PiPhone – telefon komórkowy z Raspberry PI

Jakiś czas temu pojawiła się chwilowa moda na korzystanie ze starych telefonów komórkowych zamiast nowoczesnych smartfonów, celem podkreślenia swojej indywidualności. Nie polemizujemy z tą modą, ale dotarliśmy do projektu telefonu, który zdecydowanie wyróżniłby w tłumie miłośnika alternatywnych rozwiązań – samodzielnie wykonany telefon z Raspberry PI. I nawet, jeśli Czytelnicy nie pałają potrzebą pokazania się w towarzystwie z takimi gadżetami, jest to świetny przykład jak stworzyć nowoczesne, przenośne urządzenie i zaimplementować komunikację GSM z użyciem popularnego komputerka jednopłytkowego.

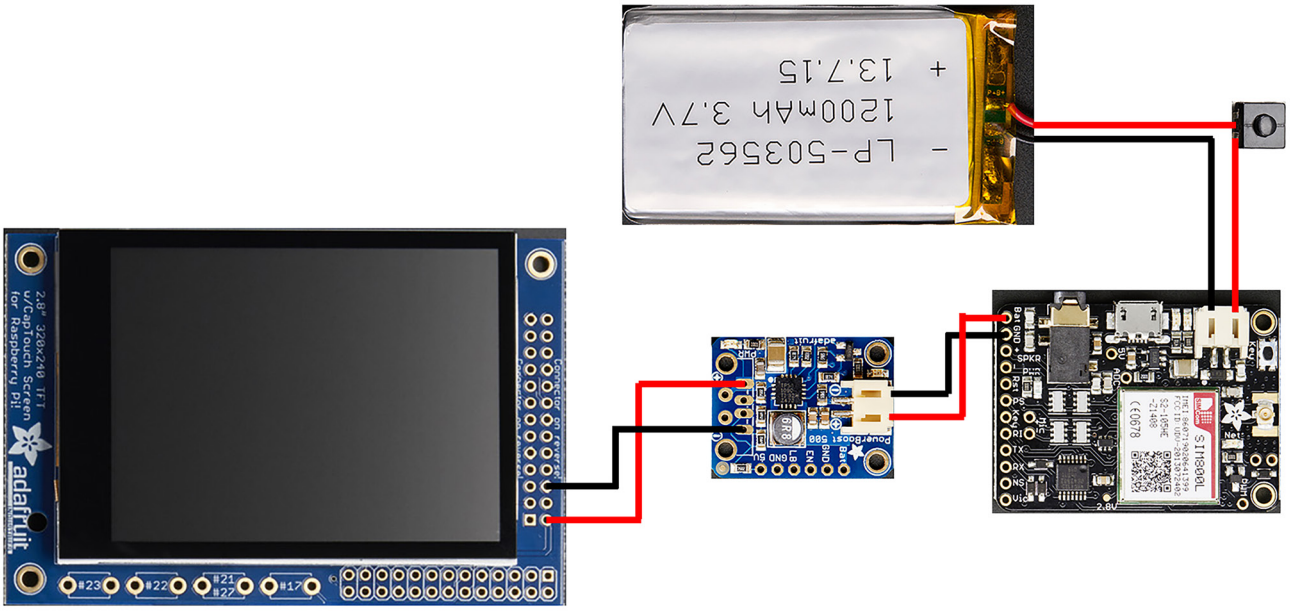
Omawiany projekt został wykonany przez Davida Hunta z Irlandii. Bazą wykonanego telefonu jest komputer jednopłytkowy Raspberry Pi Model B, choć inne modele też mogą zostać użyte. Model A pozwoli np. na obniżenie poboru energii. Interfejs użytkownika zrealizowano na wyświetlaczu ciekłokrystalicznym TFT o rozdzielczości 320×240 pikseli, dostępnym w postaci modułu dla Raspberry PI. Wyświetlacz ma ekran dotykowy, przy czym nie ma większego znaczenia, czy zastosuje się model z interfejsem rezystancyjnym, czy dotykowym. Ten drugi będzie droższy, ale pozwoli łatwiej zrealizować zaawansowane interakcję z użytkownikiem, gdyby zaszła taka potrzeba. Idealny rozmiar wyświetlacza, pasującego do omawianej konstrukcji to 2,8". Autor wybrał model PiTFT firmy Adafruit.

