UNIVERSITATEA „ALEXANDRU IOAN CUZA” DIN IAŞI
FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENŢĂ

Morseus

Propusă de

Cosmin Poieană

Sesiunea: **Iulie, 2018**

Coordonator ştiinţific

Lector, Dr. Alex Moruz

UNIVERSITATEA „ALEXANDRU IOAN CUZA” DIN IAŞI
FACULTATEA DE INFORMATICĂ

Morseus

Traducător de semnale luminoase Morse

Cosmin Poieană

Sesiunea: **Iulie, 2018**

Coordonator ştiinţific

Lector, Dr. Alex Moruz

Avizat,

Îndrumător Lucrare de Licență

Titlul, Numele și prenumele _____

Data _____ Semnătura _____

DECLARAȚIE privind originalitatea conținutului lucrării de licență

Subsemntatul(a) .....

domiciliul în .....

născut(ă) la data de ....., identificat prin CNP ....., absolvent(a) al(a) Universității „Alexandru Ioan Cuza" din Iași, Facultatea de..... specializarea ....., promoția....., declar pe propria răspundere, cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr.1/2011 art.143 al. 4 si 5 referitoare la plagiat, că lucrarea de licență cu
titlul:_____
_____
_____
_____elaborată sub îndrumarea dl. / d-na
_____, pe care urmează să o susțină în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului său într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice in vederea facilitării fasificării de către cumpărător a calității de autor al unei lucrări de licență, de diploma sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Dată azi, ... Semnătură student .....

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „*Morseus - Traducător de semnale luminoase Morse*”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, *24.06.2018*

Absolvent *Cosmin Poieană*

_____

(semnătura în original)

Cuprins

Introducere

Motivație

Practică

Cu toții cunoaștem cât de importantă este comunicarea atât în viața de zi cu zi a fiecărei persoane, cât și în orice fel de domeniu de activitate al acesteia. Fie că e vorba de un simplu curs la școală sau acasă, fie că lucrezi în serviciul militar, naval sau ești un turist dornic de aventură ce explorează necunoscutul până și în cele mai puțin frecventate locuri de om, întotdeauna comunicarea stă la baza oricărei interacțiuni și poate face diferența dintre viață și moarte în momentul în care suntem în pericol, de aceea am ales să aduc un plus acestui studiu de caz, prin a facilita acest tip de comunicare neconvențional (semnale Morse), atunci când putem fi surprinși neplăcut de absența semnalului GSM.

Tehnică

Într-adevăr, este o șansă de 1 la un milion să te afli într-un pericol iminent, captiv pe vârful unui munte spre exemplu, eventual să fii în incapacitatea de a ajunge la timp la bază (rănit), să nu ai semnal GSM la telefon, dar încă să mai ai baterie (sau vreun dispozitiv electric, încărcat, capabil să ruleze software) și să încerci a comunica cu orice preț cu o bază sau persoană aflată în proximitatea câmpului vizual, prin semnale Morse fără a cunoaște acest limbaj. Această tentativă de comunicare s-ar putea transforma totuși într-o reușită în momentul în care vei scoate dispozitivul și vei rula aplicația *Morseus*, capabilă atât de trivialitatea de a converti textul în semnale luminoase Morse, cât și de posibilitatea decodării în timp real a acestora (emise chiar și cu o lanternă), primite de la emițător (și posibil viitor salvator) și captate cu ajutorul camerei dispozitivului.

Cu toate acestea, motivația principală mi-am regăsit-o prin însăși domeniul de cercetare și suita de tehnologii la care m-a împins implementarea acestei soluții, fapt ce m-a ajutat să aprofundez și să dobândesc cunoștințe noi în domeniul științei datelor, dar și ingineriei software în general, atât pe platforme desktop cât și mobile.

Context

Scopul aplicației

Scopul este simplu: acela de a putea descifra în timp real semnale Morse trimise de către om, indiferent de mijlocul de generare al acestora, fără vreun fel de calibrare necesară în prealabil. Scopul secund este de a putea trimite înapoi semnale (convertite din text latin) și a purta astfel o conversație cu un interlocutor ce se folosește de orice fel de sursă coerentă de trimis semnale Morse, fie aceasta o lanternă, o aplicație similară sau de ce nu, chiar *Morseus*. Interesant de precizat este faptul că semnalele trimise de către *Morseus* sunt generate la aceeași rată cu care au fost primite și analizate în trecut, astfel creându-se un set de parametri de comunicare pe înțelesul interlocutorului, vorbind cu acesta în același ritm.

Domeniu de cercetare

Motivația mea principală a plecat de la faptul că acest utilitar și lucrare abordează mai multe domenii de cercetare: învățare automată, procesare de imagini, abilități avansate de inginerie a programării, bune practici ale ei, algoritmică, structuri de date, programare asincronă, orientată pe obiecte și în final testare, controlul calității, documentare și controlul versiunii.
Toată această muncă cu sursă liberă poate fi văzută ca pe un produs final, dezvoltat cu profesionalism încă din primele stagii de cercetare, până în momentul lansării finale, reprezentate de aplicația desktop multi-platformă sau cea mobile din Play Store (Android).

Aplicații similare și avantaje

Piața *mobile* este plină de astfel de aplicații, dar asta nu m-a împiedicat să caut și să observ că aproape toate dintre acestea (cel puțin cele gratuite) nu sunt pregătite să fie folosite din prima, fără vreun fel de învățare sau calibrare manuală, realizată în prealabil. De multe ori acestea depind de anumite condiții și setări prestabilite atât de emițător cât și receptor la momentul folosirii. Combinații din **problemele** listate mai jos se regăsesc în majoritatea aplicațiilor existente de acest gen:

- Necesită calibrare în prealabil: trimiterea unor semnale de test asociate unui text deja cunoscut în avans.
- Semnale fixe/”robotice”: faptul că semnalele primite nu trebuie să varieze în lungime și viteză, iar durata unității (cea mai mica particulă indivizibilă de (absență de) semnal) trebuie să rămână fixă pe toată durata conversației, altfel se produc erori ce duc la un text neinteligibil.
- Sunt afectate de zgomotul (*noise*) din jur (alte surse de lumină secundare ce nu fac parte din mijlocul principal dorit de trimitere a unor astfel de semnale).
- Incapacitatea de a corecta automat erorile de transmisie umane și de a face predicții din ele, pentru a salva și rectifica acele porțiuni eronate din conversație.
- Necesitatea de a apăsa un buton de *start* pentru a inițializa procesul de primire al semnalelor
- Comportament nedorit în momentul în care există pauze mari (mai mult de 7 unități de absența luminoasă), lucru ce duce la deteriorarea conversației ce va urma.
- Incompatibilitatea cu alte platforme.

Eu mi-am propus astfel să rezolv toate aceste probleme printr-o aplicație extrem de ușor de folosit, ce necesită doar deschiderea ei și îndreptarea camerei către sursa de lumină în cauză.

Cerințe funcționale

Evident, cerințele sunt minime, unde singura grijă a utilizatorului este sa aibă această aplicație deja instalată. Ea nu necesită crearea unui profil, conexiune la internet sau vreun fel de drept special, altul în afară de accesul la camera video și a blițului.

Caracteristicile ar fi următoarele:

1. Posibilitatea de a utiliza aplicația direct, fără crearea vreunui cont sau configurări necesare.
2. Folosirea ei fără a avea nevoie de conexiune la internet sau vreun fel de bază de date.
3. Ușurința de a naviga între cele trei meniuri disponibile: *Send*, *Receive* și *Options* (pentru a trimite, respectiv a primi și decodifica astfel de semnale, dar și setările aferente).
4. Avantajul de a scrie text într-o casetă de intrare care prin apăsarea butonului **Send** se va trimite sub formă de semnale luminoase interlocutorului, pe înțelesul acestuia.
5. Simplitatea de a primi și decodifica în timp real semnale luminoase, captate cu ajutorul camerei, în momentul deschiderii aplicației și al accesării în mod automat al panoului pentru recepționat semnale.
6. Configurarea opțională a unor preferințe de trimitere a semnalelor, unde primirea acestora se ajustează automat în funcție de toți factorii principali luați în considerare.

Abordare tehnică

Aplicația în discuție este formată din două mari componente decuplate și de o coeziune maximă, pentru a asigura buna funcționare și testare a ei ca pe un tot unitar. Aceste două componente esențiale ei sunt:

- **libmorse**: https://github.com/cmin764/libmorse (bibliotecă, utilitare, apeluri)
- **Morseus**: https://github.com/cmin764/morseus (aplicația propriu-zisă, interfață)

libmorse

Aceasta este biblioteca principală, ce stă la baza “creierului” aplicației, capabilă să primească din orice mediu, oricum, semnale de forma unor tuple de *(stare, cuantă)* unde **starea** o reprezintă prezența sau absența unui semnal (ex.: True/False), iar **cuanta** se referă la durata de timp a acelei stări în milisecunde (ex.: 300ms, 764ms etc.). Scopul ei este de a primi și de a analiza concomitent liste de astfel de tuple, pentru a le transforma în final în șiruri de caractere ale alfabetului latin, caractere ce reprezintă de fapt textul transmis prin astfel de semnale.

Morseus

Aceasta este aplicația propiu-zisă ce înglobează atât interfața grafică cât și modulul de procesare al imaginilor, cu scopul de a obține tuple de forma *(stare, cuantă)* cât mai bine aproximate, identificând corect sursa de lumină principală, dacă există o astfel de sursă, mai multe sau chiar niciuna. În cele din urmă, aceste tuple sunt redirecționate mai departe către bibliotecă pentru analiză, clasificare și convertire în textul aferent ce este introdus automat în caseta text de ieșire a interfeței pe măsură ce sunt decodificate noi litere, odată cu primirea semnalelor.

Ambele proiecte sunt scrise în Python 2 cu ajutorul bibliotecilor explicitate mai jos și se face controlul versiunii cu **Git** urmând îndeaproape standardul *gitflow workflow*. Dezvoltarea în sine s-a făcut în cadrul IDE-ului PyCharm Community Edition de la JetBrains.

Acestea sunt sub egida licenței MIT, ceea ce conferă cel mai înalt grad de libertate, adoptat de regulă de majoritatea proiectelor cu sursă liberă. Permisivitatea acesteia oferă posibilitatea de a copia conținutul, de-al modifica, prelucra, extinde, împacheta, privatiza și chiar comercializa, fără absolut nicio obligație sau aprobare necesară din partea autorilor sau persoanelor terțe implicate, singurul act etic rămânând acela de recunoaștere a drepturilor de autor prin însușirea contribuțiilor proprii.

Scipy & numpy

Am folosit intens bine cunoscuta suită de *data-science*, ce de regulă cuprinde biblioteci cum ar fi: numpy, scipy, scikit-learn, pandas, matplot, seaborn, tensorflow, keras, horovod, jupyter etc., limitându-mă bineînteles doar la ceea ce am avut nevoie pentru a implementa și testa cu succes un algoritm stabil și eficient de clusterizare partițională (cu învățare lenesă) a acestor semnale, mai exact **k-means**. Scopul folosirii acestuia este de a clasifica semnalele (și absența lor) în funcție de durata acestora, pentru a le eticheta ulterior ca *punct (1x), linie (3x), intra-spațiu (1x - între semnalele aceluiași caracter), spațiu mic (3x - între caractere complete)* și *spațiu mediu (7x - între cuvinte complete).* Numărul din paranteze se referă la factorul aproximativ de multiplicare al unității pentru a obține (sau a eticheta) entitatea respectivă (exemplu: **spațiu mic**: 3x => *3 x unitate*, cca. 900ms (0.9s) ca durată pentru o unitate standard de 300ms).

Structuri de date & algoritmică

Am beneficiat de ajutorul unor structuri de date ca *deque* (coadă din care se adaugă și extrag elemente eficient din ambele capete) pentru a-mi marca o rază de acțiune mobilă ce dă curs unei învățări și clusterizări corecte, prin a ne uita la o porțiune ideală ca lungime din trecut și a avansa cu ea în prezent într-o manieră continuă.

Au mai fost utilizați algoritmi de bază *greedy*, *whitening,* de sortare, *hashing*, asociere și parcurgere pentru a valida clasificările astfel obținute.

Pillow/PIL

Fără **P**ython **I**mage **L**ibrary nu aș fi reușit într-un timp atât de scurt și într-o manieră atât de eficientă să fac importul și transformările de imagine necesare direct din cameră, în cadre alb-negru cu *blur*, ce sunt normalizate în final la stadiul monocromatic în funcție de un prag adaptabil. Tot acest proces pentru a identifica și izola eficient sursa principală de lumină și a înlătura zgomotul aferent, în timp ce nici detecția corectă a absenței luminii nu a fost neglijată.

Kivy

Framework-ul **Kivy** constituie biblioteca principală a aplicației în sine, Morseus, cu care a fost creată și animată întreaga interfață a acesteia, odată cu posibilitatea expunerii și afișării unei camere capabile să livreze cadre într-un format universal, ușor de procesat ulterior. Kivy nu este doar o biblioteca grafică, este un **N**atural **U**ser **I**nterface capabil să suporte orice platformă multi-touch pe varii arhitecturi și sisteme de operare ca: Windows, Linux, MacOS/OSx, iOS și Android, chiar și sisteme disponibile pe alte dispozitive ARM precum Raspberry Pi. Ea este accelerată 3D cu OpenGL ES 2.0, iar secțiunile critice computaționale sunt delegate cu Cython pentru a aduce un impact pozitiv performanței.

În altă ordine de idei, cu această bibliotecă am creat întreaga interfață a aplicației, dezvoltată pe Linux și portabilă pe Android, dar și primit acces ușor în misiunea de a manipula camera eficient pentru a obține în timp real, cu întârzieri insesizabile, cadrele esențiale prelucrării intrării video.

Python

Python este un limbaj de nivel înalt, orientat pe obiecte în totalitate, ce oferă o gramatică excelentă pentru a proiecta rapid și lizibil aplicații scalabile, sustenabile și chiar rapide. De regulă are la bază interpretorul CPython, ce analizează scripturi *.py*, transformate în propriul bytecode și apoi rulate de proces, fără a ține cont prea mult de arhitectura țintită în momentul dezvoltării codului. Marea comunitate din spate nu m-a dezamăgit niciodată prin oferta bogată de biblioteci și module ajutătoare tangente cu orice domeniu am abordat până acum. Posibilitatea rulării acestuia și în aplicații pentru mobile mi-a hrănit apetitul de a-l alege în dezvoltarea proiectului propus, dar și experiența de peste 10 ani ce mi l-a menținut pe locul întâi în topul uneltelor/limbajelor de programare de-a lungul întregii cariere.

