

**Tema 1 -Arrow Puzzle Game**

Model – View – Presenter

Problema 35

**Proiectare Sofware**

**-Calculatoare si Tehnologia Informatiei-**

Student: Iamnitchi Bogdan

Grupa: 30235

An: 3

1. **Descrierea problemei**

În cadrul proiectului meu, am dezvoltat o aplicație Python care reprezintă jocul Red Light, Green Light, cunoscut și sub numele de "Kocoum" din Squid Game. Acest proiect se axează pe implementarea acestui joc popular folosind biblioteca OpenCV pentru capturarea și analiza fluxului video.

În cadrul aplicației mele, am simulat mediul de joc și am creat un algoritm pentru detectarea mișcării jucătorului. Prin utilizarea tehnicii de frame differencing, am putut identifica mișcarea în timp real și am permis jucătorului să se deplaseze înainte atâta timp cât era detectată mișcare.

In acest joc exista o persoana care conduce jocul (in cazul meu laptopul) si unul sau mai multi care joaca, acestia fiind asezati la o distanta relativ mare de cel care initiaza jocul, si scopul final e sa ajunga la aceasta fara a fi surprins in miscare. Prima data persoana ce organizeaza sa intoarce cu spatele la ceilalti si incepe sa zica: „Green Light!” atunci toata lumea poate sa fuga catre persoana care da comenzile, la un moment anume aleator, persoana care initiaza jocul zice: „Red Light!” si se intoarce catre participanti. In acest moment toata lumea trebuie sa fie nemiscata, in caz ca acesta este surprins in miscare de catre persoana ce organizeaza jocul acesta este eliminat.



**Figura 1:** Imagine preluata din serialul Squid Games cu jocul Red Light Green Light

1. **Proiectare**

Pentru a implemeneta acest proiect o sa folosesc camera de la laptop impreuna cu o interfata grafica prin care utilizatorul poate sa dea start la joc sau sa isi testeze camera, atunci cand se afla la distanta potrivita de laptop jocul va incepe. Green Light va fi semnalat printr-un semnal sonor dar si vizual aplicatia afisand papusa cu mainele la ochi ceea ce inseamna ca ea nu va vedea ce miscari faci. Red Light o sa vina intr-un moment random decis de aplicatie, cu atat mai rapid cu cat te apropi de final si va fi semnalat printr-un semnal sonor si vizual papusa fiind acum cu ochi deschisi, se afiseaza ce vede camera iar cat timp jucatorul nu se misca este desenat un dreptunghi verde in jurul jucatorului. Daca atunci cand s-a stringat Red Light esti surprins in miscare, atunci jucatorul va fi incercuit cu un dreptunghi rosu se va auzi un sunet de pusca simbolizand eliminarea ta din joc.

Target-ul final este sa supravituiesti cat mai multe runde dar si sa ajungi la laptop si sa apesi tasta W, semnaland faptul ca ai casigat.

**Obiectivele pentru acest proiect:**

• O interfata grafica simpla unde poti apasa pe start si eventual sa testezi ca merge camera si lumina buna

• Detectez ca e o persoana si nu un alt obiect care se intampla sa se miste pe acolo

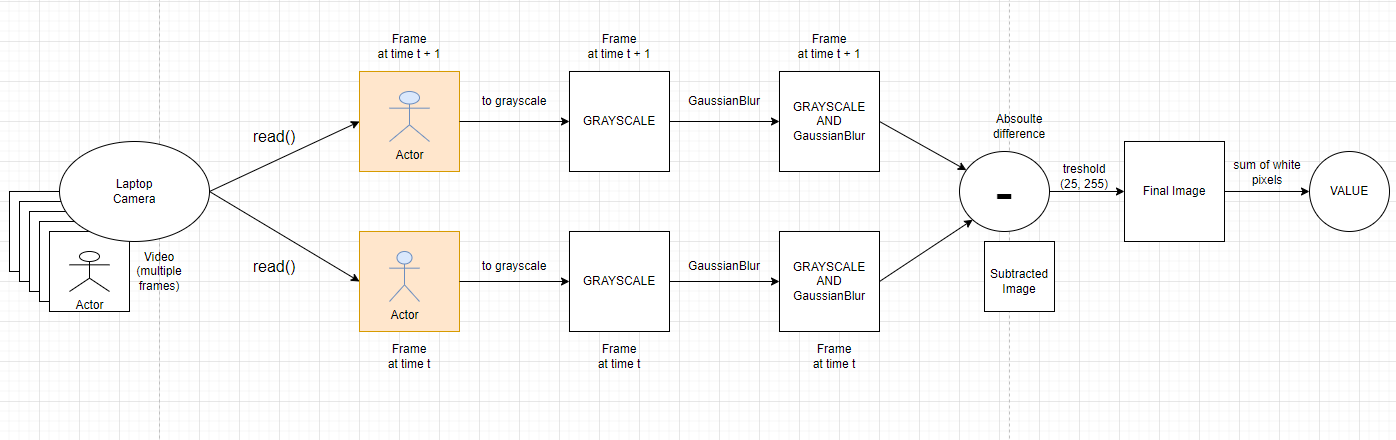
• Daca e o persoana si e la distanta corecta atunci porneste un countdown ca sa te pregatesti

