin(19), sck=Pin(18))
drv = ili9xxx.Ili9341(spi=spi, dc=15, cs=17, rst=20)

adc_x = ADC(Pin(28))
select = Pin(17)
```

2

Što se tiče elemenata, inicijalizirali smo na početku ekran preko SPI, i analogni ulaz na pinu 28 koji je ustvari horizontalno pomjeranje analognog sticka, i posljednje digitalni ulaz na pinu 17, koji je ustvari pritisak na analog stick.

DEFINISANJE POČETNOG EKRANA

Kako bi smo napravili ekran putem LVGL biblioteke potrebno definisati LVGL objekat i dodijeliti ga nekoj varijabli. U našem slučaju smo ga dodijelili varijabli *scr*. Dalje smo metodi *lv.label()* proslijedili naš *scr* objekat što je dodalo labelu koju ćemo koristiti kao naslov. Labeli stavljamo tekst “Connect4“ i pomjeramo 10 piksela vertikalno ka gore.

Dalje ćemo definistati dugma za singleplayer i multiplayer mod. Slično kao i za naslov, metodi *lv.button()* proslijedimo naš *scr* objekat. Ovo uradimo dva puta i dobijamo *btnSP* i *btnMP* dugme objekte koje pozicioniramo ispod “Connect4“ naslova.

Dalje, za prikazivanje teksta na *btnSP* i *btnMP* potrebno je definistati labele *labelSP* i *labelMP* koje se proslijeđuju buttonima preko *lv.label()* metode.

```
15 # Make screen object
16 scr = lv.obj()
17
18 # Create a title label
19 title = lv.label(scr)
20 title.set_text("Connect4")
21 title.align(lv.ALIGN.CENTER, 0, -10)
22
23 # Create the buttons
24 btnSP = lv.button(scr)
25 btnMP = lv.button(scr)
26 btnSP.align(lv.ALIGN.CENTER, -60, 30)
27 btnMP.align(lv.ALIGN.CENTER, 65, 30)
28 labelSP = lv.label(btnSP)
29 labelSP.set_text('SinglePlayer')
30 labelMP = lv.label(btnMP)
31 labelMP.set_text('Multiplayer')
```

3

ODABIR MODA

U svrhu odabira između singeplayer i multiplayer moda, inicijalizirana je globalna *selected_button_index* varijabla na 0. Kada je ova varijabla na 0 to znači da je trenutno u odabiru singleplayer mod, dok 1 znači da je u odabiru multiplayer mod.

Ovdje je i definisana *update_button_selection()* funkcija pomoću koje se čita pozicija joystick-a i pri tome se mijenja *selected_button_index* varijabla. Dalje se i u ovoj funkciji mijenja tekst da *btnSP* i *btnMP* dugmima u zavisnosti od *selected_button_index* varijable.

```
# Current selected button index
selected_button_index = 0

# Function to update button selection based on analog stick input
def update_button_selection():
    global selected_button_index,labelSP,labelMP

    # Read analog values from the analog stick (adjust these based on your analog stick's behavior)
    x_val = adc_x.read_u16()

    # Example logic: determine button selection based on analog stick position
    if x_val < 2000:
        selected_button_index = 0  # Select btnSP
    elif x_val > 60000:
        selected_button_index = 1  # Select btnMP

    # Highlight the selected button
    labelSP.set_text('SinglePlayer')  # Reset btnSP to plain style
    labelMP.set_text('MultiPlayer')  # Reset btnMP to plain style
    if selected_button_index == 0:
        labelSP.set_text('Select SP?')  # Highlight btnSP
    elif selected_button_index == 1:
        labelMP.set_text('Select MP?')  # Highlight btnMP

# Load screen
lv.screen_load(scr)
```

Ispod definicije funkcije se nalazi poziv za ucitavanje ekrana *lv.screen_load()*.

4

KLASA GAME

U ovom dijelu ćemo razmotriti klasu *Game* koju koristimo kako bi se definisala logika same igre Connect4.

U konstruktoru klase se prosljeđuje broj kolona (koji je po default 7), broj redova (koji je po default 6), broj potrebnih za pobjedu (koji je po default 4) i jedna bool *singleplayer* varijabla (koja je po default false). U samom kontruktoru se inicijalizira matrica *board,* koja prestavlja ploču na kojoj se igra. Na početku je matrica puna praznih polja tj. konstantom *NONE.*