Contribuții

În mod intuitiv, structura lucrării se împarte în două mari capitole, ce tratează separat implementarea și funcționalitățile atât ale bibliotecii **libmorse** cât și ale aplicației **Morseus**. Din motive de simplitate și coerență, am ales să împart fiecare secțiune a lucrării în două jumătăți similare ca structură ce tratează fiecare proiect în parte.

libmorse

După luni bune de cercetare, întreaga bibliotecă a fost dezvoltată, testată automat și documentată de către mine, loc unde am adus următoarele funcționalități și principii:

- O structură a proiectului și o ierarhie a modulelor și resurselor conformă cu metodologia standard de organizare a proiectelor și pachetelor Python compatibile cu *pip* (manager de pachete implicit).
- Fișiere standard specifice instalării pachetului (setup.*), listării dependențelor (requirements.txt), incapsulării licenței (LICENSE), lucrurilor de făcut (TODO) și a preambulului documentației (README).
- Existența dosarelor clasice cu privire la pachetul în sine, teste, documentație, executabile, configurare și resurse.
- Un design arhitectural extensibil ce respectă principiile DRY, KISS și YAGNI prin adoptarea unui sistem de clase abstracte cu metode virtuale pure, moștenite în cele din urmă de clasele finale ce reutilizează cod comun.
- Augmentarea algoritmului clasic **k-means** pentru stabilizarea acestuia cu obiectivul de a clasifica complet și corect semnalele în cauză.
- Cozi de procesare și secțiuni critice rezistente la concurența firelor de execuție cu menirea de a paraleliza procesările independente.
- Ansamblu dinamic de unități (durată de timp indivizibilă) și rații ale acestora.
- Traducerea efectivă a tuplelor în simboluri Morse și în final în caractere text, dar și invers.
- Concatenarea fracțiunilor de semnale cu stări identice și consecutive.
- Înlăturarea zgomotului și ajustarea perioadelor de “liniște” îndelungate.
- Corutine sub formă de generatoare bidirecționale capabile de procesare sistematic asincronă.
- Un arbore lexicografic ce oferă eficient prin parcurgerea lui, dată de simboluri Morse (linie/punct), caractere text ale alfabetului latin, inclusiv cifre și chiar semne de punctuație.
- Metode robuste de prelucrare a textului și simbolurilor Morse întâmpinate în zone tampon determinate din analiză sau oferite ca intrare.
- Resursă principală ce conține un dicționar care asociază fiecărui caracter tratat, un șir de simboluri Morse (linie/punct).
- Script executabil capabil să ofere o interfață la linia de comandă prin care se pot executa diverse rutine de traducere a unor semnale Morse sub forma intrării stabilite (tuple de *(stare, cuantă)*) și respectiv de codificare a textului latin în astfel de semnale.
- Schelet pentru documentație tip *Read the Docs* cu Sphinx și reStructuredText.
- Testare automată atomică de tip *unittest*.
- Sistem versatil de excepții ierarhice adnotate de coduri de eroare ce sunt aruncate în funcție de tipul și locul erorii produse.
- Logică specifică pentru setări și utilitare.
- Salvarea pe disc a mesajelor de jurnal ce oferă informații detaliate cu privire la execuția logicii.

Morseus

Interfața aplicației finale a fost proiectată de la zero prin intermediul limbajului Kivy (.kv) și funcționalitatea ei a fost redată de extensii ce s-au ocupat de captarea cadrelor prin intermediul camerei, post-procesarea și normalizarea acestora în tuple de semnale și actualizarea câmpurilor de intrare/ieșire în funcție de evenimentele produse de-a lungul rulării.

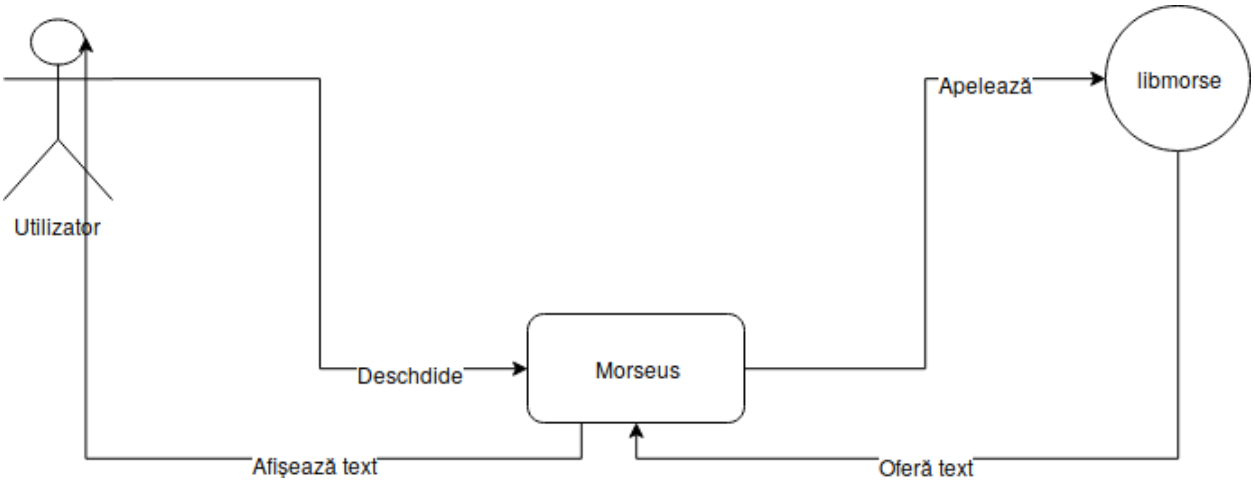
Mai jos avem și o listă de contribuții proprii:

- Același standard de împachetare ca și la biblioteca de mai sus (libmorse).
- Arhitectură MVC unde *controller-ul* este framework-ul în sine.
- Fișierul *morseus.kv* ce descrie schema grafică a întregii interfețe dar și legăturile active dintre câmpuri de date și varii evenimente.
- Pachetul principal ce reprezintă aplicația în sine, unde acesta separă și decuplează codul aferent interfeței grafice de cel al procesării cadrelor.
- Meniu cu panouri separate pentru trimitere, primire, cât și setări.
- Funcție de primire a semnalelor cu rolul de a salva cadre esențiale din filmul capturat.
- Rutine de pre-procesare și abstractizare a capturilor de interes în pete de lumină (imagine -> decupare -> alb și negru -> estompare -> transformare monocroma cu prag dinamic).
- Încadrarea petelor de lumină în perimetre minime cu eliminarea zgomotului din împrejurimi, operație prin care se determină dacă captura finală relevă sau nu prezența unui semnal.
- Chenar ajutător pentru captarea semnalelor de interes, afișat peste proiecția conținutului camerei.
- Funcție de resetare a contextului actual: cameră, casetă ieșire, instanță de traducere.
- Funcție de trimitere a semnalelor în timpi influențați de o recepționare anterioară, cu ajutorul colorării unui dreptunghi sau utilizarea blițului în cazul aplicației de pe mobil.
- Fire de execuție concurente și apeluri asincrone pentru interacțiuni line și continue.

I. Proiectare

În momentul în care vrem să analizăm astfel de semnale și să obținem textul reprezentat de acestea, putem să ne imaginăm foarte ușor, încă din fazele incipiente ale arhitecturii aplicației, faptul că avem nevoie de capabilitatea de a capta și izola cu exactitate acele momente succesive de timpi cu alternări de prezență-absență de semnal (sursă de lumină). Apoi, această caracteristică ar fi nefolositoare fără abilitatea de a transforma aceste alternări în funcție de timp în niște simboluri cunoscute (linie/punct), mai exact digitalizarea unui semnal analog și totodată, conversia simbolurilor digitale în caractere text majuscule ale alfabetului latin ce reprezintă efectiv textul final dorit de către utilizator.

Arhitectura soluției

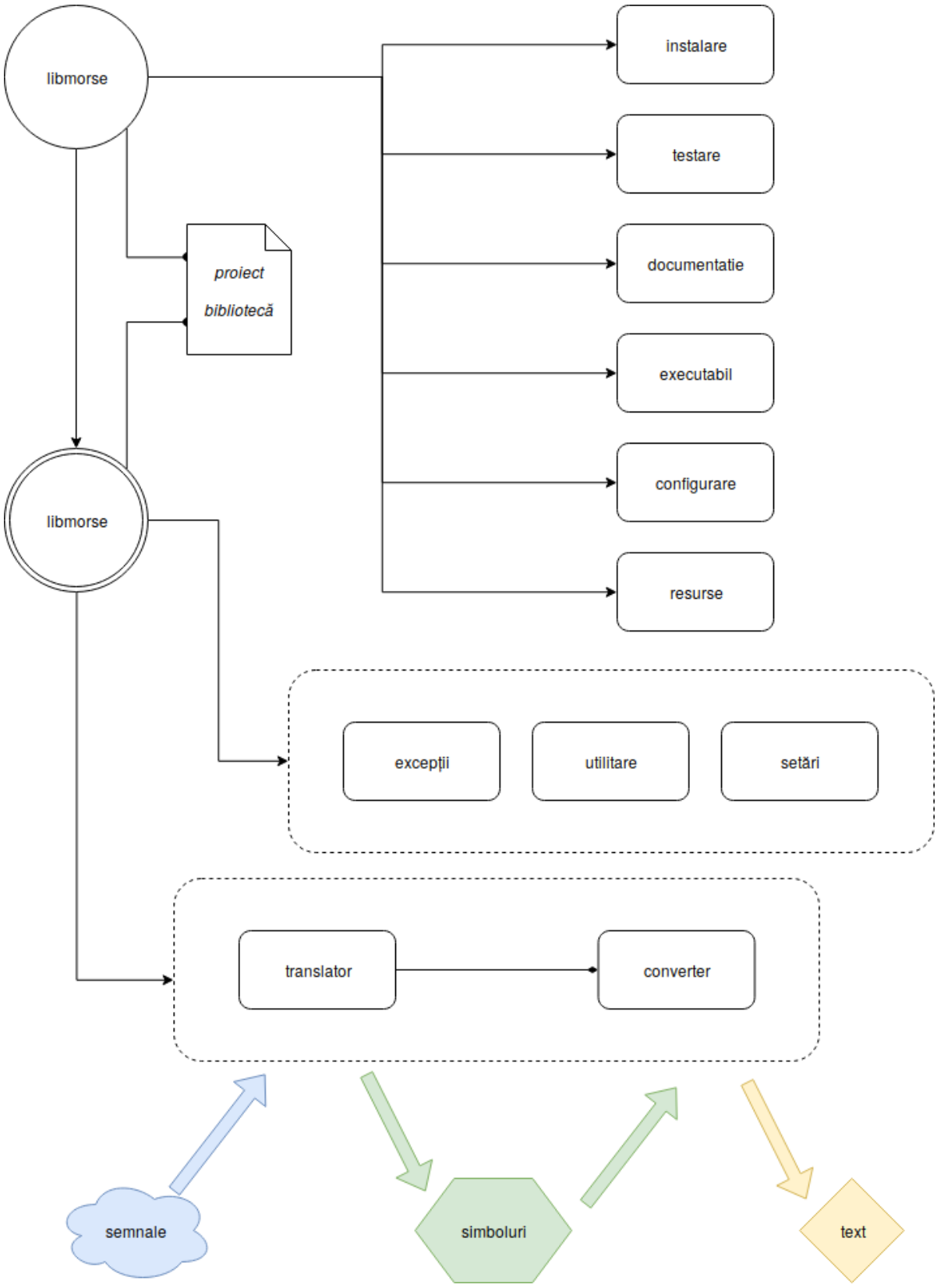


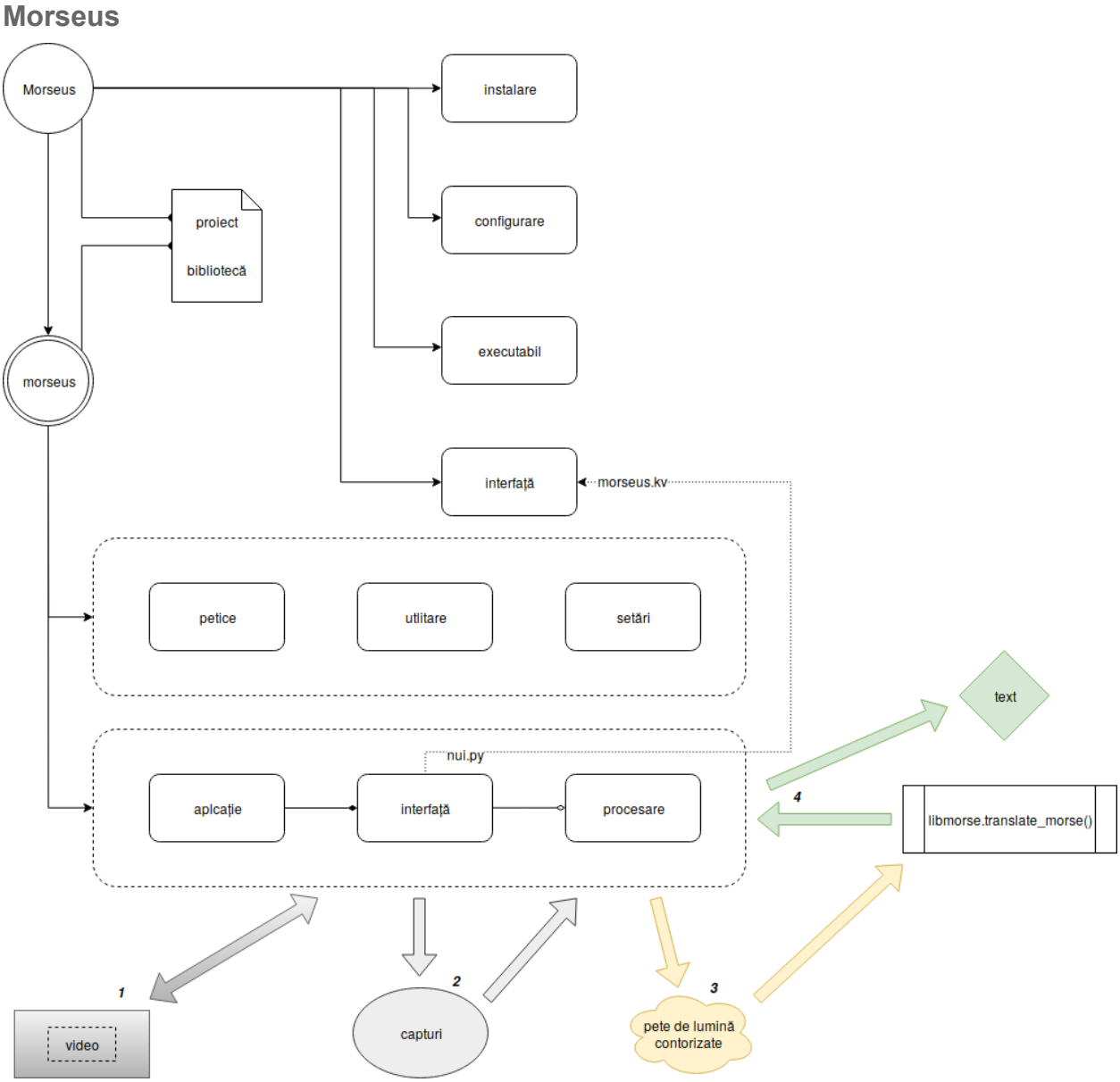
Componente

Din acest exercițiu simplu al gândirii, se pot contura clar două mari componente, decuplate una de alta și de o coeziune maximă fiecare, astfel încât vom avea proiectul **Morseus**, ce ne dezvăluie o interfață simplă și minimalistă, afișându-i utilizatorului camera și caseta în care va fi pus textul decodificat și proiectul **libmorse**, ce va prelucra semnalele normalizate captate de cameră în textul afișat în final în acea casetă de ieșire din interfață.