• Programul afiseaza green light deci poti merge

• Programul afiseaza red light, acum detecetaza orice miscare

• Detectia de miscare prin metoda frame differencing in care se calculeaza numarul de pixeli diferiti dintre doua frame-uri consecutive(se va accepta un mic threshold), link aici, si aici

• Daca reusesti si apesi W atunci ai catsigat daca nu esti eliminat si trebuie sa dai restart jocului

**Figura 2:** Schema Bloc

1. **Implementare**

**Explicarea schemei bloc si a functionalitati proiectului**

1. Inițializare: Se începe prin inițializarea capturii video și a detectorului de poziții. Captura video este esențială pentru a obține imagini din camera web, în timp ce detectorul de poziții este folosit pentru a detecta pozițiile corpului în fiecare cadru al videoclipului.
2. Obținerea primului cadru și pregătirea acestuia: Se obține primul cadru din captură și se realizează o serie de prelucrări pentru a-l pregăti pentru comparația cu cadrele ulterioare. Acest proces include transformarea imaginii color în tonuri de gri (cv2.cvtColor) și aplicarea unui efect de blur (cv2.GaussianBlur). Scopul este de a reduce zgomotul din imagine și de a evidenția doar detaliile importante pentru detectarea mișcării.
3. Procesarea fiecărui cadru în buclă: În cadrul unei bucle infinite, fiecare cadru al capturii video este prelucrat. Aici, detectorul de poziții este utilizat pentru a identifica pozițiile corpului în fiecare cadru. În același timp, se aplică și prelucrări pe imaginea în tonuri de gri pentru a detecta mișcarea.
4. Detectarea mișcării: Pentru a detecta mișcarea, este calculată diferența între cadru și primul cadru pregătit. Aceasta se realizează prin aplicarea funcției cv2.absdiff între imaginea curentă și prima imagine. Apoi, se aplică un prag (thresholding) pentru a converti diferența într-o imagine binară, unde pixelii cu diferență semnificativă devin albi, iar ceilalți devin negri.
5. Verificarea pragului de mișcare: Este calculată suma valorilor pixelilor din imaginea binară (threshold). Această sumă reprezintă cantitatea totală de mișcare detectată în cadru. Dacă această sumă depășește un anumit prag (300 în acest caz), se activează un contor de alarmă. Acest contor de alarmă este utilizat pentru a evita declanșarea alarmei din cauza zgomotului sau fluctuațiilor minore în imagine.
6. Activarea alarmei: Dacă contorul de alarmă depășește un anumit număr de cadre consecutive (10 în acest caz), se declanșează alarma și se emite semnalul changeState către interfața grafică pentru a indica că jucătorul a pierdut jocul. În același timp, se desenează un dreptunghi roșu în jurul poziției detectate a corpului jucătorului pentru a indica că a fost detectată mișcare în momentul nepotrivit.
7. Actualizarea și emiterea imaginilor procesate: În fiecare iterație a buclei, imaginile procesate sunt actualizate și trimise către interfața grafică folosind semnalele changeColorImage și changeBWImage. Astfel, interfața grafică poate afișa imagini color și în tonuri de gri pentru vizualizarea mișcării și a detecției mișcării.

