Uso di OpenCV per addestrare un calcolatore a riconoscere le carte e a giocarci

Laureando
Luca Sannino
1542194

Relatore
Roberto Capobianco

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Dipartimento di Ingegneria Informatica, Automatica e Gestionale
Corso di laurea in Ingegneria Informatica e Automatica

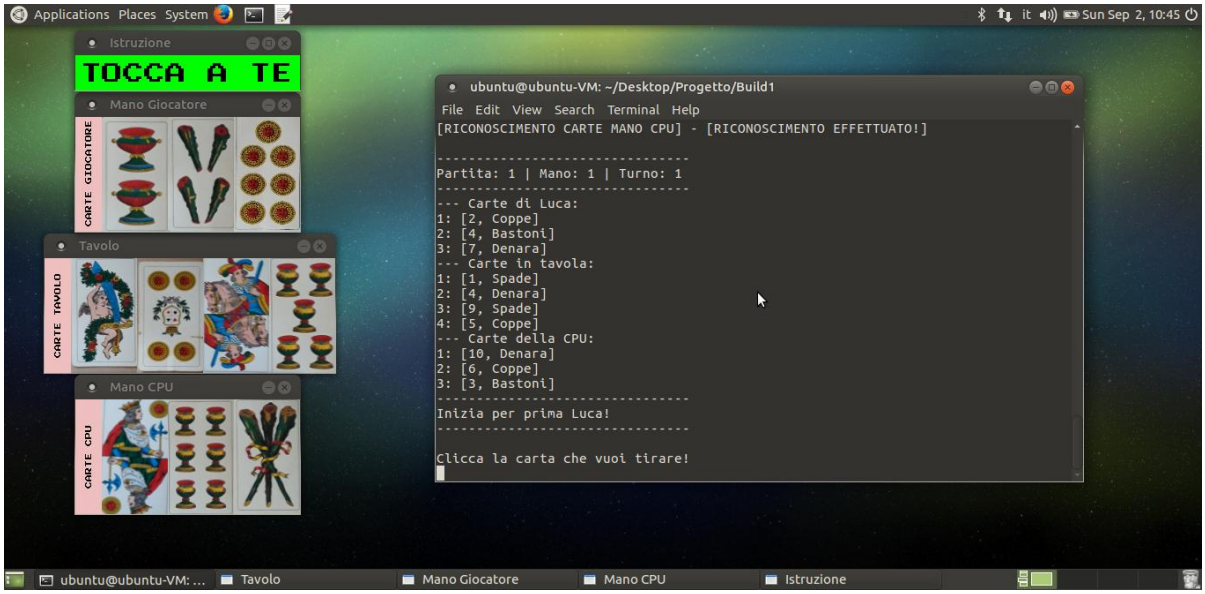
SAPIENZA
UNIVERSITÀ DI ROMA

A/A 2017/2018

Indice

1. Introduzione al progetto

Il progetto consiste nella creazione del classico gioco di carte Scopa, scritto interamente in C++. Rispetto ad altri programmi alternativi, questa versione si distingue per la capacità dell'utente di fornire in input al programma delle foto che rappresentano le sue carte appoggiate su superfici, quali tavoli e pavimenti. In seguito, durante l'esecuzione del programma, tali foto saranno analizzate grazie agli strumenti di individuazione degli oggetti offerti dalle librerie di OpenCV, in modo che le carte al loro interno siano individuate, riconosciute e implementate nella partita stessa.
Essendo ovviamente un gioco per due giocatori, l'utente sfiderà una CPU avversaria che basa dinamicamente le sue scelte in base a vari fattori, come le carte che sono state riconosciute in quel turno, quelle già precedentemente uscite e quelle che si trovano sul tavolo.
Il programma è provvisto di una GUI grafica, sempre realizzata con OpenCV, con cui l'utente può osservare le condizioni della partita in corso e fare le sue mosse. Sul terminale dei comandi, invece, apparirà un "log di gioco" con il compito di trascrivere ogni singolo evento accaduto dal primo avvio del programma.
Delle opzioni nel menu principale offrono all'utente la possibilità di giocare a carte scoperte o a carte coperte, o di testare la validità delle foto date in input.



Il gioco in azione

1.1 Motivazioni dietro al progetto

L'idea dietro al programma nasce dalla volontà di approfondire e affinare alcune delle conoscenze acquisite durante la sezione di Grafica Interattiva del Laboratorio di Intelligenza Artificiale, in particolare quelle riguardanti l'utilizzo di OpenCV per risolvere

problemi nel campo della detection e della recognition degli oggetti o per realizzare delle GUI con cui l'utente può interagire. In particolare, l'obiettivo è stato quella di testare la versatilità e le potenzialità di OpenCV con la creazione di un gioco in cui viene sfidata una CPU che effettua le sue strategie unicamente in base a delle informazioni che vengono ottenute dopo un processo di object recognition. Informazioni che quindi non sono già impiantate nel codice anticipatamente, ma che vengono scoperte nel momento stesso in cui vengono analizzate le foto ricevute come input, a tempo di esecuzione del programma.

1.2 Brevi considerazioni sugli strumenti scelti

Il programma è stato scritto in C++, uno dei linguaggi object oriented più popolari al mondo[1]: la necessità della scelta di un linguaggio di tale tipo deriva dalla facilità con cui è possibile realizzare classi che rappresentino i concetti di carta e mazzo; concetti che, grazie alla modularità delle classi stesse, si prestano al riutilizzo, con le lievi modifiche del caso, in potenziali giochi successivi sempre basati sull'utilizzo di carte.
Ovviamente ci sono altri linguaggi che offrono servizi simili, tra cui Java e Python, ma C++ è stato scelto per il modo intuitivo con cui implementa le librerie di OpenCV, scritto a sua volta proprio in quel linguaggio[2].
OpenCV, invece, è stato selezionato perché offre degli strumenti per l'individuazione degli oggetti che sono particolarmente efficaci e spontanei da comprendere[3].
Inoltre OpenCV rende anche possibile l'implementazioni di GUI interattive, anche se l'opzioni a disposizione del programmatore sono relativamente scarne. Come ambiente di sviluppo è stato utilizzata la Virtual Machine offerta dal corso, al cui interno si trova una versione di Ubuntu con tutti i software necessari già pre-installati[4].

1.3 Sintesi del lavoro svolto

Dopo aver analizzato una serie di tecnologie per svolgere le operazioni di object detection e recognition, la prima parte del progetto ha riguardato la creazione di una serie di files .xml, chiamati Cascade Classifiers, in grado di individuare le carte all'interno di una foto. In seguito, dopo un accurato processo di testing, si è scelto quello più performante tra quelli generati e si è passati alla fase successiva: la realizzazione del gioco stesso.
Qui sono state affrontate e risolte varie problematiche, la prima delle quali è stata la creazione di classi che rappresentassero le entità di carte e mazzi. Successivamente si è dovuta creare una logica di gioco in grado di emulare perfettamente le regole e i meccanismi dietro alla Scopa. Altre sfide sono state la creazione di un'IA avversaria, che utilizzasse tattiche in grado di mettere in difficoltà il giocatore, e di un sistema di recognition, che fosse in grado di specificare il valore e il seme delle carte trovate durante una object detection.
Ultimato il gioco, si è deciso di crearne una seconda versione che accettasse gli input

dell'utente tramite una GUI, anziché dal terminale dei comandi. Durante queste fasi finali è stato anche introdotto un menù principale.
Si è deciso di trattare l'analisi delle possibili soluzioni al problema della object detection nel Capitolo 2, mentre il training di un Cascade Classifiers viene descritto nel Capitolo 3; le operazioni dietro alla loro creazione sono completamente scollegate dalla realizzazione del gioco stesso, che viene invece affrontata nel Capitolo 4. Il Capitolo 5, infine, mostra come è stata implementata la GUI, mentre il Capitolo 6 contiene una breve conclusione.