```
153 class Game:
154     def __init__(self, cols=7, rows=6, requiredToWin=4, singleplayer=False):
155         """Create a new game."""
156         self.cols = cols
157         self.rows = rows
158         self.win = requiredToWin
159         self.singleplayer = singleplayer
160         self.board = [[NONE] * rows for _ in range(cols)]
161
```

Dalje u metodi *insert()* se prosljeđuje boja *color* i kolona *column* u koju se ta boja ubacuje. Ako je ta kolona već puna onda se baca izuzetak. Nakon što se *board* ažuira sa novom bojom, onda se provjerava da li ima pobjednika u metodi *checkForWin()*. Na kraju metode, ako je *singeplayer* true (tj. ako je igra u singleplayer modu) i proslijeđena boja je crvena onda se poziva metoda *ai_turn()*.

```
162     def insert(self, column, color):
163         """Insert the color in the given column."""
164         c = self.board[column]
165         if c[0] != NONE:
166             raise Exception('Column is full')
167
168         i = -1
169         while c[i] != NONE:
170             i -= 1
171         c[i] = color
172
173         self.checkForWin()
174
175         if self.singleplayer and color == 'R':
176             self.ai_turn()
```

5

Metoda *ai_turn()* radi na jednostavan način. Prvo prolazi kroz sve kolone i poziva funkciju *can_win()* kako bi provjerili da li može pobijediti u toj koloni, i ako može onda poziva metodu *insert()* kojoj prosljeđuje tu kolonu i žutu boju (jer žutu boju će uvijek AI koristiti).

Ako AI ne uspije pobijediti onda će probati blokirati igrača koji igra sa crvenom bojom tako što će slično kao u prvom koraku, provjeravati putem *can_win()* funkcije da li može u svakoj koloni crvena boja pobijediti. Ako igrač sa crvenom bojom može pobijediti onda se ponovno poziva metoda *insert()* sa tom kolonom i žutom bojom kako bi blokirao crvenu boju.

Dalje, AI će pokušati da stavi žutu boju u srednju kolonu kako bi se tabla "rastavila" na dva dijela.

Na kraju ako AI potroši sve ove opcije, pogledat će slobodne kolone, i staviti boju u random kolonu.

```
178     def ai_turn(self):
179         """Make the AI (yellow) move."""
180         # Try to win
181         for col in range(self.cols):
182             if self.can_win(col, YELLOW):
183                 self.insert(col, YELLOW)
184                 return
185 
186         # Try to block opponent's win
187         for col in range(self.cols):
188             if self.can_win(col, RED):
189                 self.insert(col, YELLOW)
190                 return
191 
192         # Play in the center column if possible
193         center_col = self.cols // 2
194         if self.board[center_col][0] == NONE:
195             self.insert(center_col, YELLOW)
196             return
197 
198         # Play a random move as a fallback
199         available_columns = [col for col in range(self.cols) if self.board[col][0] == NONE]
200         if available_columns:
201             chosen_column = random.choice(available_columns)
202             self.insert(chosen_column, YELLOW)
```

Sljedeće ćemo pokazati *can_win()* metodu koja samo pravi privremenu ploču kako bi testirali potez koji je AI napravio u metodi *ai_turn()*. Na kraju metode se poziva metoda *checkLineForWinner()* kojoj se prosljeđuje ova privremena ploča.
Metoda *checkLineWinner()* i ostale metode koje provjeravaju stanje ploče i da li je neki igrač pobjedio će biti objašnjene u nastavku.

```
204     def can_win(self, column, color):
205         """Check if playing in the column can result in a win for the color."""
206         # Create a temporary board to test the move
207         temp_board = [col[:] for col in self.board]
208         c = temp_board[column]
209         if c[0] != NONE:
210             return False
211 
212         i = -1
213         while c[i] != NONE:
214             i -= 1
215         c[i] = color
216 
217         return self.checkLineForWinner(c)
```

Metoda *checkForWin()* će prvo pozvati *getWinner()* da uzme potencijalnog pobjednika. U slučaju da ima pobjednika, pozvat će se metoda *printBoard()* (ova metoda će biti pojašnjena kasnije) koja prikazuje stanje ploče na ekranu i zadržava se jednu sekundu.
Ako je pobjednik crvena boja, baca se izuzetak sa porukom “RED won!“, a ako je žuta boja pobjednik onda je poruka “YELLOW won!“.