Do komunikacji z siecią komórkową użyto modułu GSM/GPRS. W pierwotnej wersji projektu autor skorzystał z modelu SIM900 firmy ElecFreaks, w której znajdował się układ SimCom SIM900. W nowszej rewizji projektu użyto zminiaturyzowanego modułu Adafruit FONA, w którym zainstalowany był modem SimCom SIM800L. W tym drugim przypadku konieczne jest jeszcze dokupienie anteny.

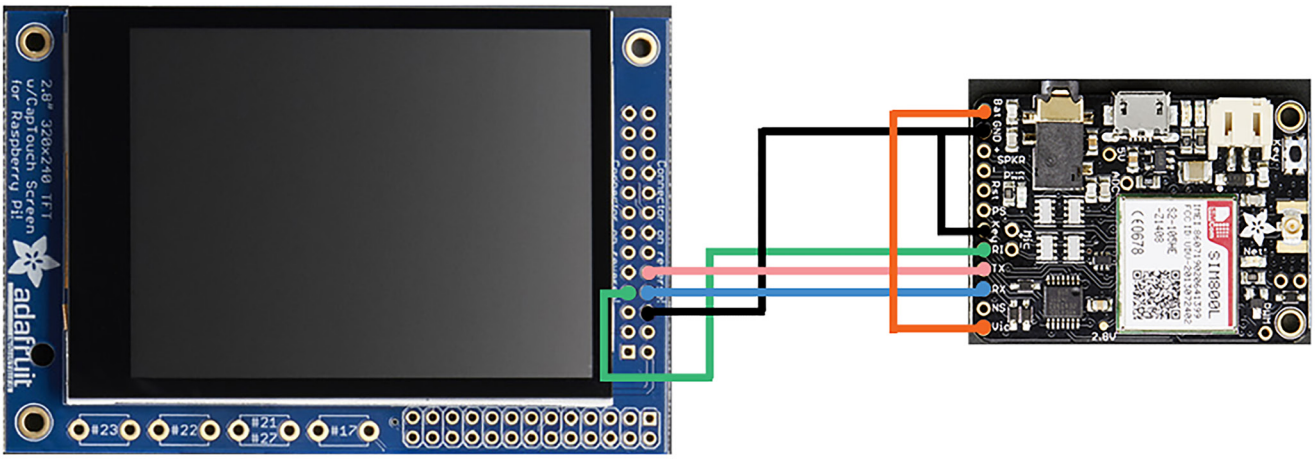
Aby zapewnić przenośność telefonu, konieczne było użycie akumulatora. Wybór padł na model litowo-polimerowy o pojemności 2500 mAh. W wersji usprawnionej, aby zmniejszyć całkowite wymiary sprzętu i zapewnić lepszą wentylację użyto mniejszego akumulatora litowo-polimerowego o pojemności 1200 mAh. Ponieważ Raspberry Pi wymaga zasilania napięciem 5 V, a akumulator dostarcza 3,7 V, potrzebna była przetwornica podwyższająca napięcie. I znowu – pierwotnym wyborem był chiński podzespół, ale docelowo użyto przetwornicy Adafruit PowerBoost 500 z układem Texas Instruments TPS61090.

Potrzebne komponenty:

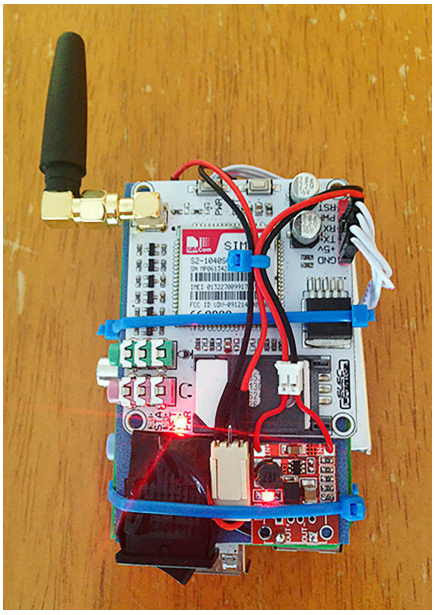
- Komputer Raspberry PI Model B; możliwość zastosowania innego modelu; koszt: 20-35 USD.
- Moduł wyświetlacza 2,8" Adafruit PiTFT z ekranem rezystancyjnym; koszt 35 USD, a w wersji z ekranem pojemnościowym – 45 USD.
- Akumulator litowo-polimerowy, 2500 mAh lub 1200 mAh (koszt: 10-15 USD).
- Moduł GSM z układem SimCom lub moduł Adafruit SONA + antena; koszt ok. 50 USD.
- Przetwornica DC/DC podwyższająca napięcie; koszt: 10 USD.
- Karta pamięci SD 4 GB; koszt: 3 USD.
- Zestaw słuchawkowy lub mały głośnik i mikrofon oraz przewody i konektory.