1.4 Istruzioni per l'uso e requisiti

Per compilare il programma, che è compatibile solo su sistemi Linux, basta spostarsi nella sua directory con il terminale di comando, eseguire CMake su CMakeLists.txt e poi avviare il main che si trova in Scopa.cpp.
Ovviamente è necessario che l'utente abbia installato C++ e OpenCV sul suo computer.
Le cartelle Target e Grafica contengono elementi essenziali per il funzionamento del programma, rispettivamente nel campo del riconoscimento delle carte e la realizzazione della GUI interattiva, quindi la rimozione di elementi al loro interno porterà a malfunzionamenti durante l'esecuzione, tutti gestiti da apposite eccezioni.
Le cartelle CarteA, CarteB e CarteC contengono invece le foto delle carte inserite dall'utente e che saranno utilizzate nel corso del gioco. Il programma leggerà 40 foto da ciascuna cartella, ognuna teoricamente contenente una carta differente, e le analizzerà alla ricerca delle stesse. In caso di carte mancanti, o doppioni, il programma riporterà un'eccezione; se invece il tipo di carte è uguale o molto simile a quello supportato dal programma (carte piacentine), saranno allora riconosciute, salvo eventuali casi di falsi negativi.
Per ottenere risultati migliori, si consiglia di porre le proprie carte in maniera dritta su una superficie piana, come un tavolo, in condizioni di ottima illuminazione.

Directory del programma:
/Scopa
/Carta.hpp
/Carta.cpp
/Mazzo.hpp
/Mazzo.cpp
/Giocatore.hpp
/Giocatore.cpp
/Utente.hpp
/Utente.cpp
/CPU.hpp
/CPU.cpp
/ScopaUtil.hpp
/ScopaUtil.cpp
/Scopa.cpp
/CMakeLists.txt
/CarteX
/1.jpg
…
/Target
/1.jpg
…
/Grafica
/Menu1.jpg
…
/CarteCascade.xml

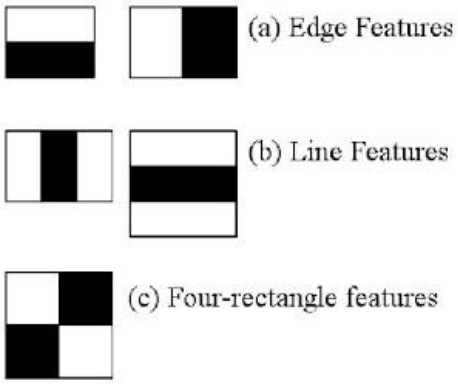
I nomi di queste foto devono essere del tipo "x.jpg", dove 'x' è un numero che va da 1 a 40, altrimenti non saranno lette. Per conoscere anticipatamente la validità delle foto al momento usate, si può decidere di testarle scegliendo l'appropriata opzione nel menù principale di gioco. Infine, CarteCascade.xml è il detector usato nella ricerca delle carte: anche la sua rimozione genererà un'eccezione e la terminazione del programma.

2. Possibili soluzioni e lo stato dell'arte

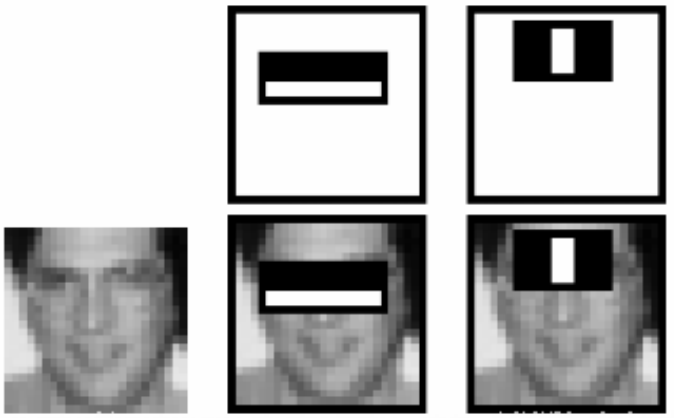
Esistono una varietà di tecnologie con lo scopo di risolvere problemi nel campo della object detection e recognition. L'obiettivo di questo capitolo è quindi di elencarne brevemente le più diffuse, riportando quella utilizzata in fase di progetto e un paio di soluzioni alternative. Di quest'ultime saranno elencati pro e contro, e sarà spiegato perché non sono state prese in considerazione.

2.1 Cascade Classifiers

I Cascade Classifiers sono uno degli strumenti più popolari di OpenCV per la object detection: questi detectors sono spesso basati sulle features Haar, ovvero dei piccoli rettangoli (in genere di 24x24 pixels) che vengono fatti scorrere per tutta la foto alla ricerca dell'oggetto che vogliamo trovare. I rettangoli sono a loro volta suddivisi in due rettangolini, uno bianco e uno nero. Quando un rettangolo viene posizionato nella foto, i valori dei pixel dell'immagine che corrispondono alla parte nera vengono sommati tra loro, per poi essere sottratti a quelli che corrispondono alla parte bianca. Se la somma così ottenuta viene ritenuta accettabile secondo dei criteri interni, allora viene decretato che dentro quel rettangolo si trova un'istanza di quello che stiamo cercando[5, 6, 7].



Esempi di rettangoli Haar



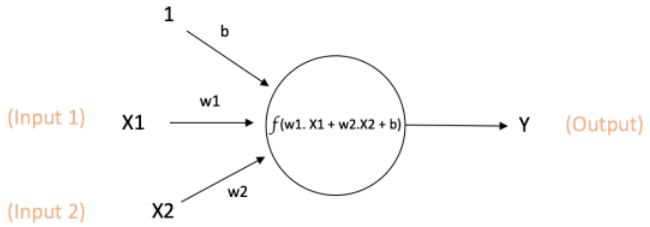
Rettangoli Haar applicati su una foto

Si parla di "Cascade" perché all'interno di un rettangolo 24x24 si possono già potenzialmente cercare più di 160000 features diverse: per diminuire i tempi di ricerca queste features vengono quindi suddivise in una sequenza di stages, dalla complessità sempre crescente, in modo che se un livello dovesse produrre un risultato al di sotto di una soglia aspettata, viene immediatamente scartata la possibilità che l'oggetto possa trovarsi in quel rettangolo e le features appartenenti ai stages successivi non saranno applicate[5, 6, 7]. Un fattore da considerare è l'evenienza che l'oggetto da cercare nell'immagine sia sì presente, ma di dimensioni diverse rispetto a quelle per cui il Cascade è stato allenato. Dopo aver quindi applicato il rettangolo in ogni parte dell'immagine,

quest'ultima viene ridimensionata di un certo fattore e il processo ha di nuovo inizio, un ciclo che si ripeterà fino a quando le dimensioni dell'immagine saranno sufficientemente grandi da garantire una ricerca efficace dell'oggetto[6, 7].
I Cascades sono stati scelti perché il loro training tramite un dataset di immagini è particolarmente semplice, anche se faticoso, e non richiede prestazioni proibitive dal calcolatore. Non sono tuttavia efficaci per la recognition, per la quale si è deciso di trovare una soluzione alternativa tramite un confronto con delle speciali foto, chiamate Target, per scoprire il seme e il valore delle carte trovate. Maggiori informazioni saranno riportate nel paragrafo 4.3.