```
219     def checkForWin(self):
220         """Check the current board for a winner."""
221         w = self.getWinner()
222         if w:
223             self.printBoard(-1, NONE)
224             time.sleep(1)
225             if w=='R':
226                 raise Exception('RED won!')
227             else:
228                 w=='YELLOW'
229             raise Exception('YELLOW won!')
```

7

Metoda *getWinner()* stavlja sve moguće linije (redove, kolone i dijagonale) gdje se može pobjeda desiti i stavlja ih u tuple pod imenom *lines*.
Nakon toga se poziva metoda *flatten()* koja pretvara tuple u jedan generator koji daje jednu liniji (redove, kolone ili dijagonale).
Metoda *checkLineForWinner()* provjerava da li u liniji ima pobjednik. Ima brojač da broji uzastopna polja sa istom bojom i *last_color* da pamti koju je zadnju boju provjerio. Ako je trenutna boja jednaka posljednjoj i nije NONE onda se brojač inkrementira. Ako je brojač jednak potrebnom broju uzastopnih polja za pobjedu onda se vraća boja koja je pobjednik.

```
231     def getWinner(self):
232         """Get the winner on the current board."""
233         lines = (
234             self.board,  # columns
235             zip(*self.board),  # rows
236             diagonalsPos(self.board, self.cols, self.rows),  # positive diagonals
237             diagonalsNeg(self.board, self.cols, self.rows)  # negative diagonals
238         )
239 
240         for line in self.flatten(lines):
241             winner = self.checkLineForWinner(line)
242             if winner:
243                 return winner
244 
245     def flatten(self, lines):
246         """Flatten the lines generator of generators into a single generator."""
247         for line_group in lines:
248             for line in line_group:
249                 yield line
```

```
251      def checkLineForWinner(self, line):
252          """Check a single line (column, row, or diagonal) for a winner."""
253          count = 0
254          last_color = NONE
255          for color in line:
256              if color == last_color and color != NONE:
257                  count += 1
258                  if count >= self.win:
259                      return color
260              else:
261                  last_color = color
262                  count = 1
263          return None
```

8

```
143 def diagonalsPos(matrix, cols, rows):
144     """Get positive diagonals, going from bottom-left to top-right."""
145     for di in ([(j, i - j) for j in range(cols)] for i in range(cols + rows -1)):
146         yield [matrix[i][j] for i, j in di if i >= 0 and j >= 0 and i < cols and j < rows]
147 
148 def diagonalsNeg(matrix, cols, rows):
149     """Get negative diagonals, going from top-left to bottom-right."""
150     for di in ([(j, i - cols + j + 1) for j in range(cols)] for i in range(cols + rows - 1)):
151         yield [matrix[i][j] for i, j in di if i >= 0 and j >= 0 and i < cols and j < rows]
```

U metodi *printBoard()* koja prima parametre *position* (kolona u koju se ubacuje boja) i *turn* (boja koja se ubacuje na trenutnom potezu) se prikazuje na LCD ekranu stanje table.
Kao i na početku definišemo objekat *game_scr* koji će biti prikaz ekrana. Dalje ćemo definisati objekat *base_obj* koji će sadržavati našu tablu i ćelije u njoj. Kako bi smo obojili ćelije u tabli moramo definisati stilove različitih boja: *style* (prazna ćelija crne boje), *redStyle* (crvene ćelije), *yellowStyle* (žute ćelije), *selectRedStyle* (blijedo crvena boja prikazuje se na vrhu table da prikazuje u koju se kolonu ubacuje), *selectYellowStyle* (blijedo žuta boja prikazuje se na vrhu tabele da prikazuje u koju se kolonu ubacuje).