**Explicarea metodei alese “Frame Differencing”**

**Premize**

1) Camera statică

2) Obiectele se mișcă în mod uniform între fiecare cadru (adică nu se mișcă prea rapid)

**Metodă**

1) Obtinem diferența de cadre: d = cadru(t+1) – cadru(t)

2) Aplicam o valoare de prag pentru diferența de cadre pentru a obține o mască

3) Efectuam operații morfologice pentru a reduce zgomotul și pentru a evidenția mai bine obiectele în mișcare

4) Realizam detectarea pe mască prin găsirea contururilor

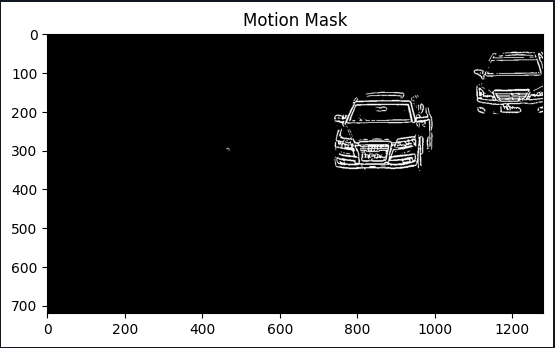
5) Suprimarea vreunui maxim dintre detecțiile propuse

6) Trasam casetele delimitatoare

Incepem prin citirea celor doua cadre si facem diferenta absoluta intre acestea aici putem vedea rezultatele obtinute.

**Figura 3:** Diferența absoulta intre doua cadre consecutive

Mai apoi cadrele urmeaza a fi prelucrate ca mai apoi să obținem o mască de imagine cu valori prag. Această mască de imagine ne va oferi locațiile relative ale tuturor țintelor în mișcare.



**Figura 4:** Masca cu valori prag

Următorul pas este să găsim contururile pe mască imaginei noastre. Aceste contururi ne vor ajuta să găsim obiectele mari în mișcare pe care dorim să le detectăm.

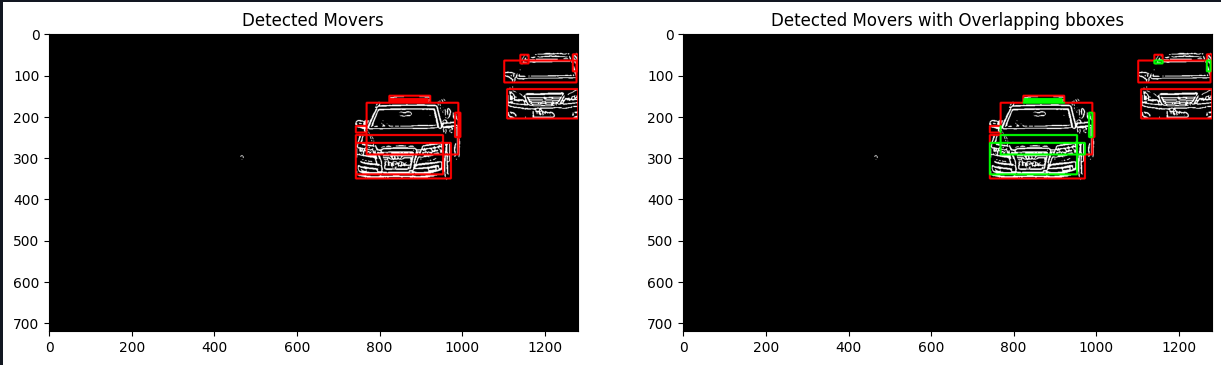
Apoi vom folosi contururile detectate pentru a găsi obiectele în mișcare. Pentru a face acest lucru, vom folosi funcția „findContours” pentru a obține locațiile casetelor delimitatoare ale detectărilor. Poate ai observat că imaginea mască nu a fost perfect conturata; în loc să conturam imaginea mască, vom folosi aria fiecărei casete delimitatoare detectate ca prag pentru a determina detectările inițiale. Majoritatea detectorilor de obiecte au un scor sau un nivel de încredere care de obicei variază între 0 și 1. În acest caz, nu avem niciunul dintre acestea, dar vom folosi un scor augmentat bazat pe aria fiecărei casete delimitatoare. Principala presupunere pentru acest lucru este că cel mai mare contur (și cea mai mare casetă delimitatoare) va corespunde de obicei obiectului real sau grupului de obiecte.

Următorul pas este să efectuăm suprimarea non-maximală. Mai întâi, vom avea nevoie de o metodă de calcul al Intersecției peste Uniune (IoU). Vom scrie funcții pentru acest lucru și vom afișa zonele intersectate pe mască noastră RGB. Înainte să începem, vom nota toate casetele care se suprapun pe mască.

Vom adopta o abordare specializată pentru suprimarea non-maximală (NMS). În mod obișnuit, NMS elimină casetele delimitatoare care au o suprapunere care depășește un prag predefinit. În acest caz, dorim să adaptăm NMS la cazul nostru de detectare a contururilor. După cum se poate vedea în imaginea de mai sus, există multe casete mai mici conținute în casetele mai mari, iar noi dorim să eliminăm toate aceste casete mai mici.