2.2 Reti neurali

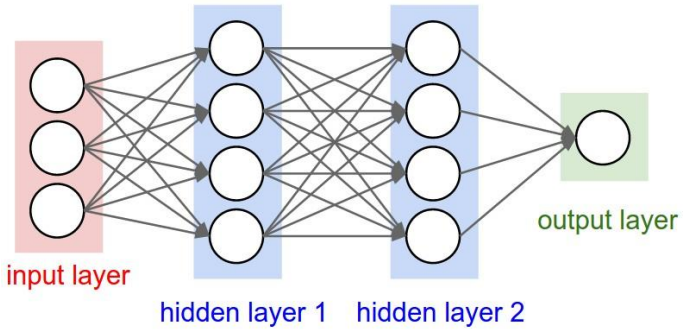
Lo stato dell'arte in materia è indubbiamente l'utilizzo di reti neurali per creare e allenare, tramite un dataset di immagini, dei classificatori in grado di distinguere degli oggetti all'interno di una foto, per poi assegnare loro una categoria di appartenenza. L'elemento base di queste reti è appunto il neurone, che accetta in input dei numeri a cui vengono associati dei pesi, più il bias. Successivamente, viene applicata una funzione di attivazione non-lineare sulla somma pesata dei numeri ricevuti in input. Se l'output della funzione supera una certa soglia, allora il neurone si attiva[8].



Template di un neurone

I neuroni sono disposti in un numero variabile di layers, tutti collegati tra loro, anche se non ci sono connessioni tra neuroni all'interno dello stesso strato. Il layer finale è quello di output, dove viene stimata la categoria di appartenenza di un'immagine: si nota che nell'ultimo layer ci sono tanti neuroni quante sono le categorie[8].
In alcuni tipi di reti neurali le immagini fornite in input sono "schiacciate" in una struttura ad una dimensione, in cui ciascun elemento contiene il valore di un solo pixel. I neuroni dell'input layer sono successivamente inizializzati con i valori di questa struttura[9].
Questi tipi di reti neurali, sebbene garantiscano processi di training relativamente rapidi, offrono delle prestazioni inferiori rispetto a quelle che fanno uso di convolutional layers, come ad esempio la rete YOLO, una delle configurazioni più popolari ed efficaci in

esistenza[10]. Questi strati impiegano dei filtri che simulano dei feature detector per effettuare convoluzioni sulle foto: in questo modo viene mantenuta la spazialità dell'immagine più a lungo, a discapito di un tempo di training considerevolmente più duraturo[11].



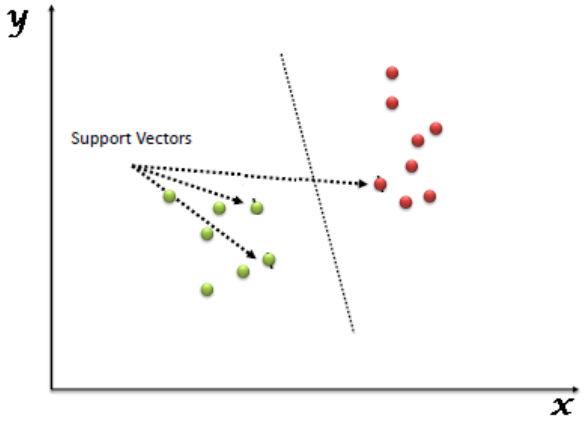
Esempio di rete neurale con 4 layers

Il primo passo nel training di un classificatore è l'inizializzazione in maniera casuale dei pesi: in seguito la rete si allena con il dataset di immagini fornito in input, per poi confrontare i risultati ottenuti dal layer di output con le categorie effettive. Se il tasso di errore è superiore a quello aspettato, allora si aggiornano i pesi con un processo di backpropagation per poi tentare nuovamente finchè non viene raggiunto quel margine di errore desiderato[8].
Un approccio tramite reti neurali avrebbe sicuramente reso più robusta la capacità di recognition del programma: i Cascades, come già menzionato nel precedente paragrafo, non offrono questa funzione. Usando le reti neurali, invece, sarebbe bastato associare ad ogni tipo di carta una sua specifica categoria.
D'altro canto ciò avrebbe richiesto la costruzione di un dataset di immagini di training più complesso e che avrebbe contenuto numerosi esempi per ogni tipo di carta, a differenza di quanto effettuato con i Cascades di OpenCV (maggiori informazioni nel capitolo 3): un processo che non sarebbe stato triviale, nonostante la possibilità di effettuare una data augmentation sulle proprie foto[12]. Il vero problema, tuttavia, sarebbe ricaduto nella mancanza di potenza computazionale del calcolatore usato durante la realizzazione del progetto: allenare un classificatore tramite rete neurale, infatti, è un processo costosissimo, specie se si fa uso di convoluzioni, per il quale viene spesso consigliato l'utilizzo di una GPU ben prestante, di cui, però, si è sprovvisti. Inoltre si sarebbe dovuta trovare la giusta quantità di layers e neuroni per ogni strato, rendendo quest'approccio meno desiderabile rispetto ad un'altro più intuitivo con i Cascades.

2.3 Support Vector Machines

Un altro metodo alternativo per classificare oggetti è tramite l'utilizzo di Support Vector Machines, che rappresentano le immagini che ricevono in input come punti nello spazio.

Questo spazio viene diviso in due sezioni dopo un processo di training, ognuna rappresentante una categoria differente: questo significa che alle immagini vengono assegnate le classi corrispondenti semplicemente osservando in quale parte dello spazio sono state rappresentate[13].



Rappresentazione grafica dell'output di una SVM

A differenza di quanto avviene con le reti neurali, il classificatore delle SVM è tipo binario, ovvero è in grado di stimare solo due categorie, mentre nel nostro caso ne sarebbero servite ben 40. Per estendere il numero di classi supportate è necessario dunque implementare delle soluzioni alternative, di cui ne vengono riportate un paio:
- **One vs. All**: creare per ogni categoria una SVM che specifica se un oggetto ricade in quella particolare categoria o nelle altre, assegnando un grado di confidenza nel caso lo faccia. All'oggetto viene infine assegnata la categoria con il grado di confidenza migliore[14].
- **One vs. One**: viene creata una SVM per ogni singola coppia di categorie. La classe di appartenenza è poi ottenuta combinando i risultati[14].

Come si può notare, nessuna delle due soluzioni è particolarmente intuitiva, poiché a quel punto sarebbe convenuto usare una rete neurale.
Per quanto riguarda la decisione tra Classifier e SVM, invece, la scelta finale è ricaduta sulle prime per preferenze personali.

3. Le basi

Prima di osservare gli aspetti più salienti del programma, quali la logica di gioco, l'IA della CPU avversaria, il riconoscimento delle carte stesse o come è stata realizzata l'interfaccia grafica, è opportuno fare un passo indietro e analizzare il fondamento sul quale tutto il progetto si regge in piedi: la creazione di un Cascade.