Ambele proiecte sunt pur independente și de sine stătătoare, în ideea că biblioteca **libmorse** poate funcționa fără o interfață atașată, iar aplicația principală **Morseus** poate folosi un cu totul alt mecanism de digitalizare și traducere a semnalelor, întrucât aplicația nu face decât să ofere o interfață, să trimită semnale corect interceptate de cameră la bibliotecă, apoi să preia ieșirea acesteia și s-o afișeze înapoi în interfață.

libmorse


libmorse
instalare
testare
documentatie
executabil
configurare
resurse
proiect
bibliotecă
libmorse
excepții
utilitare
setări
translator
converter
semnale
simboluri
text



Structura proiectelor

Ambele proiecte urmează o structură similară, respectând standardele de împachetare din Python și adoptând modelul *bibliotecii*, loc unde fiecare proiect are un pachet de bază alături de celelalte fișiere ce asigură instalarea, documentarea, testarea și managementul corect al acestuia. Acest pachet este de fapt o ierarhie de module și sub-pachete ce pot fi importate și folosite în orice script/program Python ce are acces la ele și de regulă, proiectul aferent lor vine însoțit de un script executabil (sursă sau binar) Python care exploatează funcționalitățile bibliotecii (pachetului) prin diverse comenzi materializate în apeluri ale acesteia. Astfel, avem o interfață la linia de comandă ce ne asigură *view*-ul către capabilitățile bibliotecii și prin comenzile expuse, putem efectiv să folosim biblioteca (executând comenzi în mod direct) fără necesitatea de a crea script-uri de test sau proiecte propriu-zise ce ar face apeluri către aceasta.

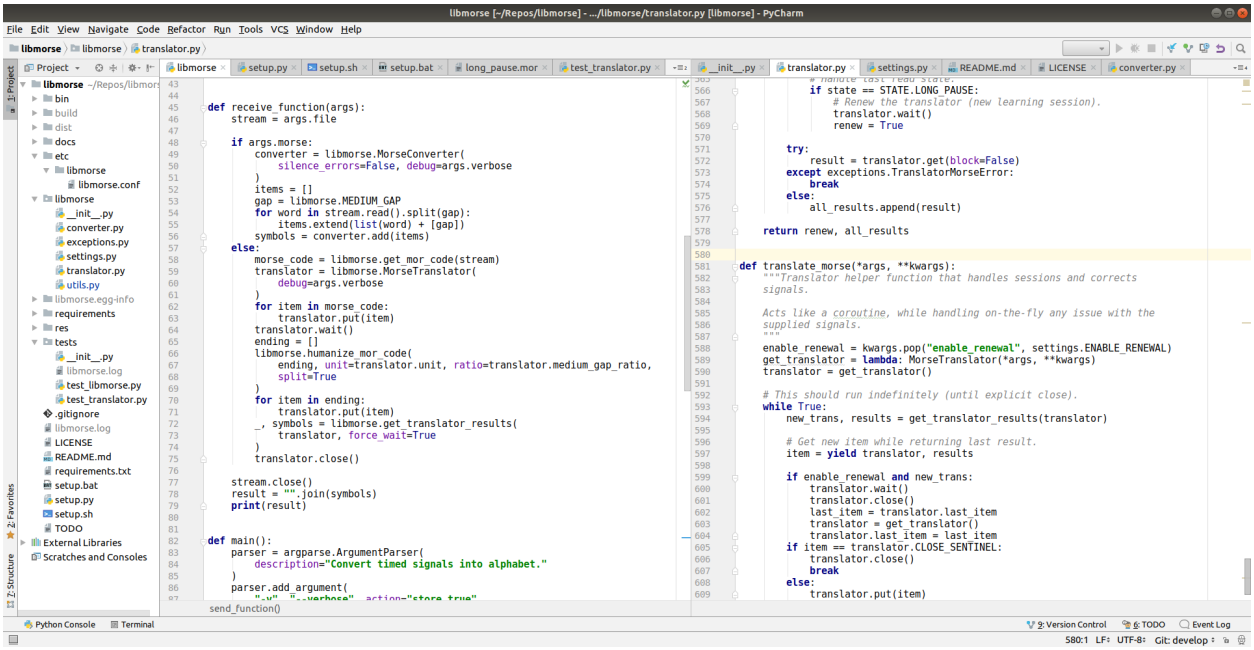
Structură comună

În rădăcina fiecărui proiect vom regăsi dosare și fișiere ca:

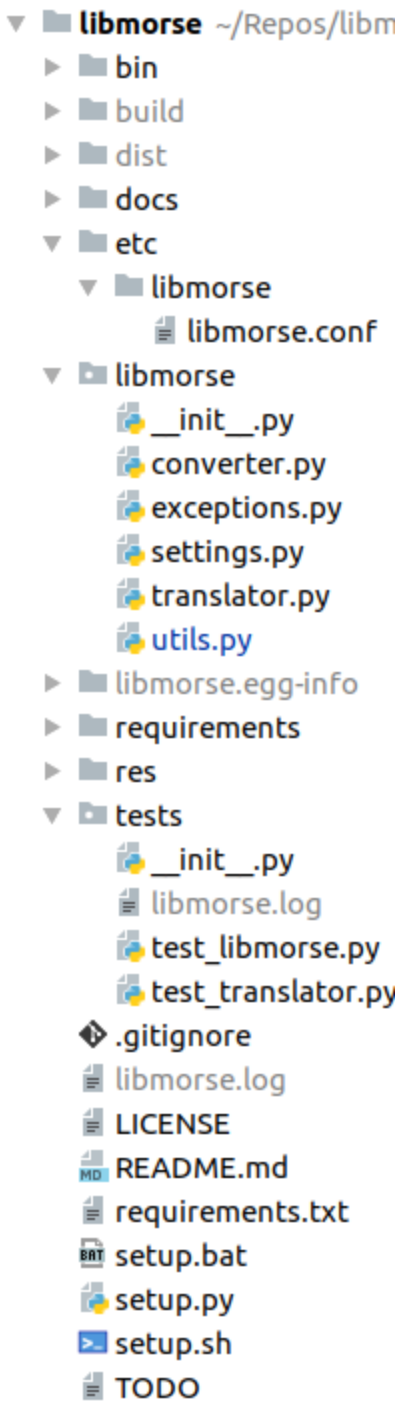
- *bin/* - Loc unde se află scriptul executabil principal al bibliotecii.
- *docs/* - Documentație creată cu Sphinx și fișiere **.rst*.
- *etc/* - Fișiere de configurare.
- *requirements/* - Cerințele pachetului sub formă de liste de pachete adiționale dependente.
- *res/* - Resurse extra utilizate în funcționarea logicii și testare.
- *tests/* - Testele pachetului aflate în exteriorul acestuia.
- **LICENSE** - Licența sub care se află proiectul.
- **README.md** - Prefața documentației și introducerea sumară a întregului pachet.
- **setup.*** - Fișiere de instalare și dezvoltare ale pachetului.
- **TODO** - Sarcini rămase de rezolvat/implementat.

libmorse

Mai jos se poate observa în mare structura pe disc a proiectului, dar și câteva bucăți de cod relevante din fișierul executabil (panou editare stânga) și din modulul de traducere (panou editare dreapta).



libmorse - privire de ansamblu



***libmorse** - structură fișiere disc*

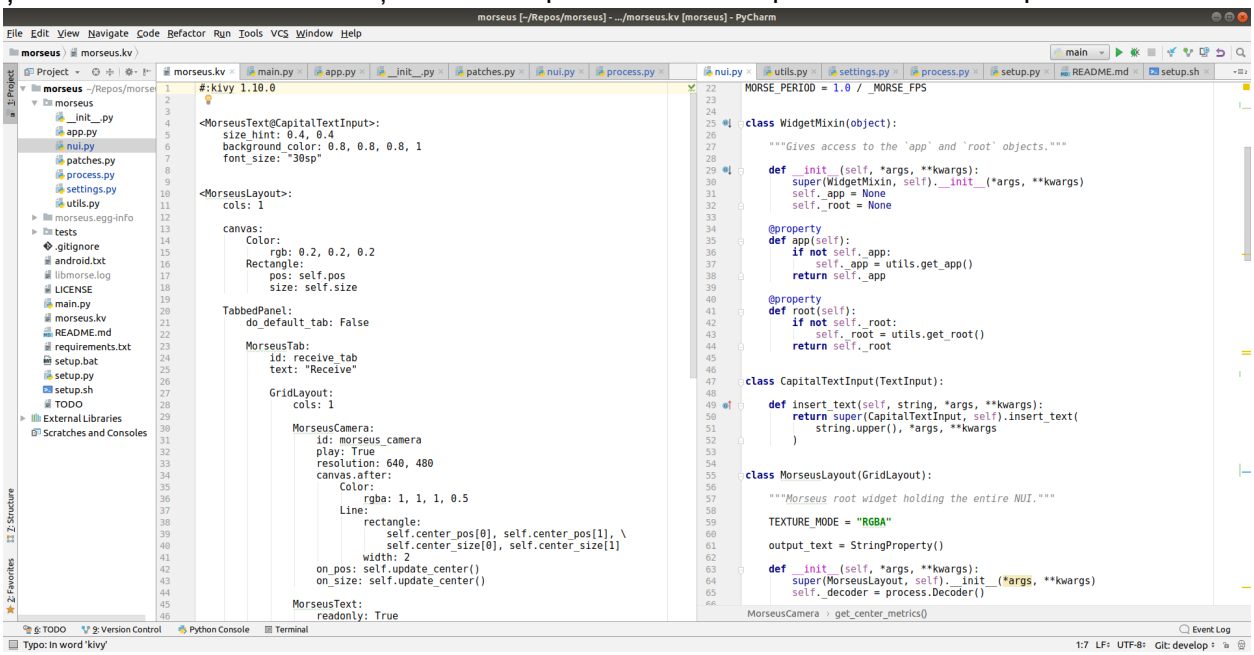
Putem identifica pachetul sub denumirea **libmorse** (exactă cu a proiectului) ce conține mai multe module de interes:

- converter: Acesta cuprinde clasa de bază și clasele finale prin intermediul cărora se face conversia din cod Morse în alfabet și invers (pentru situația în care trimitem semnale).
- exceptions: Clase de eroare aruncate în funcție de locul și contextul producerii acestora.

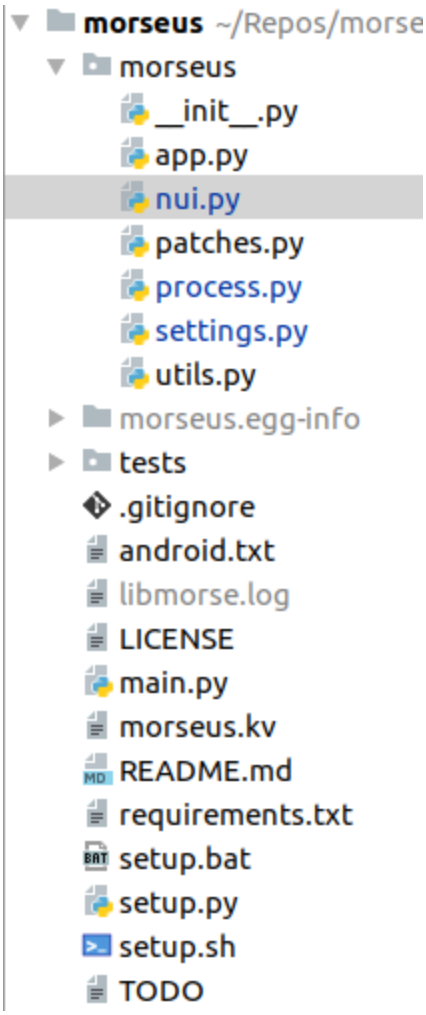
- settings: Aici putem configura diverse setări și parametri pentru a nu *hardcoda* literali direct în cod și pentru a strânge la un loc variabile sensibile modificărilor ocazionale, ce particularizează modul de funcționare al bibliotecii.
- translator: Modul ce se ocupă de digitalizarea perechilor *(stare, cuantă)* în simboluri Morse (linie/punct) și care mai apoi folosește converter-ul pentru a obține textul final.
- utils: Diverse utilitare și funcții ajutătoare cu uz general asupra întregului proiect și nu numai.

Morseus

De data aceasta, structura pe disc a proiectului este ceva mai simplă, însă urmează același standard de împachetare și cuprinde aplicația dezvoltată cu *Kivy* (și kivy language), dar și animarea efectivă a interfeței alături de procesarea în timp real a cadrelor captate.