Vom folosi următorul algoritm pentru a implementa suprimarea non-maximală:

* Sortăm casetele detectate în ordine descrescătoare a scorului și le stocăm într-un tablou
* Eliminăm casetele care sunt complet cuprinse într-o altă casetă
* Începem cu cea mai mare casetă
* O comparăm cu toate celelalte casete
* Dacă IoU depășește pragul, o eliminăm
* Reiterăm pentru următoarea cea mai mare casetă rămasă



**Figura 5:** Detectarea și eliminarea dreptunghiurilor care se suprapun

1. **Modul de utilizare a aplicatiei**

Primul pas este sa clonam repository-ul de github local pe calculatorul nostru:

- git clone git@github.com:bogdan-iamnitchi-school/ip-opencv-squidgame.git

Aplicația este făcută în python și pentru a o rula trebuie mai întâi sa instalam pachetele necesare care le regasim în requirements.txt prin comanda:

- pip install -r requirements.txt

După ce toate pachetele sunt instalate tot ce trebuie să facem e sa rulam aplicația aceasta poate fi făcută utilizand comanda:

- python main.py



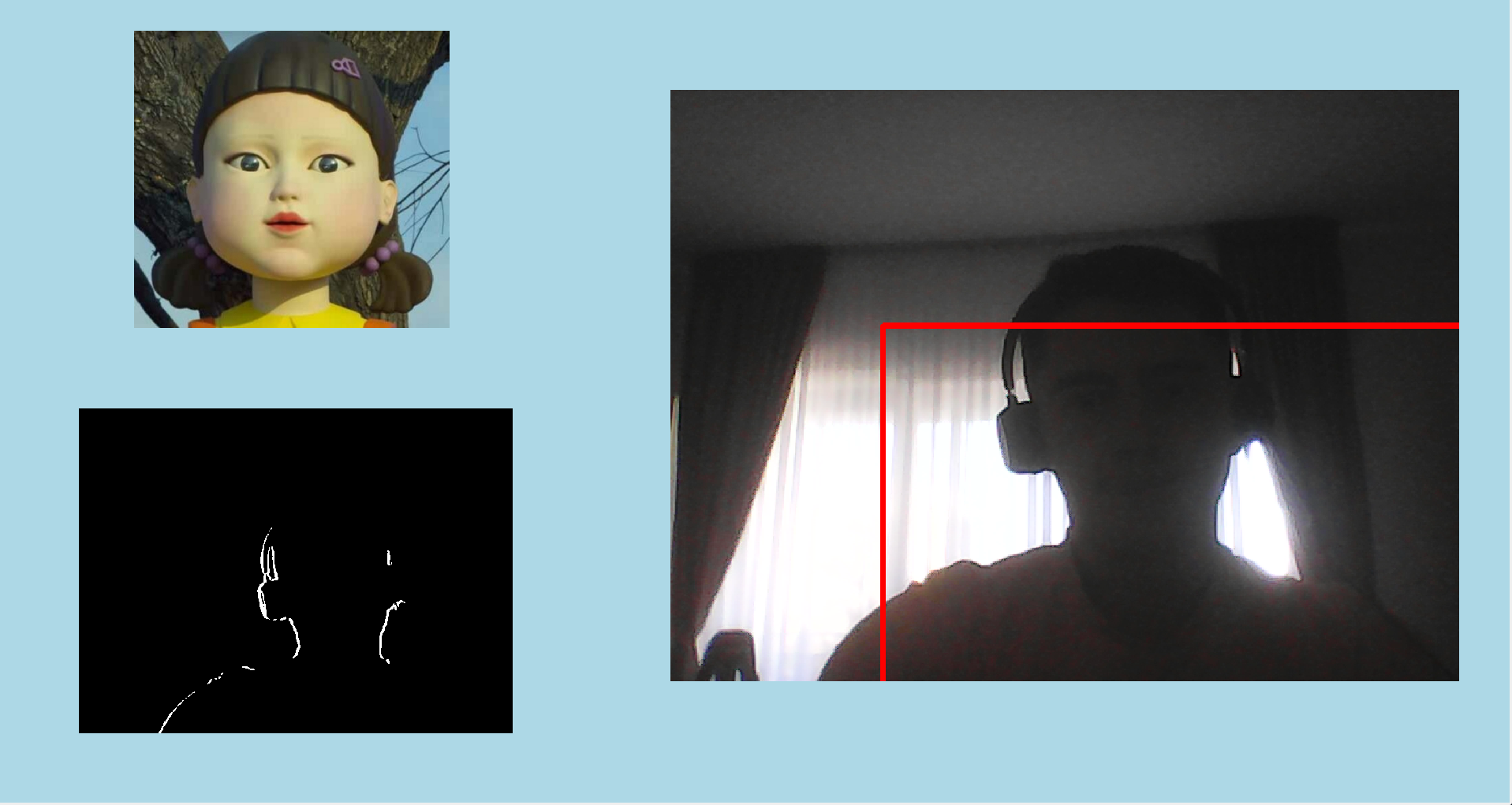
**Figura 6:** Pagina principala a aplicației