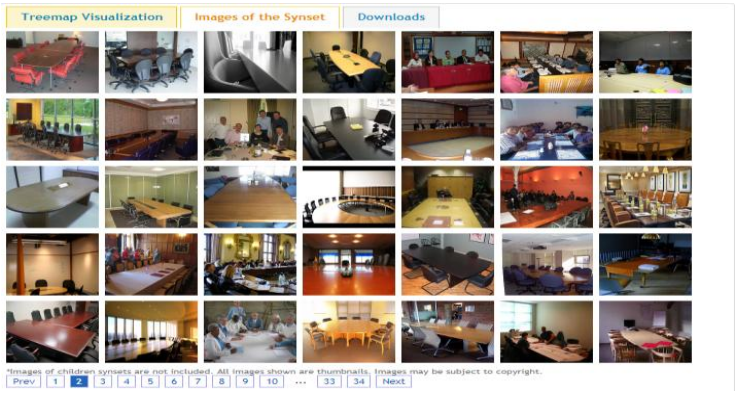
3.1 Creazione di nuovo Cascade Classifier

Appena installato, OpenCV offre già dei Cascades completamente allenati e pronti all'utilizzo[15], ma dato che sono specializzati nel trovare features facciali, quali occhi, naso e orecchie, sono di ben poca utilità per questo progetto: è stato quindi necessario crearne uno adatto al riconoscimento delle carte. La fase di training di un nuovo Cascade è un processo faticoso e che richiede tempo, non per la difficoltà in sé, ma per la raccolta di un numero considerevole di immagini, senza le quali il Cascade non potrebbe mai essere allenato in modo efficace[16]. Sono infatti servite decine di migliaia di foto che ritraggono persone di ogni nazionalità per allenare i Cascades citati precedentemente. Nel nostro caso, tuttavia, è stato sufficiente un numero inferiore di foto, in quanto le carte da gioco italiane sono degli oggetti fissi e immutabili, a differenza delle diverse features facciali che cambiano da persona a persona, la cui varietà può essere considerata praticamente infinita[16].

Ma non sono servite solo foto che ritraggono l'oggetto cercato: per allenare correttamente il Cascade sono necessarie anche delle immagini in cui l'oggetto è del tutto assente, che vengono chiamate "negative" o di "background". In genere questo foto possono essere di qualsiasi tipo, anche se è preferibile che raffigurino situazioni in cui di norma si trova l'oggetto che stiamo cercando: è per questo motivo che, nel nostro caso, sono state scelte tutte foto che mostrano tavoli o pavimenti, luoghi in cui ovviamente è solito porre le carte da gioco[16].

Fortunatamente non è stato necessario perdersi nel web alla ricerca di un numero intimidatorio di foto che soddisfacevano i nostri criteri: il sito ImageNet, infatti, mette a disposizione un database di oltre 14 milioni di link di immagini, tutti suddivisi per categorie, che ha semplificato notevolmente il processo di creazione di un training dataset[17, 18].

Dato che il sito non offre una via diretta per scaricare insieme tutte le immagini, si è deciso di utilizzare un programmino scritto in Python con il compito di scaricare una foto per volta, scorrendo tutti i link elencati da ImageNet per una particolare categoria, per poi ridurle di dimensioni (100x100 pixels) e salvarle in bianco e nero grazie a delle funzioni di OpenCV. Questo perché avere a che fare con delle immagini di dimensioni ridotte aiuta notevolmente i tempi di training, che altrimenti impiegherebbero giorni per terminare.



Categoria "conference table" del sito ImageNet

Dopo aver scaricato e ridimensionato circa 2000 foto raffiguranti tavolini e pavimenti, e dopo aver eliminato quelle foto irregolari ottenute da link invalidi (che si trovano frequentemente in ImageNet), è stato necessario creare una seconda collezione di immagini 100x100, ma questa volta "positive", ovvero che contengono l'oggetto che stiamo cercando: le carte da gioco[16]. ImageNet purtroppo non contiene un archivio di link di foto di carte italiane, e l'idea di scattare migliaia di foto alle nostre carte di casa non è tra le più agevoli: fortunatamente OpenCV permette la creazione automatica, per via di un comando sul terminale, di nuove immagini a partire da una foto "sample", data in input insieme ad un file di testo contenente una lista di directories di immagini già salvate localmente. Questa foto viene poi continuamente ruotata, distorta e ridimensionata in maniera casuale (anche se l'utente ha un certo controllo sulla severità di queste trasformazioni) e infine sovraimpressa nelle foto indicate dal file di testo[16].
Utilizzando una piccola foto "sample" di 37x70 pixels raffigurante un 6 di denari, e un file "bg.txt" che contiene la posizione delle foto negative, si è quindi creata una nuova collezione di immagini positive in cui la foto input è stata modificata e sovraimpressa sulle foto background scaricate precedentemente, salvando il risultato in nuovi files e lasciando le foto negative intatte.

```
ubuntu@ubuntu-VM:~/Desktop/Progetto/cascade_maker$ opencv_createsamples -img den
ara6.png -bg bg.txt -info pos/info.lst -pngoutput pos -maxxangle 0.5 -maxyangle
0.5 -maxzangle 0.5 -num 1974
```

Il comando per creare foto positive

La foto sample

Una foto negativa

Una foto positiva generata artificialmente

A termine del processo è stato automaticamente prodotto il file "info.lst", che indica il numero di carte e le coordinate della loro posizione per ogni singola nuova foto che è stata generata. Questo file è stato poi essenziale per la creazione, tramite un altro comando su terminale, di un vettore che contiene tutte le foto positive e che viene usato durante il training[16].

```
ubuntu@ubuntu-VM:~/Desktop/Progetto/cascade_maker$ opencv_createsamples -info po
s/info.lst -num 1974 -w 20 -h 20 -vec positives.vec
```

Il comando per creare il vettore di immagini positive

Il dataset è stato completato: circa 4000 immagini in bianco e nero di 100x100 pixels, metà positive e metà negative, con vettore associato e tutti i relativi file di testo.

Directory a fine processo:
/cascade_maker
  /neg
    /1.jpg
    /2.jpg
    /3.jpg
    …
  /bg.txt
  /pos
    /1.jpg
    /2.jpg
    /3.jpg
    …
    /info.lst
  /positives.vec

Con l'inserimento del terzo, ed ultimo, comando su terminale, si è potuto iniziare il training. Durante questa fase il Cascade usa le foto fornitegli per allenarsi ad individuare l'oggetto che stiamo cercando. Al termine del processo le sue prestazioni sono giudicate in base al suo hit rate, la cui soglia minima può essere quella di default o una scelta dall'utente. In caso di fallimento, il Cascade inizia un nuovo stage di training, che sarà più rigido e che impiegherà più tempo di quello precedente. E' da notare che i progressi di ogni stage sono salvati localmente, quindi non è necessario addestrare il Cascade in un'unica sessione. Infine, se uno stage dovesse produrre un risultato positivo, il processo termina con la generazione di un file in formato .xml: si tratta del Cascade stesso[16].
Generati i primi Classifiers, il passo successivo è stato verificare la loro effettiva validità con alcune foto delle nostre carte.