***Morseus** - privire de ansamblu*



***Morseus** - structură fișiere disc*

Aici avem proiectul **Morseus,** conținând pachetul Python cu același nume, *morseus*, în interiorul căruia se găsesc mai multe fișiere ce separă preocupările diverse ale aplicației în module aparte:

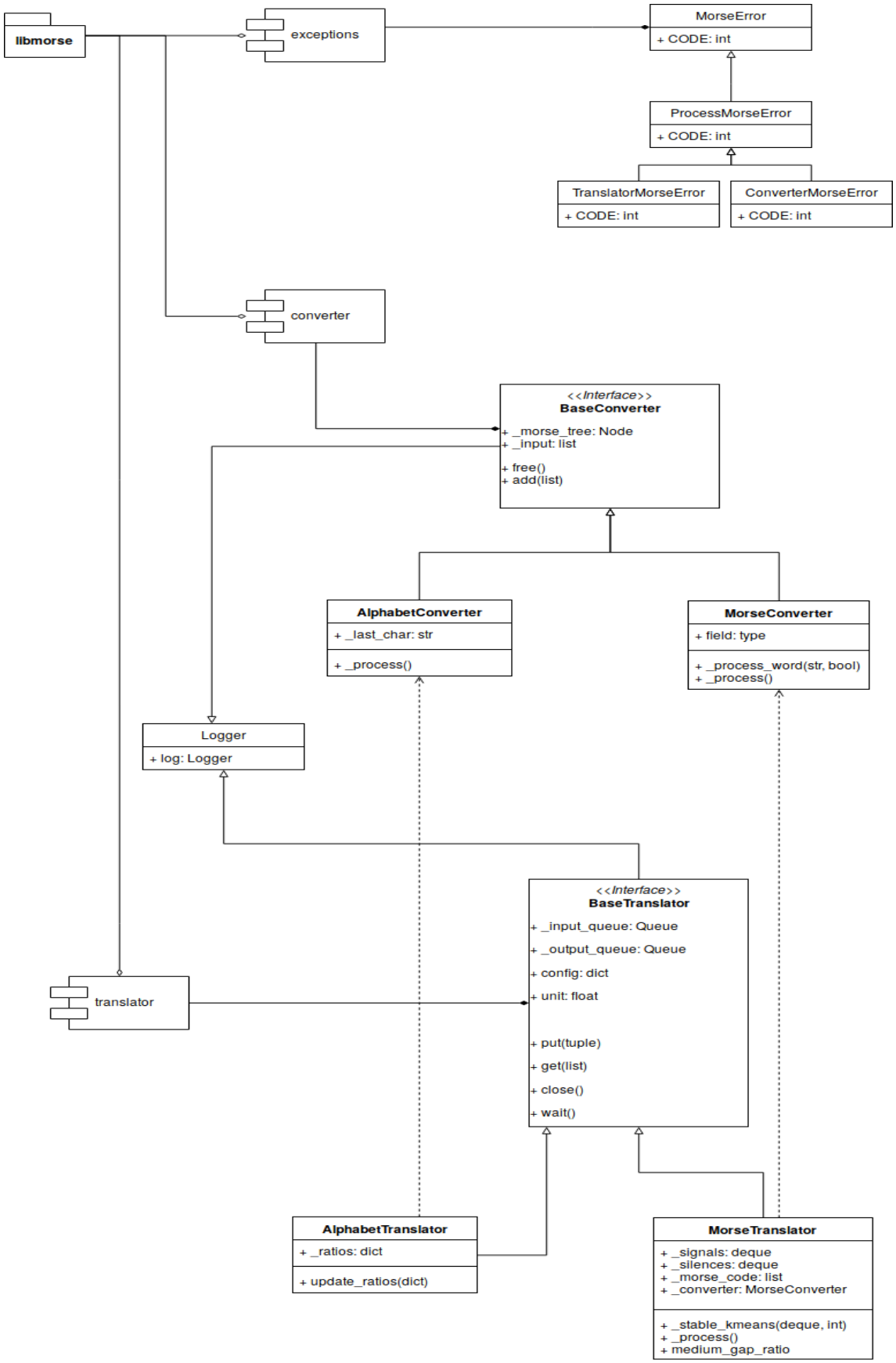
- app: Aplcația în sine (*controller*-ul Kivy) răspunzătoare de instanțierea și servirea interfeței.
- nui: Aici sunt augmentate și configurate toate *widget*-urile (componentele grafice) definite în limbajul Kivy (morseus.kv) cu ajutorul căruia a fost descrisă structura ierarhică a interfeței și proprietățile fiecărei componente în parte.
- patches: Din cauza unor probleme de sincronizare la nivelul camerei, am fost nevoit să aplic ideea de *monkeypatching* în momentul rulării, pentru a cârpi secțiunile problematice din *framework*-ul Kivy cu privire la captarea fluxului video.
- process: Aici se află “creierul” traducerii semnalelor, loc unde fiecare cadru captat (și adnotat de o durată de timp) este normalizat la monocrom (doar alb și negru) și supus unui proces de eliminare a zgomotului, determinând în final dacă pata de lumină identificată reprezintă cu adevărat prezența unui semnal sau din contra, absența ei.
- settings: Aici putem configura diverse setări și parametri pentru a nu *hardcoda* literali direct în cod și pentru a strânge la un loc variabile sensibile modificărilor ocazionale, ce

particularizează modul de funcționare al aplicației. Mai exact, de aici se face reglarea fină și constantă a numărului de cadre pe secundă, raporturi dintre chenarul de încadrare și spațiul întreg al capturilor și alte varii setări cu privire la recunoașterea petelor de lumină principale.
- utils: Diverse utilitare și funcții ajutătoare cu uz general asupra întregului proiect și nu numai. Tot aici avem o funcție importantă care face conversia metricilor din rezoluția de textură în cea de interfață grafică.

Diagrama de clase și obiecte

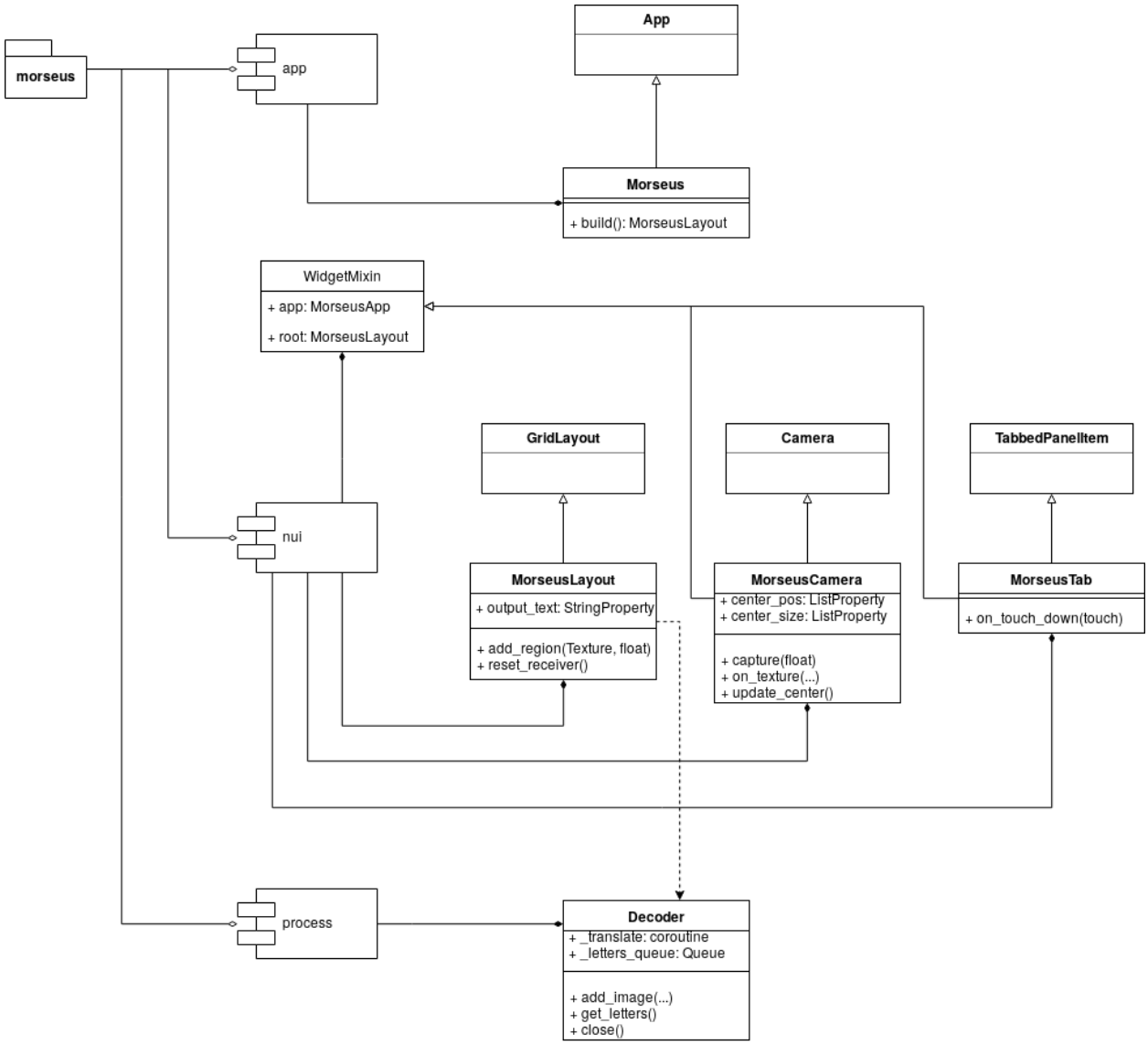
libmorse

Biblioteca urmează modelul claselor abstracte și al moștenirii, dar și conținerii de instanțe ale acestora pentru a facilita procesarea semnalelor fără a scrie cod repetitiv și redundant. Pe lângă aceste derivări, mai avem și câteva funcții ajutătoare ce instanțiază și folosesc aceste clase, respectiv obiecte pentru a procesa mai ușor și interactiv informația, adoptând modelul corutinelor.


libmorse
exceptions
MorseError
+ CODE: int
ProcessMorseError
+ CODE: int
TranslatorMorseError
+ CODE: int
ConverterMorseError
+ CODE: int
converter
<<Interface>>
BaseConverter
+ _morse_tree: Node
+ _input: list
+ free()
+ add(list)
AlphabetConverter
+ _last_char: str
+ _process()
MorseConverter
+ field: type
+ _process_word(str, bool)
+ _process()
Logger
+ log: Logger
translator
<<Interface>>
BaseTranslator
+ _input_queue: Queue
+ _output_queue: Queue
+ config: dict
+ unit: float
+ put(tuple)
+ get(list)
+ close()
+ wait()
AlphabetTranslator
+ _ratios: dict
+ update_ratios(dict)
MorseTranslator
+ _signals: deque
+ _silences: deque
+ _morse_code: list
+ _converter: MorseConverter
+ _stable_kmeans(deque, int)
+ _process()
+ medium_gap_ratio

Morseus

Arhitectura produsului **Morseus** este mult mai simplă, ea modelându-se în jurul *framework*-ului Kivy și plecând de la punctul de intrare *main.py* care instanțiază și pornește efectiv aplicația și bucla de evenimente. Rolul pachetului *morseus* este de a agrega toate derivările și augmentările claselor standard din *framework* ce reprezintă varii componente de interfață și așezare în ecran, dar și de a crea o colaborare eficientă între logica ce afișează și animează interfața **și** secțiunea de cod ce face transformarea cadrelor în pete de lumină contorizate și în final în text redat în interfață (cu **libmorse**).



Modelarea datelor

În cazul ambelor proiecte nu avem nevoie de bază de date, modele sau vreun fel de metodă de stocare persistentă pe disc, deoarece avem de a face doar cu liste de tuple *(stare, cuantă)* și cozi de simboluri, respectiv caractere, iar observațiile și configurațiile învățate pe parcurs trebuie să rămână efemere și totodată ocupă o cantitate neglijabilă de memorie. Adăugarea sprijinului unei forme de persistență a datelor s-a observat a fi complet inutilă, iar despre un produs se spune că este “perfect” nu atunci când nu mai ai ce adăuga, ci atunci când nu poți renunța la nimic.

Acestea fiind spuse, vor fi descrie succint structurile de date de bază folosite activ în cadrul procesării entităților cu care avem de-a face.

libmorse

- Liste, dicționare, tuple și alte tipuri primitive.
- *collections.deque*: cozi cu două capete.
- *Queue.Queue*: cozi sigure la concurență (fire de execuție).
- *anytree.Node*: nod versatil de la care începe orice sub-arbore.

Sistemul de cozi a fost folosit pentru transferul entităților și rezultatelor din exterior către interior și invers, iar prin structurile arborescente s-a putut alcătui un arbore de prefixe *(trie)* cu ajutorul căruia se poate determina eficient un caracter asociat unui anume șir de simboluri Morse.

Morseus

- Liste, dicționare, tuple și alte tipuri primitive.
- *collections.deque*: cozi cu două capete.
- *Queue.Queue*: cozi sigure la concurență (fire de execuție).
- *numpy.ndarray*: matrice de pixeli.
- *kivy.graphics.texture.TextureRegion*: captură din camera video.

În mod similar cu *libmorse*, aceste cozi s-au folosit pentru a colecta semnalele trimise bibliotecii și literele produse de aceasta pentru a fi afișate ulterior în caseta text a interfeței, dar și pentru a memora temporar pixelii din imagini ce marchează suprafețe conexe de lumină, descoperite astfel în cadrele captate.

Cozi de procesare și comunicare

În acest capitol vom detalia fluxul de date și rutinele de procesare ale acestora, inclusiv pașii intermediari, pornind de la intrări de bază până la rezultatul final, cu pre și post procesări de-a lungul traseului.

libmorse

Totul pleacă din modulul **translator** care oferă corutina *translate_morse* ce primește elemente de forma *(stare, cuantă)* și pe parcurs ce trimitem astfel de elemente, începem să primim caractere finale text ale alfabetului latin ce alcătuiesc textul exprimat de simbolurile trimise. Fluxul de intrare/ieșire se realizează prin interacțiunea cu generatorul rezultat în urma apelului funcției de mai sus, capabil nu doar să ofere informație, ci și să primească. La fiecare astfel de schimb de date, se redă controlul apelantului, pentru a nu sta blocant în operații și pentru a granula timpul de interacțiune utilizator-generator.

După cum am observat, tuple de semnale intră în corutină și în final ies caractere text. Asta înseamnă că în interior se petrec niște procesări ale acestor intrări, mai exact niște adăugări în coada de procesare internă instanței clasei *MorseTranslator*. Pe măsură ce sunt adăugate semnale, în paralel și concurent există un fir de execuție secundar ce consumă elemente noi adăugate prin a le procesa cu metoda specifică oricărei derivări din *BaseTranslator*, numită *process*. Valorile returnate de metoda *process* sunt considerate a fi caracterele text finale, retrase din coada de ieșire mai târziu de utilizator prin firul principal de execuție.

Cum se face procesarea?

Algoritmul din spate se bazează pe ideea unei raze de semnale active și relevante, pornind de la ultimul semnal adăugat în rază (coadă) până la **N** semnale în urmă (trecut). Această rază este de obicei fixă ca lungime și călătorește odată cu ultimele **N** semnale, iar lungimea ei reflectă cantitatea de informație contiguă, relevantă învățării și clusterizării semnalelor primite. La început, este nevoie de un număr minim de astfel de semnale, apoi, după ce limita inferioară este depășită, cu fiecare semnal nou adăugat se rulează algoritmul de clusterizare (un *k-means* augmentat, detaliat în capitolul “Implementare”) al cărui scop este să grupeze și în final să clasifice aceste semnale în funcție de durată. În acest fel grupăm semnalele în două sau trei clustere (pentru prezența lor, respectiv lipsa lor), deoarece atunci când avem lumină prezentă, putem avea de regulă linii (mai lungi ca durată) sau puncte (mai scurte) și în mod analog avem pauze intra-caracter (între simbolurile aceleiași litere), mici (între literele aceluiași cuvânt) și medii (între cuvinte diferite) prin simpla absență a luminii, cronometrată.

k-means

După ce gruparea se realizează complet (ocupând toate clusterele dorite), în funcție de rațiile prestabilite din standardul Morse:

- Punct = o unitate (lumină)
- Linie = trei unități (lumină)

- Intra-spațiu = o unitate (întuneric)
- Spațiu mic = trei unități (întuneric)

- Spațiu mediu = șapte unități (întuneric)

se realizează etichetarea semnalelor cu respectivele simboluri, prin încadrarea duratei fiecărui semnal la una din cele două/trei caracteristici. Astfel, eticheta cu durata cea mai mică devine unitate (căci unitatea e cea mai mică masură indivizibilă), adică punctul în cadrul prezenței luminii și intra-spațiul în cazul absenței ei, iar eticheta următoare ca durată, cea mai apropiată de triplul unității, devine linie, respectiv spațiul mic și în final, doar în cazul absenței luminii, eticheta cea mai mare (lungă) devine spațiul mediu. Această clasificare și etichetare ne digitalizează cumulul de semnale analogice contigue într-o listă discretă de simboluri Morse (linie, punct, spații) pe care o vom transforma în final în caractere text.