Pe aceasta pagina găsim 3 butoane primul care începe jocul propriu-zis al doilea folosit pentru a testa camera și lumina din camera, iar al 3 lea pentru a ieși din aplicație. Daca apasam play prima data păpușa se va ascunde iar juactori se pot mișca linisititi timp de câteva secunde.



**Figura 7:** Green lIght (jucătorul se poate mișca liniștit)

După câteva secunde păpușa se întoarce și verifica ce jucători se mișca din acel moment, aceasta este etapa echivalenta

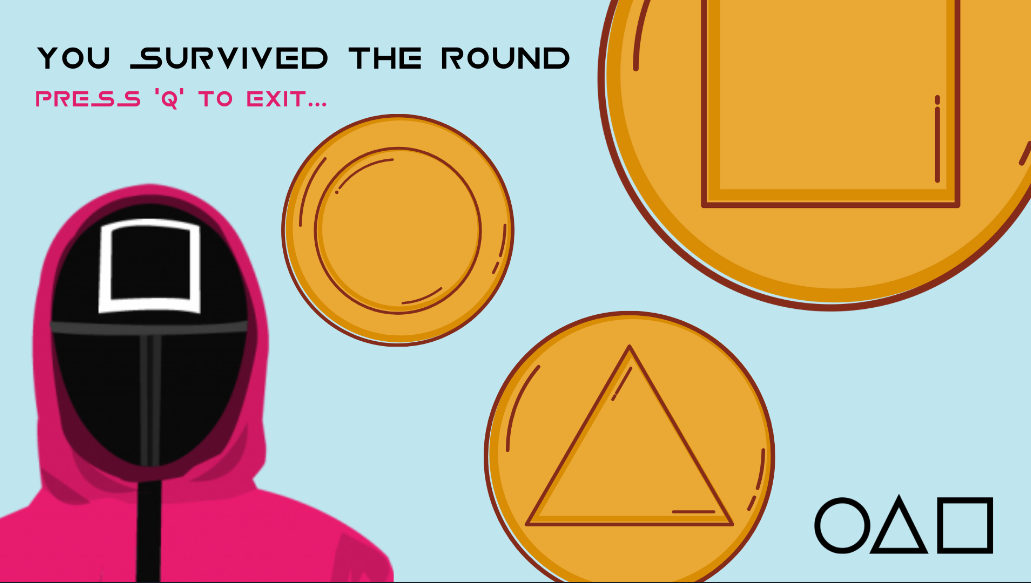


**Figura 8:** Red lIght (jucătorul este monitorizat)

În acest moment al jocului păpușa se întoarce către jucător camera este activata iar monitorizarea miscari este pornita, spre exemplu eu m-am mișcat, intre câteva cadre din captura video de la camera s-a aplicat algoritmul descris în partea de implementare care a făcut diferența absoulta a imaginilor grayscale detectand astfel ce pixeli sunt diferiți din imaginile consecutive. Acest lucru se poate observa în stânga jos unde chiar acest lucru am afisat, regiunea alba reprezintă pixeli care sunt diferiți de la un cadru la altul. Aplicația face aria acestei regiuni alge și daca a trecut de un treshold prestabilit atunci se decide ca jucătorul s-a mișcat și este împușcat astfel scos din joc, daca jucătorul nu s-a mișcat atunci el câștiga jocul.



**Figura 9:** Pagina de Game Over



**Figura 10:** Pagina de WIN

1. **Bibliografie**

* SQUID GAME | RED LIGHT GREEN LIGHT SCENE

https://www.youtube.com/watch?v=sH4Y450PSVM&t

* Pentru implementarea interfetei grafice Qt for Python

https://doc.qt.io/qtforpython-6/tutorials/index.html

* Motion Detection with Frame Differencing

https://medium.com/@itberrios6/introduction-to-motion-detection-part-1-e031b0bb9bb2

* Motion Detection Alarm în Python

<https://www.youtube.com/watch?v=QPjPyUJeYYE&t>