Rysunek 1. Sposób podłączenia zasilania w zmodernizowanej wersji projektu



Rysunek 2. Pozostałe połączenia w zmodernizowanej wersji projektu



Fotografia 3. Pierwotna wersja projektu od spodu. Problematyczne są goldpiny modułu GSM

Listing 1. Importowane biblioteki

```
import atexit
import cPickle as pickle
import errno
import fnmatch
import io
import os
import pygame
import threading
from pygame.locals import *
from subprocess import call
from time import sleep
from datetime import datetime,
timedelta
import serial
```

Listing 2. Klasy potrzebne do obsługi elementów interfejsu

```
class Icon:
    def __init__(self, name):
    self.name = name
    try:
        self.bitmap = pygame.image.load(iconPath + '/' + name + '.png')
    except:
    pass

class Button:
    def __init__(self, rect, **kwargs):
    self.rect     = rect
    self.color    = None
    self.iconBg   = None
    self.iconFg   = None
    self.bg       = None
    self.fg       = None
    self.callback = None
    self.value    = None
    for key, value in kwargs.iteritems():
        if   key == 'color': self.color    = value
        elif key == 'bg'   : self.bg       = value
        elif key == 'fg'   : self.fg       = value
        elif key == 'cb'   : self.callback = value
        elif key == 'value': self.value    = value

    def selected(self, pos):
        x1 = self.rect[0]
        y1 = self.rect[1]
        x2 = x1 + self.rect[2] - 1
        y2 = y1 + self.rect[3] - 1
        if ((pos[0] >= x1) and (pos[0] <= x2) and
            (pos[1] >= y1) and (pos[1] <= y2)):
        if self.callback:
            if self.value is None: self.callback()
            else:                  self.callback(self.value)
            return True
        return False

    def draw(self, screen):
        if self.color:
        screen.fill(self.color, self.rect)
        if self.iconBg:
        screen.blit(self.iconBg.bitmap,
            (self.rect[0]+(self.rect[2]-self.iconBg.bitmap.get_width())/2,
             self.rect[1]+(self.rect[3]-self.iconBg.bitmap.get_height())/2))
        if self.iconFg:
            screen.blit(self.iconFg.bitmap,
            (self.rect[0]+(self.rect[2]-self.iconFg.bitmap.get_width())/2,
             self.rect[1]+(self.rect[3]-self.iconFg.bitmap.get_height())/2))

    def setBg(self, name):
        if name is None:
            self.iconBg = None
        else:
            for i in icons:
            if name == i.name:
            self.iconBg = i
            break
```

Oczywiście potrzebna była jeszcze karta SD o pojemności przynajmniej 4 GB, z systemem operacyjnym oraz elementy montażowe. Warto też zwrócić uwagę na fakt, że Raspberry Pi nie ma wbudowanego głośnika i dlatego jest konieczne zastosowanie np. zestawu słuchawkowego.

Montaż podzespołów

Trzon konstrukcji to Raspberry Pi oraz płytka wyświetlacza ciekłokrystalicznego. Autor umieścił je – jedną na drugiej, a w przestrzeń pomiędzy złączami RPi wstawił wspomniany akumulator. Komponenty są zamocowane za pomocą golpinów, do których jest dołączony moduł wyświetlacza. Odwzorowując samodzielnie tę konstrukcję warto zwrócić uwagę na przewiew powietrza pomiędzy płytkami. Nieco nagrzewają się one w trakcie pracy, co podwyższa temperaturę akumulatora, prawdopodobnie nie wpływając korzystnie na jego trwałość. Zamontowanie dodatkowego wentylatora może poprawić niezawodność urządzenia.