Arbore binar lexicografic (simboluri -> caracter)

Acum că avem aceste liste de simboluri, nu ne rămâne decât să delimităm cuvintele între ele prin spațiile medii, apoi literele fiecărui cuvânt prin spațiile mici și rămânem în final cu liste de tuple de linii și puncte, unde fiecare tuplă se referă la câte un anume caracter. Aceste tuple scurte de simboluri sunt transformate într-o parcurgere a arborelui binar lexicografic de simboluri, pornind din rădăcină și explorând unul din cei doi fii pe muchia corespunzătoare simbolului (punct sau linie) până la epuizarea simbolurilor și în momentul consumării și a ultimului simbol, putem spune cu încredere că nodul pe care ne situăm reprezintă de fapt și caracterul căutat.

Morseus

Prin deschiderea aplicației, înțelegem (tehnic) rularea metodei *run* din instanța principală a clasei *Morseus* răspunzătoare construcției lui *MorseusLayout*, ce creează și afișează întreaga interfață cu tot cu animațiile și *callback*-urile de rigoare. Odată cu pornirea aplicației și *feed*-ul camerei video este pornit spre a fi redat în interfață și totodată segmentat în capturi la o anumită perioadă (inversul frecvenței) de cadre pe secundă. Aceste cadre sunt supuse unor serii de procesări în timp real și în urma lor rezultă tipul stării identificate (True - prezență a luminii / False - în caz contrar) împreună cu durata în milisecunde a respectivei stări. Aceste perechi de *(stare, cuantă)* sunt redirecționate mai departe către corutina returnată de *libmorse.translate_morse()* pentru a obține în final textul așteptat spre a fi afișat în interfață.

După cum se observă deja, aplicația în sine nu are prea multe responsabilități, deoarece toate aceste proceduri complexe sunt manipulate de biblioteca *libmorse*. Ea are totuși un punct de “concentrație” maximă, înclinat spre sarcini atașate procesorului (și nu întreruperi I/O), acela unde imaginile suferă o normalizare din color în alb și negru, apoi estompări cu discretizarea pixelilor la monocrom pentru a obține petele de interes separate de mediul înconjurător și zgomot. Identificarea acestor suprafețe se obține printr-un algoritm de umplere prin inundare și prin comparația rațiilor suprafeței maxime în raport cu toate celelalte, considerate zgomot ipotetic. Din fericire, acest set de proceduri aplicate fiecărui cadru capturat ocupă maxim câteva milisecunde în spațiul temporal, lucru ce nu afectează calitatea traducerii semnalelor în timp real din două motive complementare:

1. Însăși durata de procesare este neglijabilă pe lângă cea a unității (masură minimă indivizibilă ce stă la baza oricărui semnal).
2. Se deschide cate un fir de execuție separat pentru fiecare astfel de procesare, fir ce are responsabilitatea de a le aștepta pe toate celelalte deschise anterior în aceeași direcție. Astfel, bucla de evenimente nu va aștepta niciodată pe urma vreunei procesări și fiecare

fir își va urma cursul logic secvențial în mod natural, fără să se ivească cazul blocajului sau așteptării celui anterior, eficiența fiind maximizată.

În paralel, firul de execuție principal (cel atribuit buclei de evenimente) interoghează ocazional coada de ieșire a decodorului (instanță a clasei *Decoder*) pentru a scoate din coadă și afișa respectivele caractere text direct în caseta text de ieșire a interfeței.

Toate aceste concepte le vom înțelege mult mai ușor prin parcurgerea (deja realizată anterior) a diagramelor de clasă și a capitolului ce urmează, cu privire la “Interfața cu utilizatorul*”*.

Interfața cu utilizatorul

Atât biblioteca cât și aplicația dispun de o formă de interfață (CLI, respectiv NUI) ce face posibilă și ușurează interacțiunea dintre utilizator și logică, astfel, în cazul bibliotecii există un script executabil ce expune toate funcționalitățile ei, iar în cadrul aplicației, avem de a face cu un *GUI* conceput pentru orice platformă multi-touch ce oferă posibilitatea utilizatorului de a interacționa în mod natural și nativ, direct cu ajutorul gesturilor efectuate de către acesta pe ecranul tactil.

libmorse

Aceasta este doar o bibliotecă, ce presupune importul ei în alte script-uri și pachete, apoi folosirea unor funcții și/sau clase expuse de ea, dar din motive de testare (de integrare), am conceput și un script executabil cu interfață la linia de comandă ce exploatează următoarele capabilități:

- **send**:
  - *Semnale*: `python bin\libmorse -v send -o morse.mor "morse code"`
    Cu această simplă comandă se transformă textul “morse code” direct în semnale analogice, adică o listă de tuple de forma *(stare, cuantă)* ce semnifică prezența/absența luminii și durata acestei perioade continue.
    Exemplu (linii din fișierul *morse.mor*):
    ```
    0 300.0
    1 300.0
    0 900.0
    1 300.0
    ```
  - *Morse*: `python bin\libmorse -v send -m -o morse.txt "morse code"`
    Aceeași comanda/operație ca mai sus, numai că de data aceasta se oferă traducerea intermediară în simboluri Morse (cu parametrul **-m**), fără a le mai și converti în semnale ca cele de mai sus.
    Exemplu (șir generat și salvat opțional în fișierul *morse.txt*):
    ```
    -- --- .-. ... . / -.-. --- -.. .
    ```
- **receive:**
  - *Semnale*: `python bin\libmorse -v receive morse.mor`

Comanda inversă celei de trimitere, care efectiv transformă acele semnale pe post de tuple înapoi în textul exprimat de secvență.

Exemplu (după rularea în prealabil a comenzii de trimis, corespondentă acesteia):

MORSE CODE

- *Morse*: `python bin\libmorse -v receive -m morse.txt`

  Comandă similară cu cea imediat de mai sus, singura diferență rămânând faptul că intrarea o constituie simboluri Morse și nu acele tuple de stări și durate.

  Exemplu (la fel, după rularea celei de trimis, cea cu parametrul **-m** (adică simboluri Morse, nu semnalele din final)):

  MORSE CODE

Funcții ajutătoare

Pentru a folosi biblioteca în codul programatorului, este de ajuns sa fie importată, apoi folosită funcția *translate_morse*, al cărei apel întoarce o corutină prin intermediul căreia se trimit secvențial tuple de semnale și odată cu trimiterea acestora, se primesc prin iterarea ei (pe post de generator) caractere text ale alfabetului, exprimate întocmai de semnalele trimise inițial.

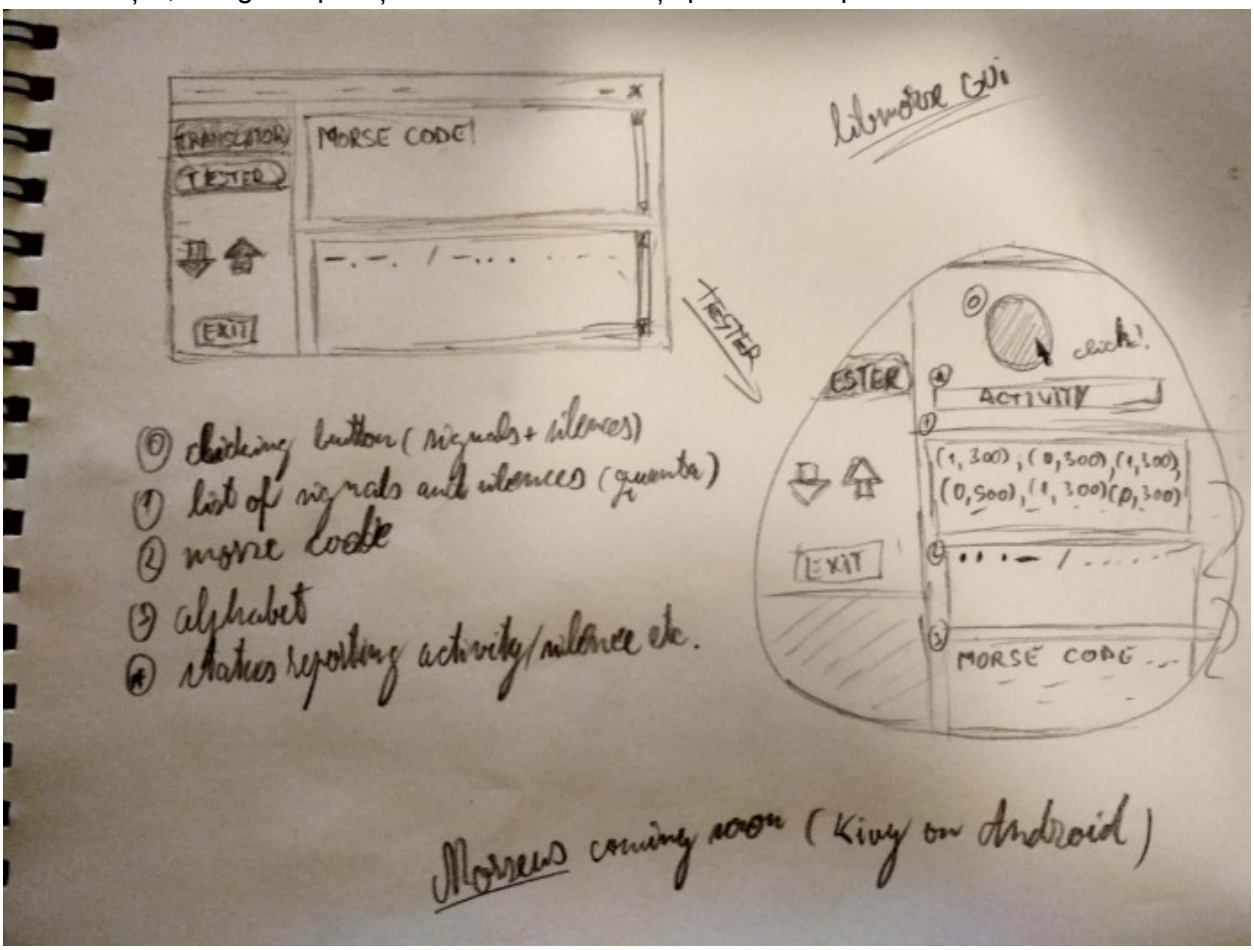
Mai multe detalii despre modul de funcționare al acesteia, aici: https://github.com/cmin764/libmorse#within-module (descrie prin exemple cum funcționează apelul).

Nu există funcții ajutătoare (încă) și pentru cealaltă capabilitate de a converti și trimite astfel de semnale, fiindcă însăși procedura e simplă: se creează o instanță a clasei de trimitere, *AlphabetTranslator*, prin intermediul al cărei obiect putem trimite textul și apoi scoate din coada de ieșire semnalele generate, pe care le vom trimite mai departe sub formă de aprinderi/închideri ale blițului dispozitivului (sau prin alte mijloace).

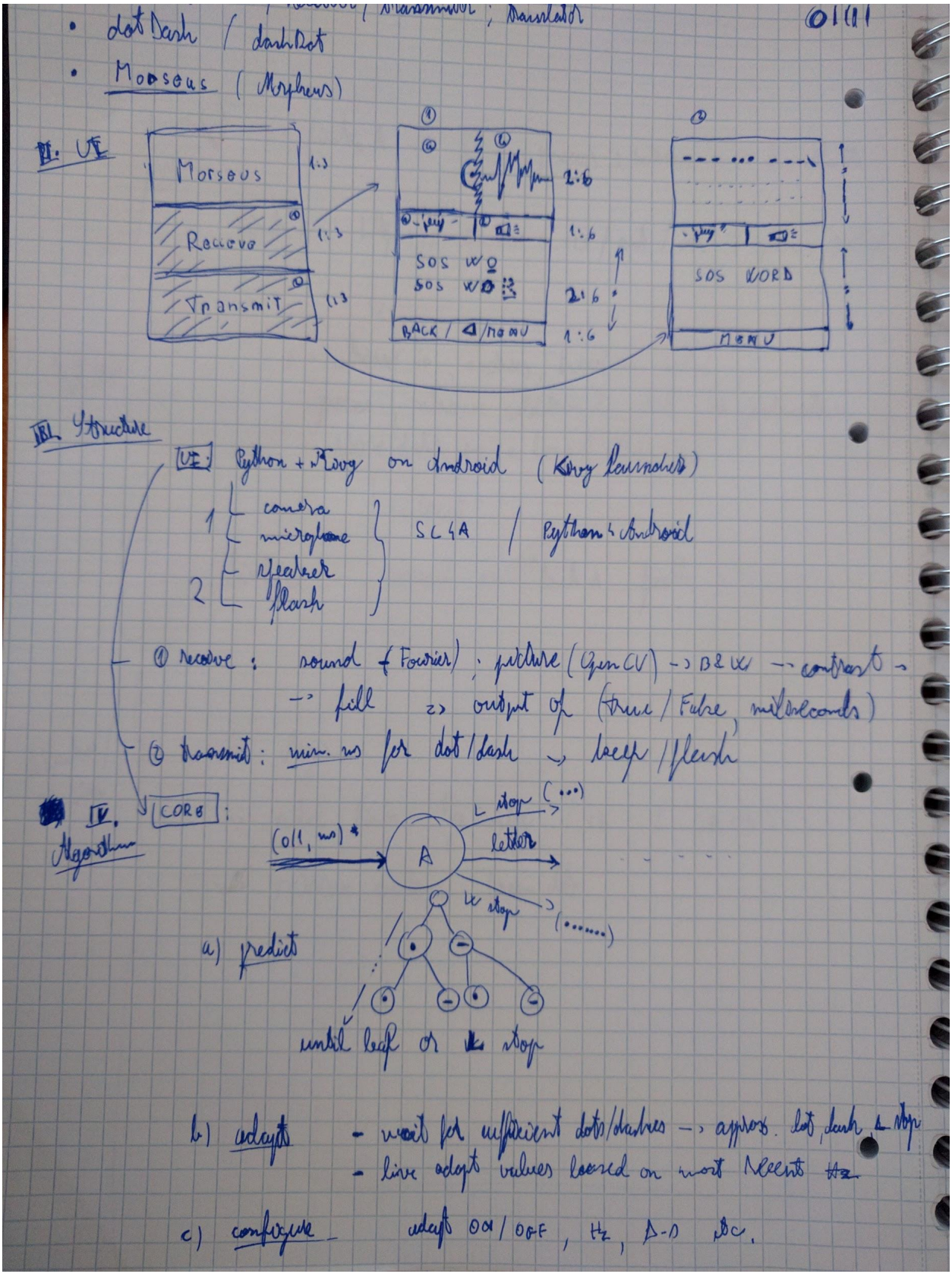
Morseus

Interfața a fost concepută direct în modul nativ al framework-ului Kivy, limbajul kivy, ce ia forma unui fișier *yaml* cu extensia **.kv**, cuprinzând detalierea și configurarea fiecărui widget ce urmează a fi afișat pe ecran. Animarea ei s-a făcut prin supraîncărcarea claselor componente cu adăugări de funcționalități declanșate de varii evenimente induse de ceas, utilizator sau dispozitive de intrare/ieșire.