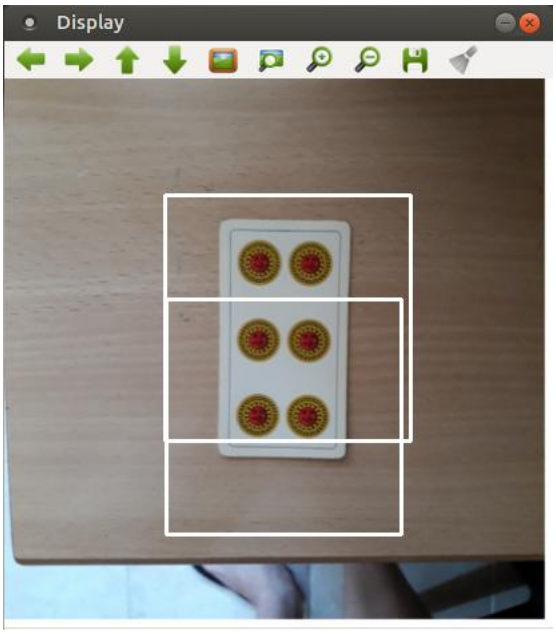
```
ubuntu@ubuntu-VM:~/Desktop/Progetto/cascade_maker$ opencv_traincascade -data dat
a -vec positives.vec -bg bg.txt -numPos 850 -numNeg 1900 -numStages 10 -w 20 -h
35
```

Comando per avviare la fase di training

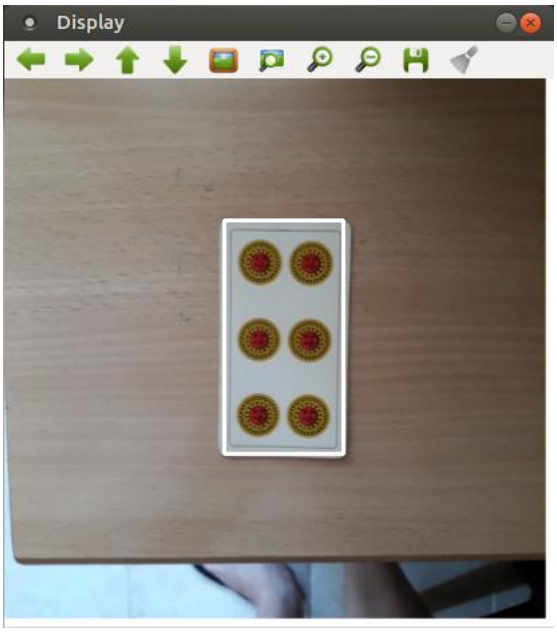
3.2 Verifica e testing di un nuovo Cascade Classifier

Dato che l'utente ha un notevole controllo sulle modalità di training, partendo dal numero di immagini positive e negative fornite, fino alle dimensioni del rettangolino che si occupa di individuare l'oggetto, è possibile creare una moltitudine di Classifiers diversi anche se si usa sempre lo stesso dataset di immagini. E dato che la documentazione ufficiale può essere poco chiara e intuitiva riguardo a come settare i parametri in base ad una specifica problematica, spesso l'unico modo per creare un Cascade che soddisfi veramente l'esigenze dell'utente è attraverso un processo di "trial & error".
Nel corso del progetto sono stati infatti creati circa una decina di Cascades, con delle

prestazioni molto diverse tra loro, finché alla fine non è stato scelto quello che era il più efficiente nella maggior parte delle situazioni. Si sottolinea "maggior parte delle situazioni" perché, non importa la qualità di un determinato Cascade, possono sempre verificarsi casi di falsi positivi, ovvero quando un rettangolino Haar determina che al suo interno si trova l'oggetto che stiamo cercando ma invece non è così, o di falsi negativi, quando al contrario l'oggetto richiesto è presente nell'immagine ma non viene mai individuato, nel nostro caso quando una carta è troppo vicina o troppo lontana dall'obiettivo della camera.
Nella creazione dei vari Cascades si è notato che i più prestanti sono quelli in cui una categoria di immagini è superiore del doppio rispetto all'altra. Tuttavia tra un Cascade in cui le foto positive sono il doppio di quelle negative, ed un altro in cui accade il viceversa, si sono trovate differenze minuscole, se non insignificanti.



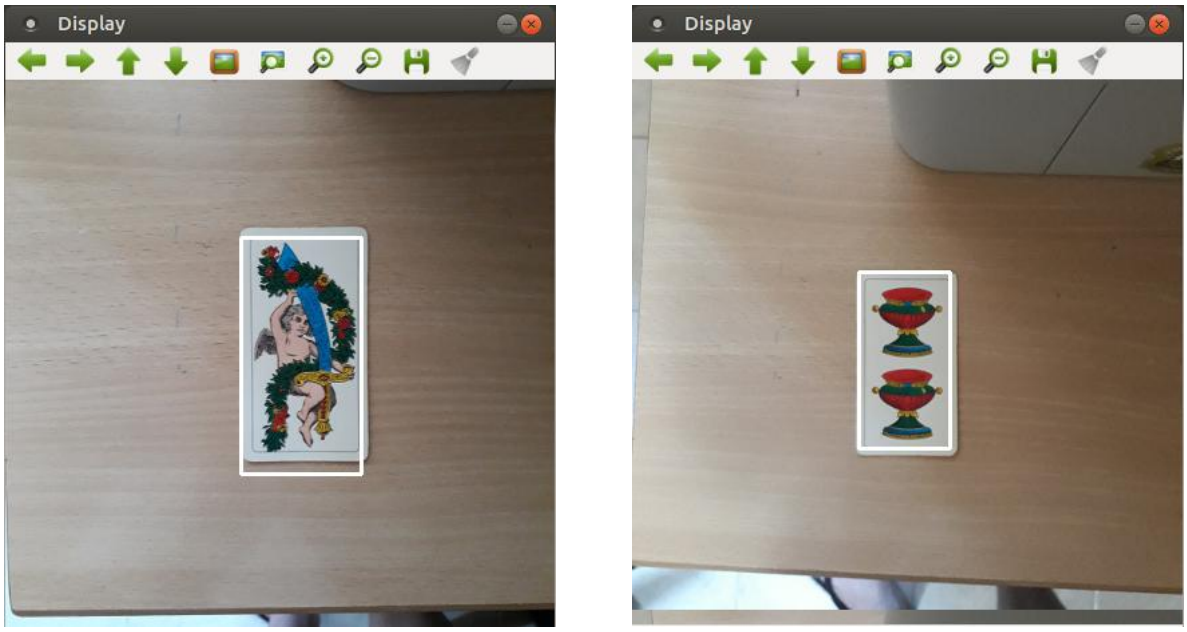
Un esempio di Cascade che non funziona a dovere



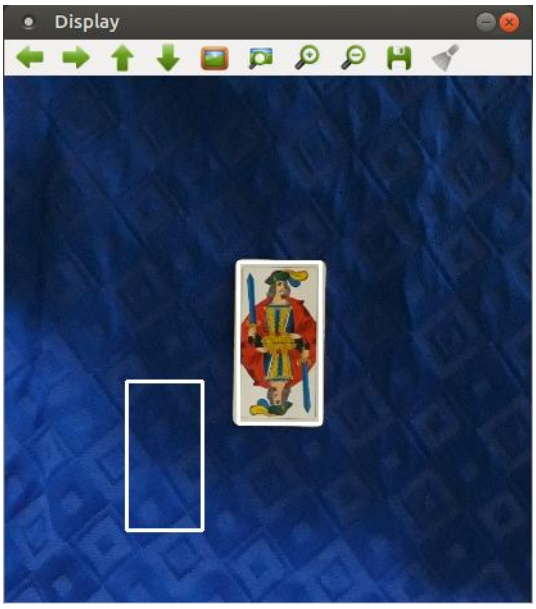
Il Cascade più prestante tra quelli generati

Il parametro che nel nostro caso ha provocato le ripercussioni più notevoli sulla buona riuscita di un Cascade è proprio quello che si occupa della dimensione dei rettangoli, o alternativamente della finestra di individuazione: più si sono aumentate le sue dimensioni, migliori sono stati i risultati, il che ha senso dato che è possibile formare più combinazioni di features Haar al suo interno. Tuttavia dimensioni eccessive comportano tempi di training più lunghi, per non parlare dello stress che viene inflitto sulla CPU e sulla GPU del computer a causa della mole di lavoro richiesta. Partendo dalla dimensione di default 20x20, si è arrivati quindi a quella ideale di 20x35: ogni altro tentativo di effettuare un training con parametri più alti ha comportato il blocco totale del calcolatore, le cui risorse erano già messe a dura prova nel tentativo di far girare la macchina virtuale con

l'ambiento di lavoro al suo interno.
Ovviamente, dato che le foto positive sono state generate artificialmente partendo dalla stessa immagine sample, il Cascade che è stato infine selezionato è meno efficace rispetto ad un Cascade che sarebbe stato allenato usando foto "tradizionali".
Un'altra cosa da notare è che ogni tipo di carta viene individuato, non solo il 6 di denari che abbiamo usato come sample, forse un effetto collaterale della riflessione precedente o di un numero relativamente basso di carte positive e negative fornite come input rispetto ad un Cascade professionale.
Si pone dunque il problema di riconoscere il numero e il seme di una carta appena individuata, un problema che sarà trattato successivamente nel paragrafo 4.3.