Moduł z modemem został umieszczony od spodu Raspberry Pi. Aby odizolować go od PCB komputera, autor odseparował go grubą, twardą gąbką. Komponenty trzymają się siebie dzięki plastikowym taśmom zaciskowym, co nie jest rozwiązaniem eleganckim, ale w prototypie jest dopuszczalne. Obok modemu doczepiono przetwornicę, a całość połączono przewodami, zgodnie z **rysunkami 1** i **2**. Raspberry Pi jest dołączony

Listing 3. Funkcja umożliwiająca reagowanie na zdarzenie wywoływane w momencie naciśnięcia przycisku na ekranie dotykowym

```
def numericCallback(n):
    global screenMode
    global numberstring
    global phonecall
    if n < 10 and screenMode == 0:
        numberstring = numberstring + str(n)
    elif n == 10 and screenMode == 0:
        numberstring = numberstring[:-1]
    elif n == 12:
        if screenMode == 0:
            if len(numberstring) > 0:
                print(„Calling „ + numberstring);
                serialport.write(„AT\r”)
                response = serialport.readlines(None)
                serialport.write(„ATD „ + numberstring + ';\r')
                response = serialport.readlines(None)
                print response
                screenMode = 1
        else:
            print(„Hanging Up...”)
            serialport.write(„AT\r”)
            response = serialport.readlines(None)
            serialport.write(„ATH\r”)
            response = serialport.readlines(None)
            print response
            screenMode = 0
        if len(numberstring) > 0:
            numeric = int(numberstring)
            v[dict_idx] = numeric
```

Listing 4. Zmienne globalne zadeklarowane w programie

```
busy            = False
threadExited    = False
screenMode      =  0
phonecall       =  1
screenModePrior = -1
iconPath        = 'icons'
numeric         = 0
numberstring    = „”
returnScreen    = 0
dict_idx        = „Interval”
v = { „Pulse”: 100,
    „Interval”: 3000,
    „Images”: 150}
icons = []
```

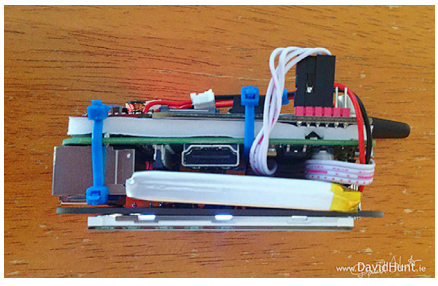
Fotografia 6. Front telefonu w wersji pierwotnej

do wyświetlacza, a wiec także i do zasilacza oraz modemu, poprzez goldpiny.

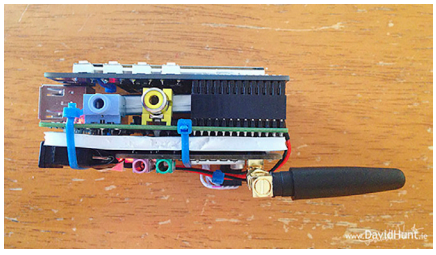
Oprogramowanie

W pierwszej kolejności należy zainstalować Raspbiana, a następnie skonfigurować obsługę wyświetlacza z ekranem dotykowym, zgodnie z instrukcją do wybranego modelu. Czasem dostawcy wyświetlaczy do Raspberry Pi dostarczają własne, już skonfigurowane wersje Raspbiana w postaci obrazów. Warto poszukać ich na stronie producenta.

Podczas pierwszego uruchomienia systemu, warto podłączyć monitor, klawiaturę i myszkę do przejścia przez procedurę konfiguracyjną. Ponieważ urządzenie będzie pracować z akumulatorem, lepiej zrezygnować z opcji przetaktowania komputera, by nie czerpał on zbyt wiele energii.