Inițial, designul aplicației în materie de schițe pe hârtie cuprindea următoarele:



*Interfață de test în **libmorse***

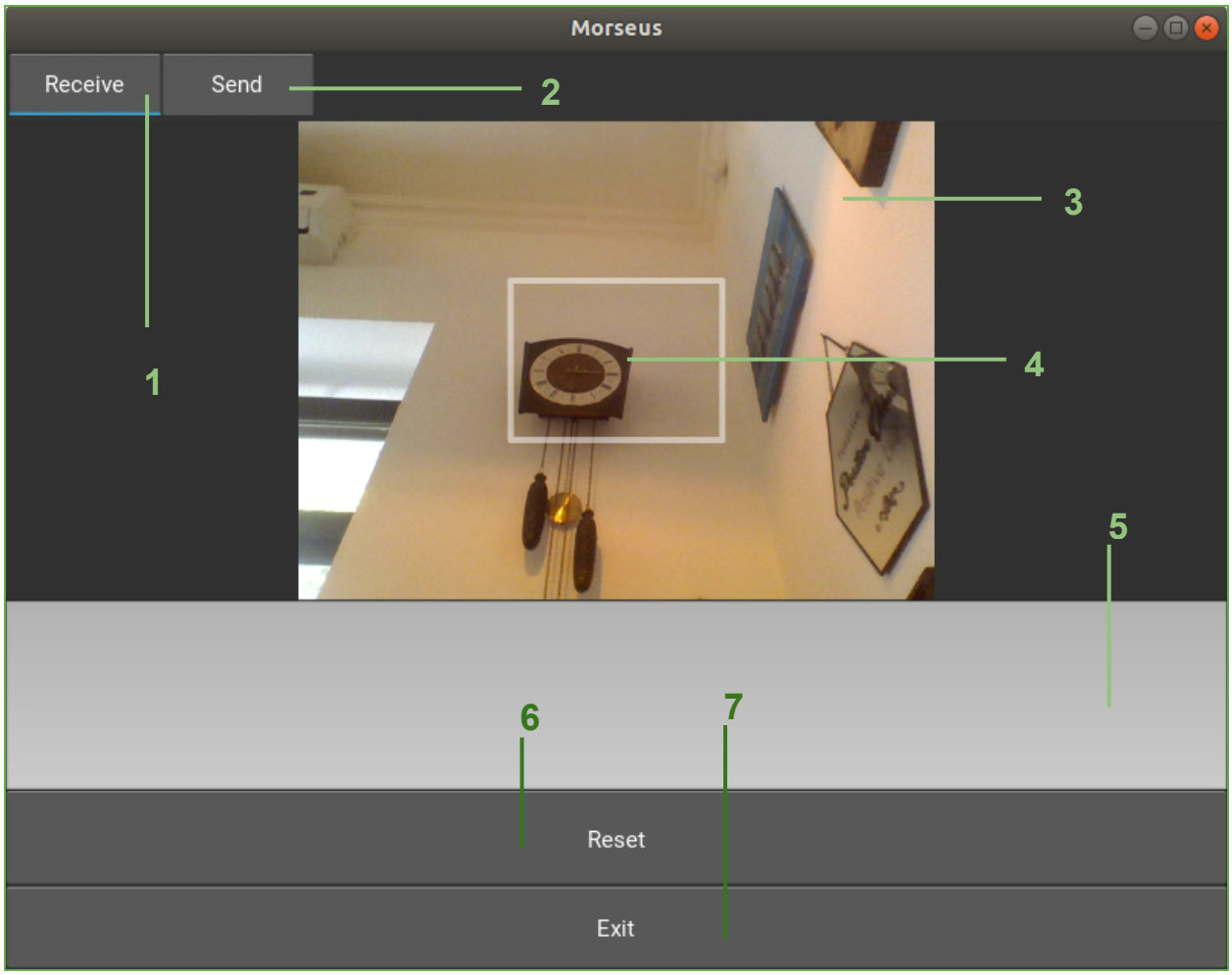


Schiță aplicație finală (**Morseus**)

În momentul de față, aplicația finală arată și funcționează în felul următor:

Funcția de primire și decodificare a semnalelor

Cum vom deschide aplicația, vom observa imaginea de mai jos, aceasta reprezentând meniul implicit al ei.

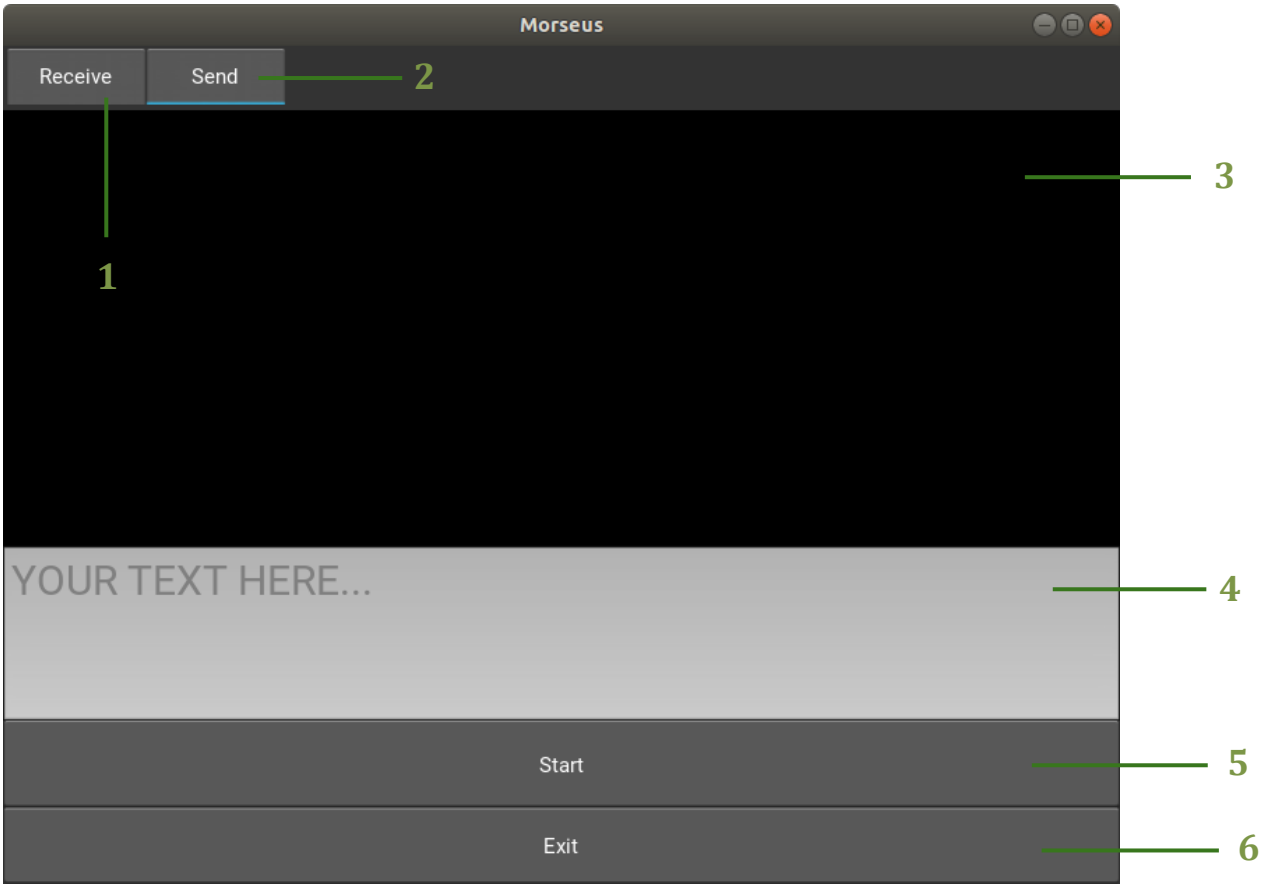


În imaginea de mai sus putem observa mai multe componente de interfață, toate structurate și ordonate în așa fel încât interacțiunea cu utilizatorul să fie cât mai lină și fluentă. Acestea au fost identificate prin următoarele numere:

1. Panoul răspunzător cu funcționalitățile de primire a semnalelor (din meniul ce include panoul de primire, cât și pe cel de trimitere, despre care vom afla mai jos).
2. Panoul ce activează interfața de trimitere de semnale.
3. Textura întreagă capturată de cameră și redată live în interfață.
4. Regiunea de interes a texturii luată în calcul în cazul detecției.
5. Caseta text de ieșire unde se afișează textul final descifrat.
6. Buton de resetare, ce reinițializează camera, decodorul și caseta text.
7. Buton de ieșire, care închide aplicația.

Funcția de codificare și trimitere a semnalelor

Dacă vom da click pe meniul de trimitere “Send” vom observa interfața de mai jos.



Punctele 1, 2 și 6 le cunoaștem deja, noutățiile fiind 3, 4 și 5.

1. Către panoul de primit semnale.
2. Către cel de trimis semnale (panoul curent).
3. Ecran proiector ce-și schimbă intermitent culoarea în alb de fiecare dată când se trimite un semnal (pe aplicația de pe mobil se va aprinde simultan și blițul în mod similar).
4. Caseta text de **intrare**, loc unde utilizatorul scrie textul dorit ce se dorește a fi codificat în semnale Morse.
5. Butonul de start care inițiază procedura de trimitere a semnalelor, buton ce se transformă din “Start” în “Stop” în momentul trimiterii și prin apăsarea din nou a acestuia, trimiterea se va opri (și va putea fi repornită ulterior).
6. Buton de ieșire prin a cărui apăsare, aplicația se va închide.

II. Implementare

În acest capitol se vor discuta punctele forte ale bibliotecii, respectiv ale aplicației și voi detalia câteva din conceptele dezvăluite în sub-capitolul “Cozi de procesare și comunicare” din “Proiectare”. Acestea sunt cele ce dau valoare proiectului și fără de care acest tip de detecție adaptabilă fără calibrare nu ar fi fost posibil.

Capitolul va fi împărțit ca și înainte în două secțiuni ce vor trata și detalia separat conceptele de interes atât pentru biblioteca **libmorse** cât și aplicația **Morseus**.

libmorse

Ordonate în funcție de complexitatea și gradul de dificultate al implementării, au fost listate însăși procesarea semnalelor în simboluri și clasificarea acestora pentru o etichetare cât mai apropiată de realitate.

Procesare

Metoda cu pricina se afla aici: https://github.com/cmin764/libmorse/blob/master/libmorse/translator.py#L472-L574. Ea este responsabilă de întreaga procesare, clasificare și încadrare a semnalelor cu scopul de a le digitaliza. Aceasta acționează în felul următor:

1. Semnalele sunt captate unul câte unul din coada de intrare, apoi distribuite în două cozi specifice prezenței semnalului (*signals*) și absenței acestuia (*silences*).
2. La fiecare nouă adăugare se verifică și se iau decizii în funcție de următoarele situații:
    a. Orice două sau mai multe semnale consecutive de același fel se reduc la unul singur cumulând duratele, astfel rezultă o alternare eficientă de *signal-silence* ce îmbunătățește calitatea procesării.
    b. Dacă se primește un semnal de o durată nesemnificativă, atunci acesta este considerat zgomot (*noise*) și înlăturat. Practica ne-a arătat că nu pot exista în mod natural o înlănțuire de astfel de zgomote suficient de mare și omogenă astfel încât totalul duratelor să invoce un semnal relevant de o durată însumată **ne**neglijabilă de data aceasta.
    c. Orice semnal care depășește rația maximă învățată dinamic (adică orice linie sau spațiu mediu) este restrâns la durata maximă posibilă tipului respectiv (*signal/silence*) multiplicând rația maximă învățată cu unitatea medie, astfel, nu vom avea semnale sau pauze anormal de lungi, care ar fi dat peste cap clusterizarea lor. Din moment ce s-ar petrece acest eveniment, orice alt viitor semnal consecutiv de același tip este ignorat (deoarece Morse presupune alternări normale de prezențe/absențe), iar acest lucru este critic bunei funcționări a algoritmului.
3. După stabilirea clară a felului în care se procedează cu un semnal, acesta ar putea afecta sau nu compoziția cozilor *signals* și *silences*, iar această “afectare” trebuie exploatată în secțiunea de “Clasificare” pentru a obține o etichetă (punct, linie, spațiu etc.) validă.
4. În final, aceste etichete, mai bine spus, simboluri, sunt direcționate mai departe către metoda de conversie de simboluri Morse în caractere text ale alfabetului latin. Aceste

majuscule întoarse sunt adăugate în coada de ieșire pentru a fi consumate mai târziu de apeluri externe și servite ulterior ca ieșire.

Clasificare

Aceasta este o sub-secțiune a procesării, loc unde sunt analizate pe rând ambele cozi atât de prezențe cât și de absențe (pauze) a sursei semnalului. Clasificarea se împarte în două faze, cea de clusterizare și validare cu *k-means* și cea de obținere a claselor de simboluri prin care se realizează etichetarea propriu-zisă.

1. Se execută o formă augmentată a algoritmului *k-means* care face următoarele:
    a. Numerele cu virgulă se trec printr-un procedeu de *whitening* cu memorarea factorului de transformare (pentru a readuce centroizii la valoarea originală la final).
    b. Se rulează algoritmii *kmeans* și *vq* din biblioteca **scipy** până când se obține numărul de centroizi așteptat (de regulă din prima dacă poziția internă de inițializare a lor este convenabilă).
    c. Se întorc mediile reale și în funcție de ele distribuția curentă a etichetelor pe vectorul analizat/clusterizat.
2. Se verifică distanțele dintre clustere și dacă acestea nu sunt suficient de mari sau dacă numărul de semnale analizate nu depășește un prag de relevanță, atunci clasificarea eșuează, lăsând cozile de prezențe/pauze neafectate, pe motivul așteptării mai multor date de antrenament-testare.
3. După orice clusterizare reușită, se actualizează media unităților și a rațiilor și se face etichetarea distribuției discrete de la indexul centroidului de care aparține la simbolul aferent lui. Astfel, dintr-o listă de durate nesupravegheate, obținem o listă de puncte și linii în cazul prezențelor și spații intra, mici și medii în cazul absențelor.

Morseus

Aplicația **Morseus** nu a presupus provocări prea mari, deoarece obiectivul acesteia este unul simplu și clar: acela de a captura și izola puncte luminoase de interes ce servesc ca intrare de semnale Morse, determinând corect când și cât timp o sursă de lumină principală stă aprinsă.

Transformare

Totuși, ar fi interesant de explicitat seriile de transformări prin care trec cadrele video din stadiul de fotograme brute până în perechi de stare și durată, iar codul răspunzător acestora se găsește aici: https://github.com/cmin764/morseus/blob/develop/morseus/process.py#L112-L152.