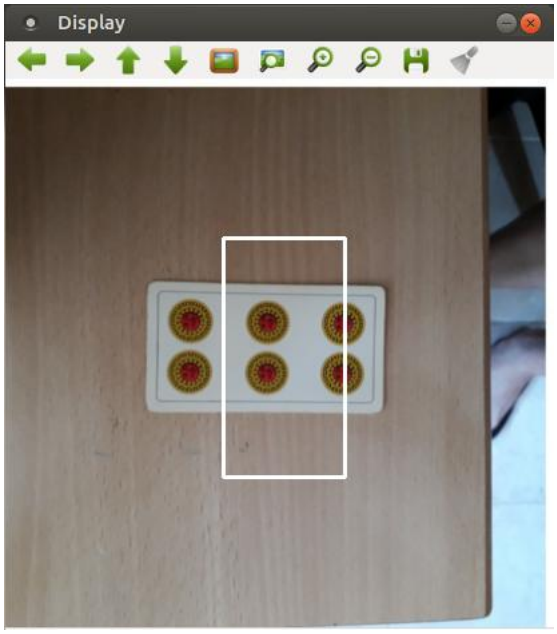


Esempi di carte di seme e numero diversi dalla foto sample che tuttavia vengono riconosciute lo stesso



Esempio di carta individuata correttamente e falso positivo nella stessa foto

Anche se tutti gli esempi finora mostrano carte posizionate in modo dritto, il Cascade è in grado di cogliere anche quelle di profilo: tuttavia i rettangoli che le delimitano, proporzionali alle dimensioni della finestra di individuazione usata nel training, non riescono a coprire le carte nella loro interezza, un fenomeno previsto dato che quelle dimensioni scelte durante l'allenamento sono state pensate proprio per le carte dritte.



Esempio di individuazione di una carta di profilo

Va inoltre notato che il settaggio di un Cascade nel momento in cui si deve utilizzarlo su una foto ha enormi conseguenze sulla qualità dell'individuazione dell'oggetto. Scegliendo dei parametri sbagliati, infatti, si rischia di rendere inutile anche il migliore dei Cascades allenati. Tuttavia, dato che la selezione di questi parametri allo scopo di ottenere la miglior finestra di individuazione possibile è collegata strettamente al modo con cui si è deciso di risolvere il problema della recognition, saranno anch'essi analizzati successivamente nel paragrafo 4.3.
In conclusione, il Cascade utilizzato dal programma è stato generato con i seguenti parametri:

Numero di foto positive	850
Numero di foto negative	1900
Dimensioni finestra individuazione	20x35
Numero stages di training	10*
Tipo di Features	Haar
Tipo di boost**	GAB
Hit-rate minimo	0.995

Il tempo di training ha richiesto circa 25 minuti.
*: Sono stati eseguiti solo tre livelli, in quanto l'hit rate richiesto è stato soddisfatto già dal terzo livello
**: Il boosting è un processo che comporta la creazione di un Cascade come somma lineare di una serie di Classifiers definiti "weak", ovvero poco efficaci, ma più semplici, e che quindi richiedono meno prestazioni da parte del PC e tempi di training inferiori[19, 20].

4. Realizzazione del gioco

Concluse le fasi preliminari con la selezione del Cascade Classifier più efficiente, si è finalmente passati alla creazione del gioco stesso. Ovviamente il primo step è stata la realizzazione di classi in grado di astrarre i concetti alla base della Scopa.

4.1 Le classi

Essendo un gioco di carte, è scontato che due delle classi siano proprio quelle di **Carta** e **Mazzo**.
La prima ha come variabili globali tutto ciò che c'è da sapere a proposito di una data carta, ovvero il numero, il seme e la directory della foto in cui è contenuta. Le ultime due variabili globali sono fotoZoom e fotoRettangolo, che contengono, rispettivamente, un'immagine zoomata della carta estratta dalla foto originale e un'immagine in cui la carta è circondata da un rettangolo blu, con all'interno numero e seme scritti in rosso; nel capitolo 4 sarà detto dove queste due foto sono utilizzate. I costruttori di questi oggetti accettano solo le directory: successivamente, quando le foto a cui puntano sono analizzate, e le carte al loro interno individuate e riconosciute, vengono aggiornati anche gli altri parametri, compresi numero e seme.
Tutti i metodi di **Carta** sono quelli standard per accedere o alterare i suoi attributi, ad eccezione del metodo riconosci() che, come suggerisce il nome, si occupa della detection e della recognition della carta.
La classe **Mazzo**, invece, utilizza uno struct **node**, dichiarato al suo interno, per tenere traccia degli oggetti **Carta** che gli appartengono: la lista è di tipo lineare, in cui un elemento punta al successivo, ma non al precedente. Le sue variabili globali sono quindi due puntatori **node**, uno che punta alla prima carta del mazzo, l'altro all'ultima. I suoi metodi contengono operazioni che alterano la lista (aggiungere e togliere carte, mischiare, unire due mazzi etc.), che danno informazioni precise sulle carte al suo interno (come il numero di denari o il valore della primiera) e che hanno a che fare con la realizzazione della GUI grafica stessa.
Inizialmente queste dovevano essere le uniche classi, ma poi si è deciso di implementare anche **Giocatore**, esteso da **Utente** e **CPU**. Queste contengono informazioni che prima venivano conservate nel main del programma, quali carte pescate, carte in mano, punteggi, numero di scope e così via: tuttavia la mole di parametri che doveva essere passata ai metodi ausiliari si era fatta insostenibile: da qui la decisione di creare classi apposite per facilitare il passaggio di tali parametri.
Utente conserva il nome che il giocatore umano decide di inserire a inizio gioco, mentre **CPU** ha il parametro coperto, che stabilisce se deve giocare a carte coperte o scoperte.
I loro metodi sono gli standard set and get.