```
265     def printBoard(self, position, turn):
266         # Create a screen object for the game
267         game_scr = lv.obj()
268 
269         # Define the number of columns and rows for the grid
270         grid_cols = 7
271         grid_rows = 6
272         cell_size = 25
273         padding = 5
274 
275         # Calculate the correct size for the base object
276         base_width = (cell_size + padding) * grid_cols + 2 * padding + 5
277         base_height = (cell_size + padding) * grid_rows + 2 * padding + 5
278 
279         # Create a base object for the grid
280         base_obj = lv.obj(game_scr)
281         base_obj.set_size(base_width, base_height)
282         base_obj.align(lv.ALIGN.CENTER, 0, 20)
283 
284         # Create the grid style
285         style = lv.style_t()
286         style.init()
287         style.set_pad_all(padding)
288         style.set_bg_color(lv.color_hex(0x000000))
289         style.set_border_width(2)
290         style.set_border_color(lv.color_hex(0xFFFFFF))
```

9

```
        # Create the grid style (RED)
        redStyle = lv.style_t()
        redStyle.init()
        redStyle.set_pad_all(padding)
        redStyle.set_bg_color(lv.color_hex(0xFF0000))
        redStyle.set_border_width(2)
        redStyle.set_border_color(lv.color_hex(0xFFFFFF))

        # Create the grid style (YELLOW)
        yellowStyle = lv.style_t()
        yellowStyle.init()
        yellowStyle.set_pad_all(padding)
        yellowStyle.set_bg_color(lv.color_hex(0xFFFF00))
        yellowStyle.set_border_width(2)
        yellowStyle.set_border_color(lv.color_hex(0xFFFFFF))

        # Create the grid style (SELECTION RED)
        selectRedStyle = lv.style_t()
        selectRedStyle.init()
        selectRedStyle.set_pad_all(padding)
        selectRedStyle.set_bg_color(lv.color_hex(0xD99888))
        selectRedStyle.set_border_width(2)
        selectRedStyle.set_border_color(lv.color_hex(0xFFFFFF))

        # Create the grid style (SELECTION YELLOW)
        selectYellowStyle = lv.style_t()
        selectYellowStyle.init()
        selectYellowStyle.set_pad_all(padding)
        selectYellowStyle.set_bg_color(lv.color_hex(0xCCCC66))
        selectYellowStyle.set_border_width(2)
        selectYellowStyle.set_border_color(lv.color_hex(0xFFFFFF))
```

Dalje, metoda prolazi kroz *board* i na osnovu stanja dodaje ćelije sa stilovima koje smo definisali gore. Nakon ovoga se ekran učitava, poziva se garbage collection i *game_scr* se vraća iz metode.

10

```
        # Create grid cells
        for row in range(grid_rows):
            for col in range(grid_cols):
                cell = lv.obj(base_obj)
                cell.set_size(cell_size, cell_size)
                if col == position and row == 0 and (self.board[position][0] != 'R' and self.board[position][0] != 'Y'):
                    if(str(turn) == 'R'):
                        cell.add_style(selectRedStyle, lv.PART.MAIN)
                    if(str(turn) == 'Y'):
                        cell.add_style(selectYellowStyle, lv.PART.MAIN)
                    if(str(turn) == '.'):
                        cell.add_style(style, lv.PART.MAIN)
                elif(self.board[col][row] == 'R'):
                    cell.add_style(redStyle, lv.PART.MAIN)
                elif(self.board[col][row] == 'Y'):
                    cell.add_style(yellowStyle, lv.PART.MAIN)
                else:
                    cell.add_style(style, lv.PART.MAIN)
                cell.set_pos(col * (cell_size + padding), row * (cell_size + padding))
        lv.screen_load(game_scr)

        gc.collect()
        return game_scr
```

GLAVNA PETLJA I FUNKCIJE ZA MODOVE

U glavnoj programskoj petlji se svakom iteracijom poziva funkcija *update_button_selection()* kako bi se ažurirao odbair modova.

Ako je joystick pritisnut, tj *select* je 0, onda u zavisnosti od *selected_button_index* se pokreće *run_multiplayer_game() ili run_singleplayer_game()*.

Na kraju svake iteracije petlje se poziva garbage collection.