1. Periodic, la un anumit interval determinat de numărul de cadre pe secundă al camerei, se creează capturi de cameră sub formă de texturi brute.
2. Textura brută oferită de camera video este secționată la dreptunghiul central marcat în interfață și acționează pe post de țintă, doar din interiorul căruia se preiau semnalele relevante.

3. Sub-textura astfel obținută (dreptunghiul central ocupând 1/9 din suprafața totală) este încărcată sub forma unui obiect *Image* din **PIL** (Python Image Library cu *Pillow*) prin care se vor realiza toate procesările ulterioare.
4. Această imagine este redusă prin numărul de canale de la RGB[A] la unul singur (cu normalizările de rigoare în vederea determinării corecte a mediei tonalității fiecărui pixel), astfel obținând o imagine alb-negru.
5. Peste imaginea alb-negru se aplică un filtru de estompare (*blur*) pentru a crea difuziuni între suprafețe (reduce zgomotul și varianța), apoi se reduce imaginea la monocrom (alb-negru discret maximal) prin trecerea fiecărui pixel printr-o procedură de transformare, ce decide în funcție de un prag adaptabil dacă un pixel va fi negru (0) sau alb (1).
6. Se obțin astfel pete de lumină (*splashes*) cu tot cu zgomot (străluciri) apărute în mod natural în jurul sursei principale, iar aceste pete sunt încadrate într-un perimetru minim, eliminând excesul de întuneric din jurul lor (*bounding-box*).
7. Această suprafață care de obicei conține una sau mai multe pete de lumină, este trecută printr-un algoritm de umplere cu inundare (*flood-fill*) ce determină aria fiecărei pete detectate și o euristică simplă și eficientă decide dacă petele observate se pliază pe un model așteptat: acela de a avea o pată principală și restul secundare și neglijabile, adică sursa de lumină alături de zgomot irelevant. Numai în cazul unui astfel de model recunoscut se consideră faptul că fotograma analizată a exprimat prezența luminii și absența ei în caz contrar.

Umplere (flood-fill)

Algoritmul este trivial, însă eficient și important în detecția formelor contigue și neregulate într-o matrice de pixeli. El are la bază o coadă de poziții de vizitat, matrice de vizită și matricea de pixeli în sine ca date de intrare. Pașii logicii acestuia sunt:

1. Se iterează fiecare pixel nevizitat al matricei și la fiecare astfel de pixel care are și culoarea albă (loc unde se află lumină) se execută rutina de umplere descrisă la pasul 2.
2. Pornind de la un pixel nevizitat și alb, se pornește o explorare *Lee* în toate cele 4 direcții (sus, jos, stânga, dreapta) până când se vizitează toți pixelii albi aparținând aceleiași arii. Algoritmul folosește o coadă de noduri pentru a marca în lățime (nu adâncime) noii vecini (pixeli adiacenți) ce urmează a fi explorați și o matrice de vizită care marchează vizita fiecărui pixel (pentru a nu cicla prin aceleași poziții).
3. Algoritmul se oprește când toți pixelii albi sunt vizitați și odată cu încheierea acestuia, în funcție de cantitatea de pixeli pentru fiecare zonă descoperită, știm și dimensiunile ariilor detectate.

Concluzii

Consider că acest proiect și-a atins obiectivul principal, acela de a traduce inteligibil în timp real semnale luminoase Morse, iar motivația descrisă în incipit m-a ajutat să aduc aplicația la stadiul de maturitate dorit. Biblioteca **libmorse** este complet documentată și testată, ușor de dezvoltat profesional și fără *bug*-uri cunoscute, iar aplicația **Morseus** captează, detectează și afișează semnalele primite cu o acuratețe destul de mare cât să-mi permită satisfacția redactării acestei lucrări de licență.

Planuri de viitor

Încă nu am reușit să ating performanța ideală a aplicației, aceea de a detecta în orice împrejurări semnale umane trimise de la distanță, problemele și gândurile actuale fiind listate în fișierul *TODO* al proiectului. Totodată, încă se lucrează la portarea aplicației pe platforma Android, deoarece cu ideea aceasta am plecat în minte în momentul conceperii proiectului, problemele actuale învârtindu-se în jurul compatibilității și sprijinului framework-ului Kivy cu privire la accesul camerei video pe un telefon mobil.

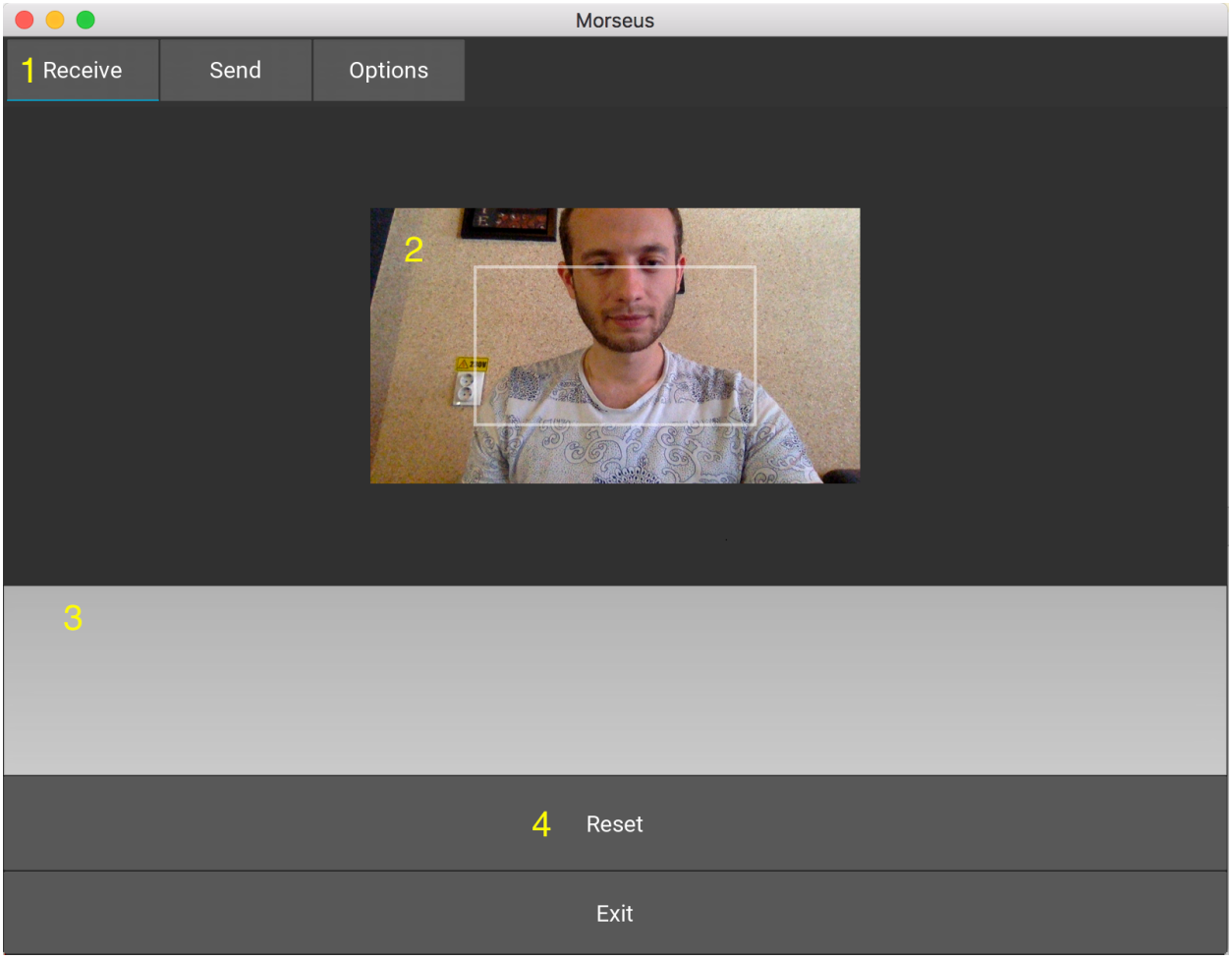
Cu toate acestea, aplicația exploatează un concept robust și versatil de transformare și procesare a intrării video, dar și de analiză și clasificare în cele din urmă a semnalelor reduse la text inteligibil, iar problemele listate mai sus sunt doar o chestiune de timp și priorități până vor fi rezolvate.

III. Manual de utilizare

Întrucât interfața aplicației este intuitivă, urmând îndeaproape bunele practici de UI & UX, un astfel de manual este cvasi-inutil, totuși, vom detalia mai jos fiecare componentă a interfeței cu exemple și situații uzuale.

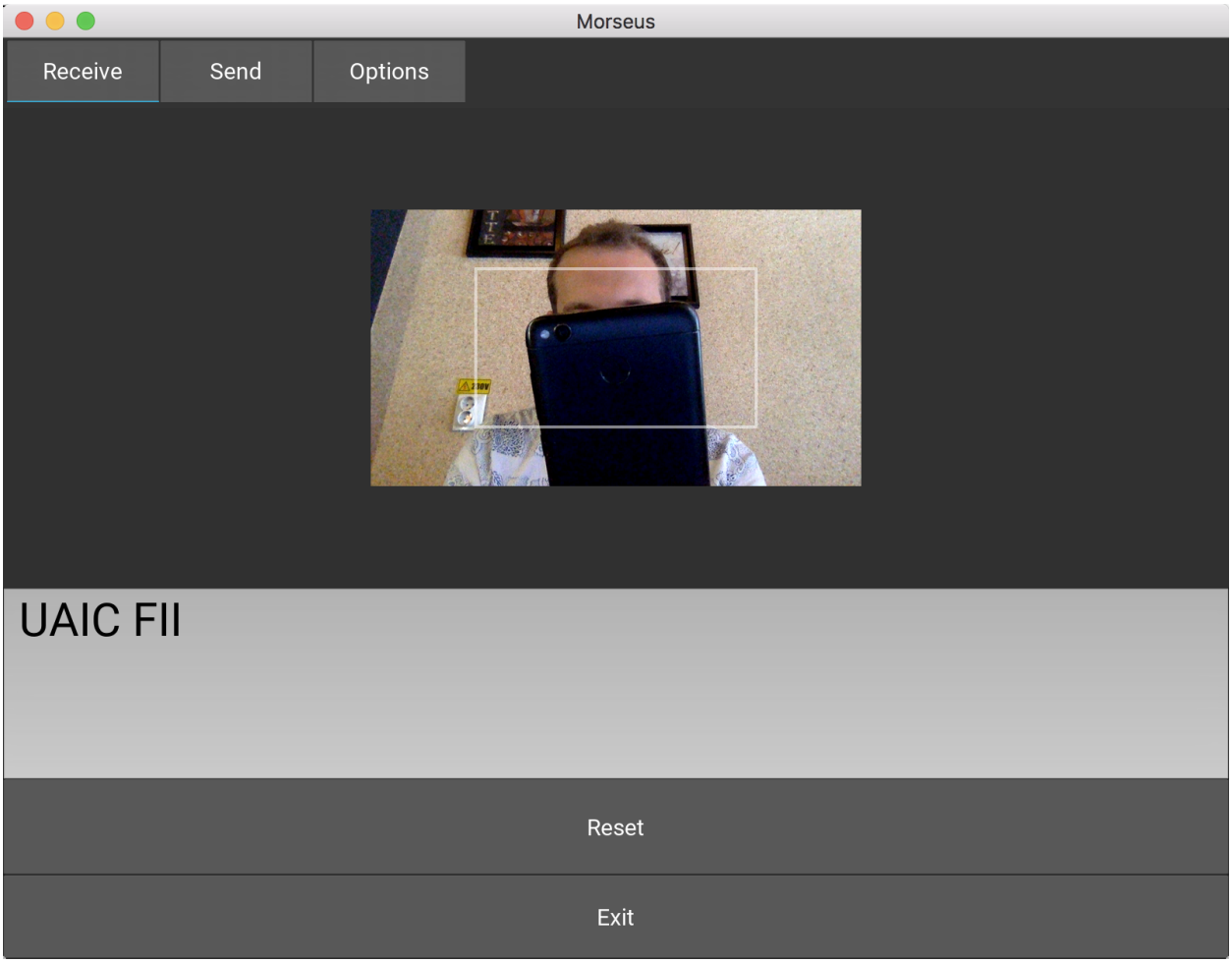
Ecranul principal conține un carnet cu 3 file (*Receive, Send, Options*), prezent în orice activitate selectată, de aceea voi trata fiecare filă în parte pentru toate cele 3 cazuri, respectiv secțiuni ale acestui manual. Butonul **Exit** este în permanență prezent și la apăsarea acestuia programul își va încheia activitatea odată cu distrugerea interfeței.

Receive



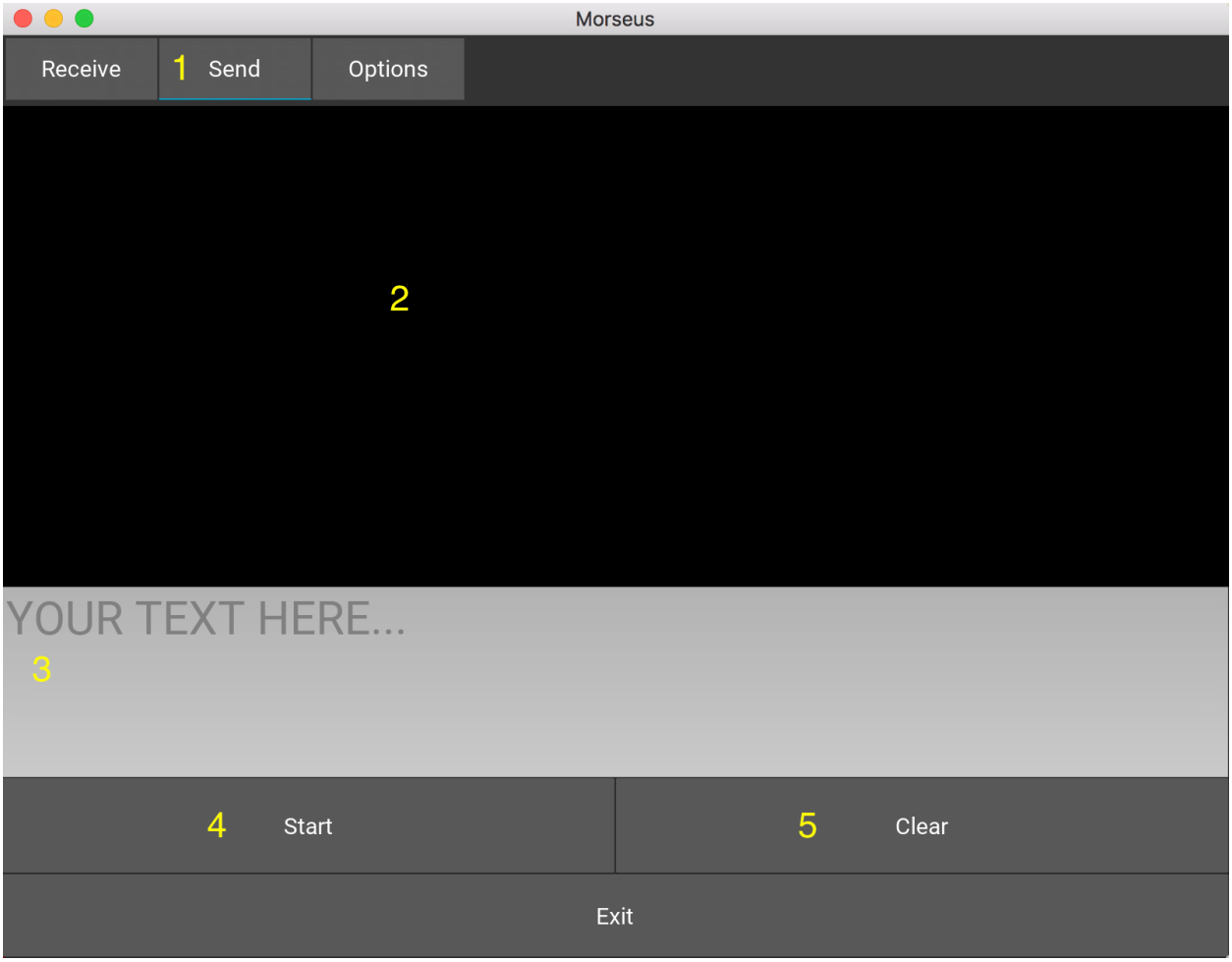
Aceasta este fila principală [1] ce se deschide implicit odată cu aplicația și încă de la apariția intrării video [2], capturile sunt făcute din sursa video de intrare și analizate în mod automat pentru a detecta prezența/absența semnalelor luminoase.