Fotografia 4. Sposób umieszczenia akumulatora w pierwotnej wersji projektu



Fotografia 5. Pierwotna wersja PiPhone z boku

Ponieważ domyślnie wyświetlacz telefonu ma pracować w orientacji portretowej, należy przygotować go do tego trybu. W przypadku wyświetlacza w postaci modułu PiTFT polega to na edytowaniu pliku **/etc/modprobe.d/adafruit.conf** i przypisaniu do znajdującego się wewnątrz parametru **rotate** wartości 0.

O ile dodatkowe przyciski, znajdujące się na PCB modułu PiTFT nie są w niniejszym projekcie potrzebne, należy za to przeprowadzić kalibrację ekranu dotykowego.

W ramach konfiguracji trzeba też sprawić, by port szeregowy komputera komunikował się z modułem GSM. W tym celu w pliku **/etc/inittab** należy wykomentować linijkę:

#T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100

Właściwy program

Kod programu PiPhone został napisany w Pythonie i nie jest skomplikowany. Dzięki

Listing 5. Deklaracja pozycji, wyglądu i funkcji przycisków

```
buttons = [
   [Button(( 30,  0,320, 60), bg='box'),
    Button(( 30, 60, 60, 60), bg='1',     cb=numericCallback, value=1),
    Button(( 90, 60, 60, 60), bg='2',     cb=numericCallback, value=2),
    Button((150, 60, 60, 60), bg='3',     cb=numericCallback, value=3),
    Button(( 30,110, 60, 60), bg='4',     cb=numericCallback, value=4),
    Button(( 90,110, 60, 60), bg='5',     cb=numericCallback, value=5),
    Button((150,110, 60, 60), bg='6',     cb=numericCallback, value=6),
    Button(( 30,160, 60, 60), bg='7',     cb=numericCallback, value=7),
    Button(( 90,160, 60, 60), bg='8',     cb=numericCallback, value=8),
    Button((150,160, 60, 60), bg='9',     cb=numericCallback, value=9),
    Button(( 30,210, 60, 60), bg='star',  cb=numericCallback, value=0),
    Button(( 90,210, 60, 60), bg='0',     cb=numericCallback, value=0),
    Button((150,210, 60, 60), bg='hash',  cb=numericCallback, value=0),
    Button((180,260, 60, 60), bg='del2',  cb=numericCallback, value=10),
    Button(( 90,260, 60, 60), bg='call',    cb=numericCallback, value=12)],
    [Button(( 30,  0,320, 60), bg='box'),
    Button(( 90,260, 60, 60), bg='hang',    cb=numericCallback, value=12)]
]
```

Listing 6. Funkcje do zapisywania i wczytywania ustawień

```
def saveSettings():
    global v
    try:
        outfile = open('piphone.pkl', 'wb')
        pickle.dump(v, outfile)
        outfile.close()
    except:
        pass

def loadSettings():
    global v
    try:
        infile = open('piphone.pkl', 'rb')
        v = pickle.load(infile)
        infile.close()
    except:
        pass
```

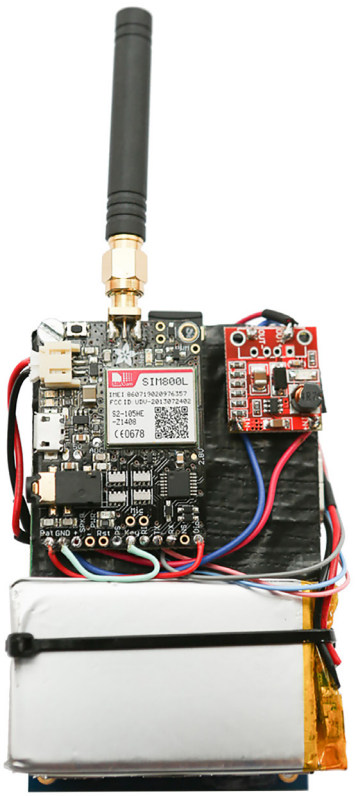
wykorzystaniu wielu bibliotek, całość sprowadza się do około 300 linijek, po usunięciu komentarzy.