```
while True:
    update_button_selection()
    if(select.value()==0):
        while(select.value()==0):
            pass
        if selected_button_index == 0:
            run_singleplayer_game()
        elif selected_button_index == 1:
            run_multiplayer_game()

    # Trigger garbage collection
    gc.collect()
    time.sleep(0.1)
```

11

Razmotrimo sada funkciju *run_multiplayer_game()*. Kako bi mogli znati u kojoj se koloni nalazimo potrebno je držati tu poziciju u globalnoj varijabli *position*. Zatim u funkciju inicijaliziramo varijablu *scrCounter* na 0 i *screenArray* na prazan niz. Ove dvije varijable su nam potrebne jer pri svakoj promjeni table se pravi novi objekat koji se mora uništiti, inače će doći do curenja memorije jer će svaki objekat biti sadržan u memoriji. Napravit ćemo jedan objekat *Game* objekat u sklopu funkcije, inicijalizirati varijablu *turn* na RED i globalnu varijablu *position* ćemo postaviti na 0.

Zatim ulazimo u beskonačnu petlju gdje pri početku svake iteracije petlje prvo putem funkcije *column_selection()* ažuiramo *position*. Zatim dobavljamo ekran putem metode *Game.printBoard(),* nakon čega taj ekran stavljamo u *screenArray* i inkrementiramo *scrCounter*.

Nakon toga prolazimo kroz *screenArray* gdje brišemo sve ekrane putem *clean()* metode osim posljednjeg ekrana u nizu.

Ukoliko je pritisnut joystick onda se poziva *Game.insert()* gdje se prosljeđuje *position* i *turn,* nakon čega se mijenja *turn* (ukoliko je bio RED onda će biti YELLOW i obrnuto).

Ova petlja će se okončati ukoliko se uhvati izuzetak koji smo ranije definisali u *Game* klasi. Ako se izuzetak uhvati onda se poziva funkcija *show_winner_screen()* kojoj se prosljeđuje tekst izuzetka.

Funkcija *run_singleplayer_game()* radi na sličan način kao i prethodna, sa malim izmjenama gdje objektu *Game* stavljamo parametar *singleplayer* na true.

12

```
def run_multiplayer_game():
    global select,position
    print("Starting Multiplayer game...")
    scrCounter = 0
    screenArray = []
    g = Game()
    turn = RED
    position = 0
    try:
        while True:
            position = column_selection()
            screen=g.printBoard(position,turn)
            screenArray.append(screen)
            scrCounter+=1
            for i in range(scrCounter - 1):
                screenArray[i].clean()
            if select.value()==0:
                g.insert(position, turn)
                while select.value()==0:
                    pass
                turn = YELLOW if turn == RED else RED
            time.sleep(0.1)
    except Exception as e:
        show_winner_screen(str(e).split()[0])
```

```
def run_singleplayer_game():
    global select,position
    print("Starting SinglePlayer game...")
    scrCounter = 0
    screenArray = []
    g = Game(singleplayer=True)
    turn = RED
    position = 0
    try:
        while True:
            position = column_selection()
            screen=g.printBoard(position,turn)
            screenArray.append(screen)
            scrCounter+=1
            for i in range(scrCounter - 1):
                screenArray[i].clean()
            if select.value()==0:
                g.insert(position, turn)
                while select.value()==0:
                    pass
                turn = YELLOW if turn == RED else RED
                if g.singleplayer and turn == YELLOW:
                    turn = RED
            time.sleep(0.1)
    except Exception as e:
        show_winner_screen(str(e).split()[0])
```

13

Još je ostalo pokazati *column_selection()* i *show_winner_screen()* funkcije. U *column_selection()* se jednostavno ažuira globalna varijabla *position* na osnovu pomjeranja joystick-a i vraća je iz funkcije.

Funkcija *show_winner_screen()* pravi novi ekran koji samo prikazuje tekst koji je proslijeđen funkciji kao parametar. Ovaj ekran se zadržava 5 sekundi nakon čega se vraća na originalni ekran gdje korisnik bira koji mod želi igrati.

```
62 def column_selection():
63     global adc_x, select, position
64     # Read analog values from the analog stick (adjust these based on your analog stick's behavior)
65 
66     # Example logic: determine button selection based on analog stick position
67     if adc_x.read_u16() < 2000 and position > 0:
68           position-=1
69           while adc_x.read_u16()<2000:
70               pass
71     if adc_x.read_u16() > 60000 and position <6:
72           position+=1
73           while adc_x.read_u16()>60000:
74               pass
75     return position
76 
77 def show_winner_screen(winner):
78     global scr
79     scr_winner = lv.obj()
80     label = lv.label(scr_winner)
81     label.set_text(f"{winner} won!")
82     label.align(lv.ALIGN.CENTER, 0, 0)
83     lv.screen_load(scr_winner)
84     time.sleep(5)
85     lv.screen_load(scr)
```