Obiectivul utilizatorului este să îndrepte chenarul alb [2], afișat peste cameră, către sursa principală de lumină emitentă de semnale și ideal să centreze respectiva sursa punctiformă/circulară în interiorul acestuia pentru o acuratețe mai mare. După câteva semnale analizate la început de sesiune, în caseta text de ieșire [3] (gri deschis, de sub afișajul capturii video) va apărea textul descifrat până în momentul curent, apoi în timp real, noi caractere pe măsură ce se primesc noi semnale luminoase, ca în exemplul de mai jos:

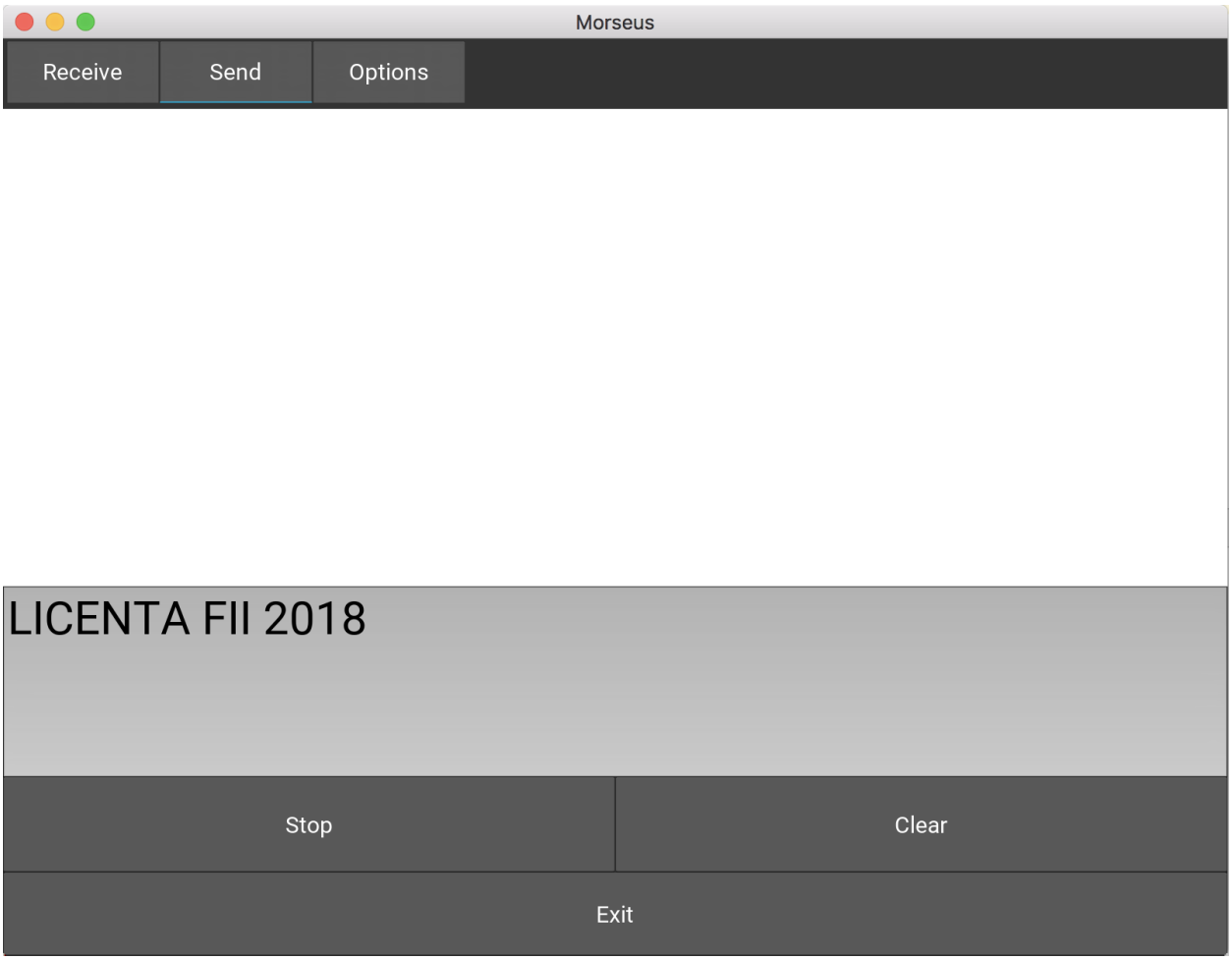


În orice moment al rulării, se poate apăsa butonul **Reset** [4] ce va conduce la curățarea casetei text (ștergerea întregului conținut) și totodată la reinițializarea camerei și instanței de decodificare a mesajelor, astfel nu se va mai ține cont învățarea anterioară, în funcție de care se ajustează și descifrează într-o manieră continuă datele din prezent.

Send



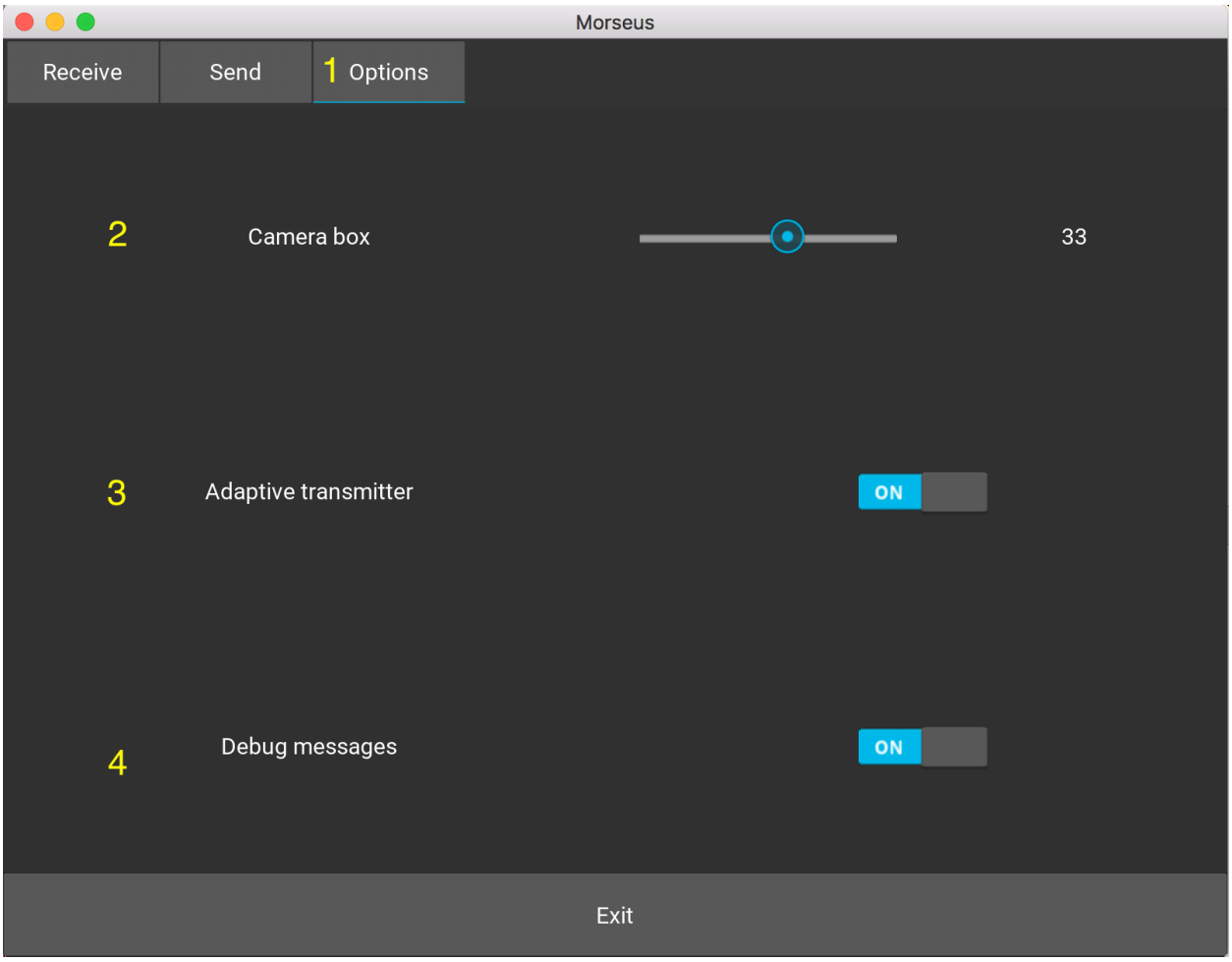
În această filă [1] se pot trimite semnale Morse interlocutorului direct din interfață. Chenarul negru [2] se va “aprinde” în alb de fiecare dată când se trimite un punct sau o linie și va rămâne negru în pauzele dintre acestea. Pentru a da curs acestei operațiuni, va trebui mai întâi să introduceți textul dorit în caseta text de intrare [3], apoi să se apese butonul **Start** [4] ce va iniția trimiterea semnalelor. Transmisiunea poate fi oricând oprită prin acționarea aceluiași buton ce va purta denumirea de **Stop** în momentul trimiterii active de astfel de semnale. În exemplul de mai jos am introdus un text și acționat butonul de trimitere:



Mai sus se poate observa textul dorit introdus și schimbarea denumirii butonului din **Start** în **Stop** ce poate fi acționat pentru oprirea procesului de trimis semnale. Activitatea acestui proces poate fi observată și prin culoarea schimbată a chenarului de mai sus, din negru în alb, de fiecare dată când se trimite un simbol Morse. În cazul aplicației de pe mobil, pe lângă colorarea chenarului, se mai aprinde concomitent blițul dispozitivului, acesta reprezentând un emițător eficace de semnale luminoase.

Oricând se poate apăsa butonul **Clear** [5] pentru a șterge conținutul casetei text de intrare.

Options



Acesta este meniul de opțiuni [1] ce are deja setată implicit configurația optimă a aplicației, nefiind nevoie astfel de vreo schimbare din partea utilizatorului. Chiar și așa, programul oferă acestuia posibilitatea de a ajusta comportamentul aplicației prin următoarele opțiuni:

- *Camera box* [2]: mărește/micșorează **aria centrală** de focalizare a semnalelor captate și luate în considerare, unde valoarea afișată de cursor este direct proporțională cu dimensiunile acesteia.
- *Adaptive transmitter* [3]: face comunicarea cu interlocutorul pe “înțelesul” acestuia în cazul activării acestei preferințe, mai exact, textul de intrare va fi trimis cu aceleași rate și ratii la care s-a făcut învățarea în timpul unei primiri anterioare de semnale.
- *Debug messages* [4]: în cazul afișării unei console, se optează sau nu pentru afișarea mesajelor de depanare în interiorul acesteia.

Referințe

Pentru crearea produsului și redactarea lucrării de licență nu a existat nevoia de prea multe resurse și materiale, căci protocolul de comunicare Morse este atât simplu cât și cunoscut, iar pentru clasificarea semnalelor și analiza video am creat algoritmi proprii (deși pot coincide din întâmplare parțial sau integral cu standarde publicate), unde singura sursă de informație a fost reprezentată de cunoștințele proprii în programare, gândire analitică și bune deprinderi din cadrul orelor de ML de la facultate.

Bineînțeles, pe parcursul dezvoltării codului, a interfeței și a materialelor aferente lucrării, m-am ajutat de documentație, biblioteci cu sursă liberă și varii utilitare necesare în îndeplinirea obiectivelor propuse. Acestea au fost amintite mai jos și prioritizate în funcție de importanță și impact.

Bibliografie

- Codul Morse: http://www.instructables.com/id/Morse-Codes/
- Arbore lexicografic: http://www.cranburyscouts.org/MorseTree.htm
- Documentație:
    - K-means (SciPy): https://docs.scipy.org/doc/scipy/reference/cluster.vq.html
    - Arrays, uniform etc. (Numpy): https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.array.html
    - Manipulare și procesare imagini (PIL): https://pillow.readthedocs.io/en/5.1.x/
    - Interfață (Kivy): https://kivy.org/docs/
    - Logică și software (Python): https://docs.python.org/2.7/index.html
- Utilitare:
    - Diagrame UML (draw.io): https://www.draw.io
    - Găzduire materiale (Google Drive): https://drive.google.com/
    - Sisteme de operare: Mac (dezvoltare), Linux (concepere), Windows (testare)

Note de final

Întregul proiect format din *cercetare, idei, algoritmi, documentație, imagini, cod sursă, arhitectură* și *resurse* îmi aparține în totalitate și reprezintă contribuție pur proprie în procent de 100%, fără să mă fi inspirat și/sau copiat din vreun material public sau privat, cu excepția adreselor de mai sus, prin intermediul cărora **doar** mi-am validat și întărit varii cunoștințe în domeniu, relevante subiectului. Totodată, am folosit biblioteci standard și terțe (amintite mai sus) pentru a nu “reinventa roata” în cazul unor algoritmi și tehnici ale căror principii și modalități de funcționare și implementare îmi sunt deja cunoscute.

Adrese utile

Mai jos avem câteva adrese utile cu referire la proiect:

- Blog: https://cosminpoieana.wordpress.com/
- Lucrare de licență: https://goo.gl/H6JfTw
- Aplicația **Morseus**: https://github.com/cmin764/morseus
- Biblioteca **libmorse**: https://github.com/cmin764/libmorse