Na początek kodu importowane są biblioteki (**listing 1**). Najbardziej pomocna jest biblioteka **pygame**, która pozwala w łatwy sposób obsłużyć wyświetlacz oraz wyjścia i wejście audio. Definiowana jest klasa **Icon** (**listing 2**), która pozwala ładować pliki PNG jako obrazy z katalogu z ikonami. Autor zdecydował, że lista nie jest definiowana globalnie, tylko ładowana dynamicznie. Drugą z definiowanych klas jest Button, która określa region ekranu oraz opcjonalnie jego kolor tła, wyświetlaną w jego obszarze ikonę, funkcję wywoływaną po przytrzymaniu lub naciśnięciu danego regionu, a także parametr przekazywany tej funkcji. Dzięki temu wszystkie przyciski korzystają z tej samej funkcji, a jedynie parametr określa, który przycisk został naciśnięty. Warto zwrócić uwagę, że jeśli przyciski będą na siebie nachodzić, istotna jest ich kolejność wpływająca na to, który z przycisków został naciśnięty.

Na **listingu 3**. Przedstawiono treść funkcji reagującej na naciśnięcie przycisku. Po pierwsze sprawdza ona, czy przycisk jest cyfrą, a po drugie, czy aktualnie trwa rozmowa. Naciśnięcie klawisza numerycznego wydłuża ciąg cyfr wybieranego numeru telefonu. Przycisk cofania skraca ciąg. Przycisk słuchawki służy do nawiązywania lub rozłączania połączenia. Polecenie wybrania numeru lub rozłączenia wysyłane jest poprzez port szeregowy za pomocą komend AT. Przykładowo, do nawiązania połączenia służy linijka

serialport.write(„ATD „ + numberstring + ';\r')

Autor przygotował zestaw zmiennych globalnych (**listing 4**), do których odnoszą się poszczególne funkcje. Zmienne te przechowują m.in. informacje o tym, czy prowadzona jest aktualnie rozmowa, który katalog zawiera ikony przycisków oraz który przycisk został ostatni naciśnięty. Jako zmienne globalne zadeklarowano także tablicę ikon i przycisków (**listing 5**), tworzoną w oparciu o podane wcześniej klasy.



Fotografia 7. Tył PiPhone w wersji zmodyfikowanej

Listing 7. Inicjalizacja programu

```
os.putenv('SDL_VIDEODRIVER', 'fbcon')
os.putenv('SDL_FBDEV'      , '/dev/fb1')
os.putenv('SDL_MOUSEDRV'   , 'TSLIB')
os.putenv('SDL_MOUSEDEV'   , '/dev/input/touchscreen')

print „Initting..."
pygame.init()
print „Setting Mouse invisible..."
pygame.mouse.set_visible(False)
print „Setting fullscreen..."
modes = pygame.display.list_modes(16)
screen = pygame.display.set_mode(modes[0], FULLSCREEN, 16)

print „Loading Icons..."
for file in os.listdir(iconPath):
    if fnmatch.fnmatch(file, '*.png'):
    icons.append(Icon(file.split('.')[0]))
print"Assigning Buttons"
for s in buttons:
    for b in s:
        for i in icons:
            if b.bg == i.name:
            b.iconBg = i
            b.bg     = None
        if b.fg == i.name:
            b.iconFg = i
            b.fg     = None

print"Load Settings"
loadSettings()

print „loading background.."
img    = pygame.image.load(„icons/PiPhone.png")

if img is None or img.get_height() < 240:
    screen.fill(0)
if img:
    screen.blit(img,
        ((240 - img.get_width() ) / 2,
        (320 - img.get_height()) / 2))
pygame.display.update()
sleep(2)

print „Initialising Modem.."
serialport = serial.Serial(„/dev/ttyAMA0", 115200, timeout=0.5)
serialport.write(„AT\r")
response = serialport.readlines(None)
serialport.write(„ATE0\r")
response = serialport.readlines(None)
serialport.write(„AT\r")
response = serialport.readlines(None)
print response
```