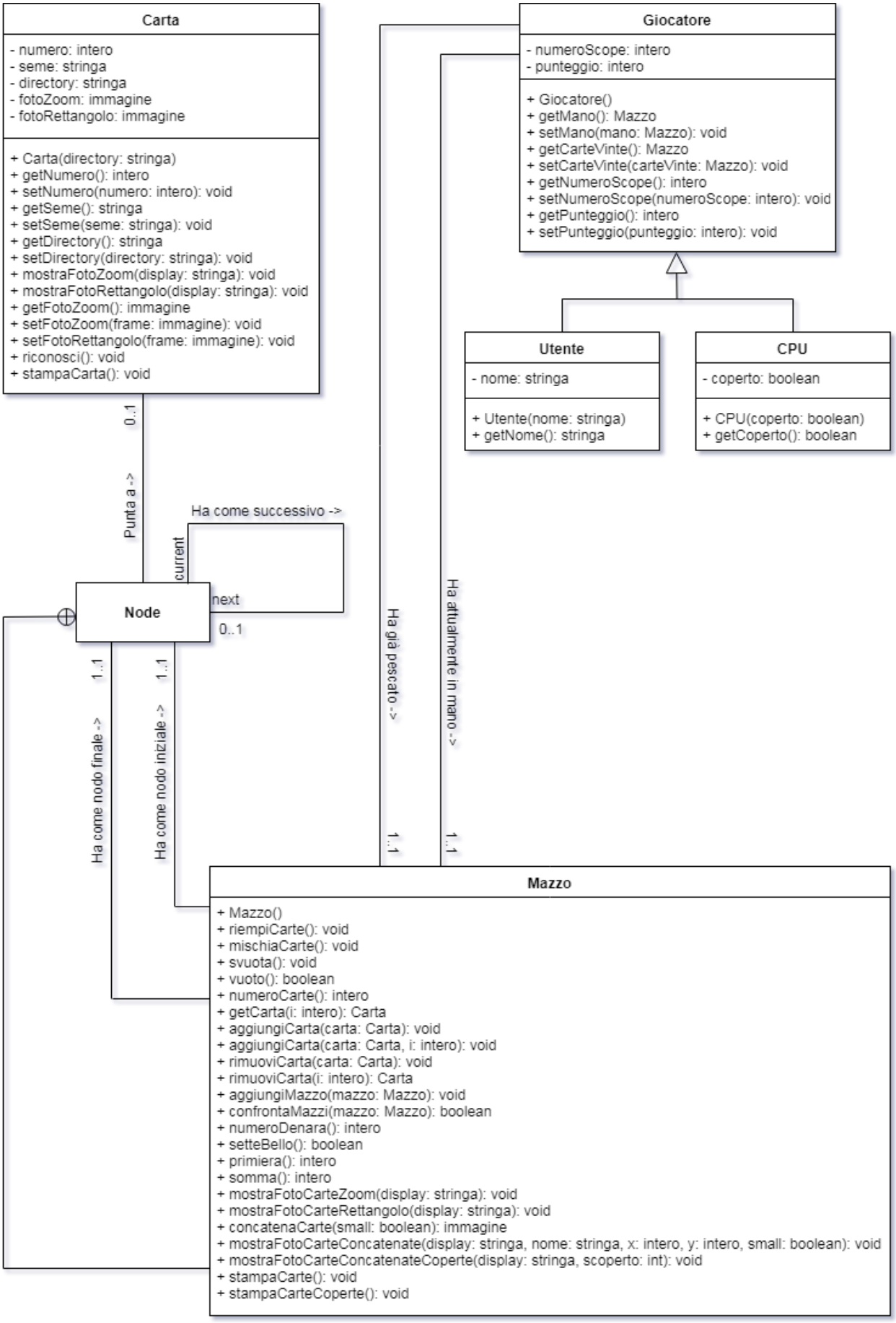


Diagramma UML delle classi

4.2 La logica di gioco

Si può ora osservare la struttura del gioco stesso, contenuta nel main del file Scopa.cpp: dopo che l'utente ha selezionato il pulsante "Play" sul menù principale (che sarà analizzato nel paragrafo 5.1), la prima cosa che deve fare è inserire un nome con il quale verrà identificato per il resto della partita.
In seguito c'è l'inizializzazione dei vari parametri, tra cui la creazione degli oggetti Utente e CPU, che a loro interno contengono molte informazioni, come le carte in mano, quelle pescate e i loro punteggi, e di un Mazzo vuoto che rappresenta le carte sul tavolo.
Finiti i preparativi, ha inizio il gioco vero e proprio, dove giocatore e CPU giocheranno una serie di partite fino a quando nessuno dei due raggiunge un punteggio superiore a 11 e a quello dell'avversario.
All'inizio di ogni partita c'è la creazione di un Mazzo di 40 oggetti Carta, le cui directory possono puntare alle foto della cartella CarteA, CarteB o CarteC. La scelta è completamente casuale e varia da partita a partita.
Una volta mischiate, le carte all'interno del Mazzo sono pronte ad essere assegnate: vengono quindi poste 4 carte sul tavolo, mentre al giocatore e alla CPU viene data una mano di 3 carte ciascuno; è proprio nel momento in cui una Carta viene sottratta dal Mazzo per essere utilizzata nel gioco che avviene la detection e la recognition della carta all'interno della foto.
Durante le prime fasi di progettazione, il riconoscimento veniva applicato sull'intero Mazzo di 40 carte istanti dopo la sua creazione: il lato positivo è che mentre ogni problema sarebbe stato catturato prima dell'inizio del gioco stesso, dall'altra parte era un processo che impiegava molto tempo e che dunque era stancante da subire ad ogni singola partita. Da qui la decisione di applicare la recognition sul momento, quando è strettamente necessaria, su un numero ridotto di carte, ma in maniera più frequente: ovviamente, per compensare il fatto che errori imprevisti possano interrompere il gioco nel suo pieno svolgimento, si è deciso di offrire al giocatore un'opzione nel menù principale che si occupa di testare anticipatamente l'integrità delle foto, in modo che ogni eventuale problema venga segnalato prima dell'inizio del gioco stesso, evitando brutte sorprese.
Nel prossimo paragrafo saranno analizzati quali sono i tipi di problemi che potrebbero verificarsi nella fase di recogniton.
Una volta assegnate correttamente le carte (il programma termina in caso di eccezioni), il giocatore ha modo di visionare la situazione sulla GUI grafica o sul terminale dei comandi, che contiene un log di gioco dove vengono trascritte tutte le azioni che sono state compiute dall'inizio delle partita.
Le carte della CPU saranno coperte o scoperte a seconda della modalità selezionata nel menù.