Listing 8. Główna pętla programu

```
print „mainloop.."
while(True):
    while True:
        screen_change = 0
        for event in pygame.event.get():
            if(event.type is MOUSEBUTTONDOWN):
                pos = pygame.mouse.get_pos()
        for b in buttons[screenMode]:
            if b.selected(pos): break
        screen_change = 1

    if screen_change == 1 or screenMode != screenModePrior: break
    if img is None or img.get_height() < 240:
        screen.fill(0)
    if img:
        screen.blit(img,
            ((240 - img.get_width() ) / 2,
             (320 - img.get_height()) / 2))

    for i,b in enumerate(buttons[screenMode]):
        b.draw(screen)
    if screenMode == 0 :
        myfont = pygame.font.SysFont(„Arial", 40)
        label = myfont.render(numberstring, 1, (255,255,255))
        screen.blit(label, (10, 2))
    else:
        myfont = pygame.font.SysFont(„Arial", 35)
        label = myfont.render(„Calling", 1, (255,255,255))
        screen.blit(label, (10, 80))
        myfont = pygame.font.SysFont(„Arial", 35)
        label = myfont.render(numberstring + „...", 1, (255,255,255))
        screen.blit(label, (10, 120))

    pygame.display.update()
    screenModePrior = screenMode
```

Na **listingu 6** można zobaczyć funkcje służące do zapamiętywania i odczytywania ustawień.

W ramach inicjalizacji programu (**listing 7**) skrypt konfiguruje zmienne środowiskowe dla ekranu dotykowego, a następnie uruchamia bibliotekę **pygame**. W następnej części ładuje ikony przycisków, wypełniając zadeklarowaną wcześniej globalną tablicę. Ładuje też dodatkowe ustawienia i tło wyświetlacza, a następnie inicjalizuje modem GSM. W tym celu również korzysta z poleceń AT przesyłanych przez port szeregowy.

Główna pętla programu (**listing 8**) nie jest długa. Przegląda ona stale listę zdarzeń zarejestrowanych za pomocą biblioteki **pygame** i jeśli znajdzie na niej zdarzenie typu **MOUSEBUTTONDOWN**, sprawdza dla każdego przycisku, czy obszar zdarzenia (pozycja kursora) nie leży czasem w zasięgu danego przycisku. Jeśli tak jest, program przechodzi do odtworzenia poprawnego wyglądu ekranu – ponownego wyświetlenia tapety, wypisania wybieranego numeru lub np. narysowania na nowo przycisków. Sama obsługa przycisków jest bowiem realizowana za pomocą zdarzeń wywoływanych niezależnie.

Gotowy program w języku Python należy dopisać do skryptu inicjacji komputera, edytując plik **/etc/rc.local** i dodając treść:

```
cd/home/pi/PiPhone-master
python piphone.py
```

przed linijką **exit 0**.

Podsumowanie

Omawiany projekt jest bardzo dobrym przykładem tego, jak łatwe jest zrobienie bardzo zaawansowanego urządzenia, jeśli korzysta się z gotowych modułów i bibliotek. Kto by pomyślał, że telefon komórkowy z interfejsem dotykowym wymaga jedynie 300 linijek kodu i połączenia ze sobą maksymalnie kilkunastu punktów lutowniczych. A ponadto, znacząca część napisanego kodu służy przede wszystkim wyświetlaniu przycisków i informacji na ekranie.

Aby telefon faktycznie był użyteczny, należałoby jeszcze poprawić jego konstrukcje mechaniczną, by się nie rozpadał i zabezpieczyć go przed warunkami środowiskowymi. Najlepiej poprzez zaprojektowanie dla niego obudowy z tworzywa i wydrukowanie jej na drukarce 3D. Można też wbudować mikrofon i miniaturowy głośnik, korzystając choćby z dodatkowych interfejsów modułu FONA, użytego w zmodernizowanej wersji projektu. Niestety, samodzielna budowa telefonu nie jest sposobem na zaoszczędzenie – łączny koszt użytych komponentów to około 150 dolarów + podatki.

Kompletny gotowy program można pobrać z adresu *https://goo.gl/buWC7D*, a opis zmodernizowanego projektu znajduje się tu *https://goo.gl/3lFHZ1*. Dla ułatwienia komplet skryptów można pobrać bezpośrednio do Raspberry Pi za pomocą komend:

```
wget https://github.com/climberhunt/Piphone/archive/master.zip
unzip master.zip
```

Spowodują one pobranie i rozpakowanie pliku z zasobami projektu, przy czym tamtejszy kod zawiera dodatkowe komentarze i linijki z starszych projektów autora, które nie są potrzebne do działania PiPhone i dla skrócenia listingów zostały w niniejszym artykule z nich usunięte.

Marcin Karbowniczek, EP