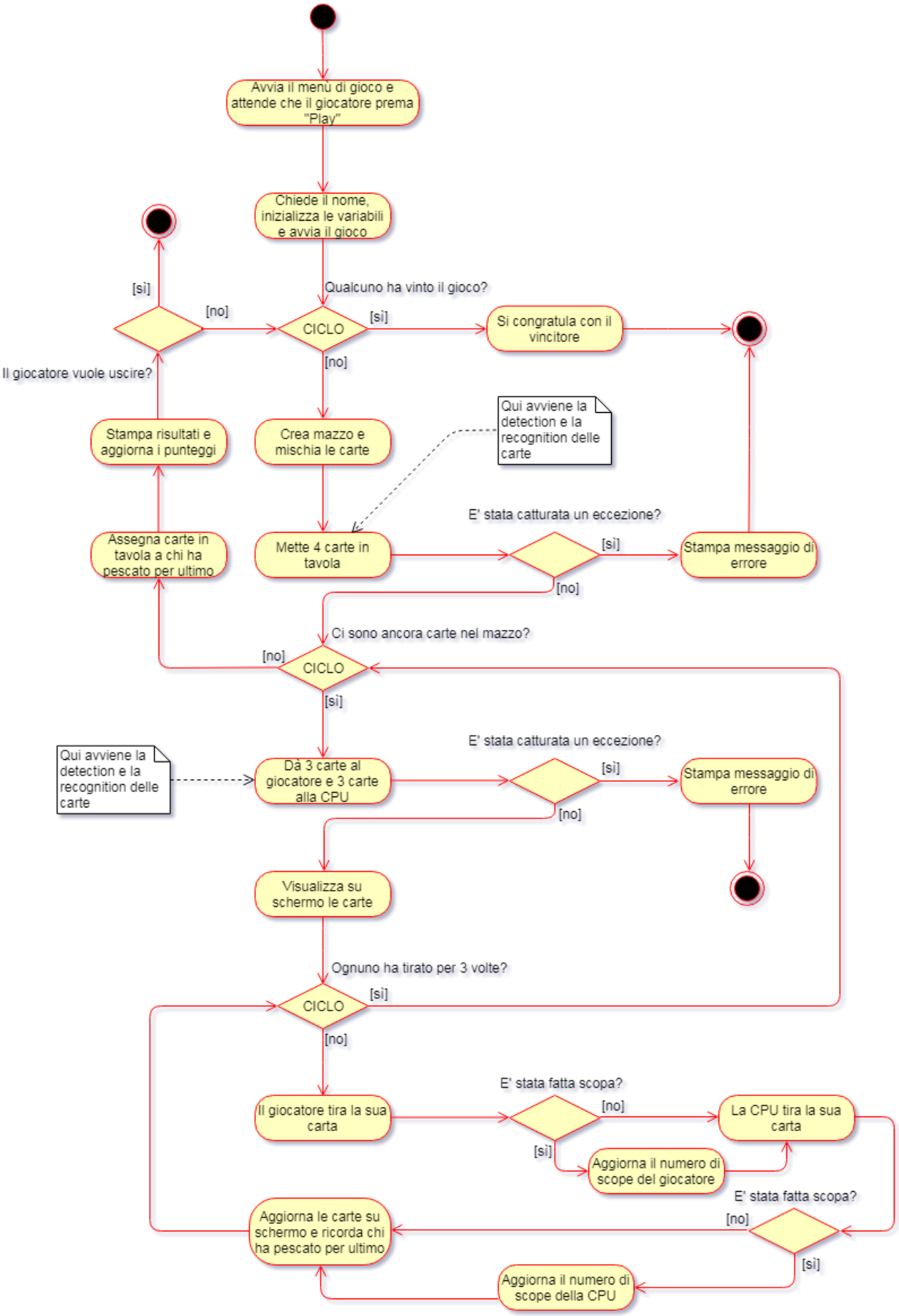


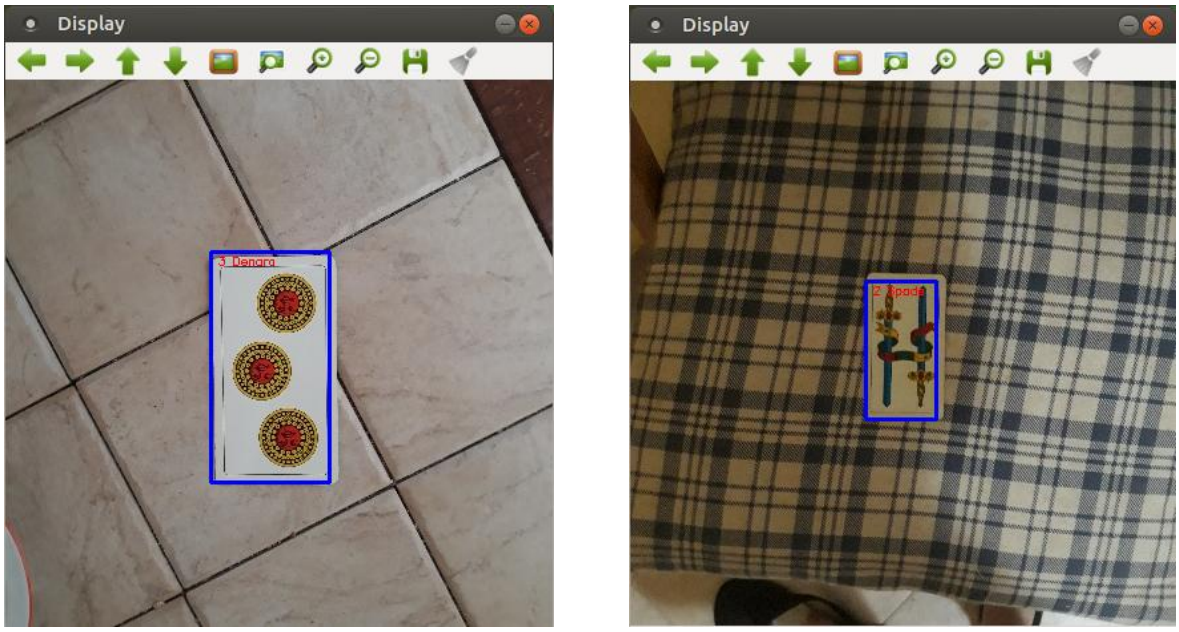
Diagramma attività del **main** in **Scopa.cpp**

A questo punto il giocatore (se inizia per primo) può fare la sua mossa toccando con il pulsante sinistro del mouse la carta che vuole tirare: le conseguenze del suo tiro sono immediatamente mostrate nella GUI grafica, con l'aggiornamento delle carte sul tavolo e nella sua mano, e nel log, in forma scritta.
Alternativamente, il giocatore può cliccare con il pulsante destro una qualsiasi carta in gioco, comprese quelle sul tavolo e della CPU (solo se sta giocando a carte scoperte), per visualizzare brevemente la foto data in input in cui compare la carta appena selezionata, con la differenza che la carta stessa è circondata da un rettangolo blu, in cui all'interno sono stati scritti in rosso numero e seme.
Dopo aver tirato, è il turno della CPU: le sue strategie saranno analizzate in seguito nel paragrafo 4.4.
Ad ogni tiro vengono salvate tutte le informazioni del caso, ovvero le carte pescate e se si è fatta Scopa.
Giocatore e CPU continuano a tirare fino a quando entrambi non finiscono le carte nella loro mano: quando ciò accade, vengono date loro altre 3 carte dal mazzo (dopo che sono state sottoposte ad un processo di recognition, ovviamente), sempre se quest'ultimo non è ancora vuoto.
In caso contrario la partita finisce, le carte sul tavolo vengono assegnate a chi ha pescato per ultimo, e sul log di gioco vengono mostrati i risultati e i punteggi ottenuti da ciascun giocatore, mentre sulla GUI vengono mostrate le carte che hanno pescato: come da regolamento, i punteggi vengono assegnati in base al numero di carte pescate, denari, Scope, Sette Bello e primiera.
Al giocatore viene quindi data la possibilità di uscire o di continuare a giocare: nel caso della seconda opzione, viene dunque iniziata una nuova partita, sempre se nessuno dei due partecipanti ha totalizzato un punteggio sufficiente da vincere il gioco.
Un paio di aspetti che sono stati omessi dal diagramma UML per semplificare lo schema sono il fatto che l'ordine di tiro tra giocatore e CPU si scambia ad ogni partita, e che le eccezioni sono catturate in più punti del programma. Tuttavia, essendo quest'ultime legate alla realizzazione della GUI, si è deciso di toglierle dallo schema per distinguerle da quelle lanciate durante le fasi di detection e recognition.

4.3 AI: detection e recognition delle carte

Come era già stato notato nel paragrafo 3.3, una delle problematiche maggiori nello sviluppo del progetto è stata come garantire il riconoscimento delle carte trovate dopo un processo di object detection, che, si ricorda, nulla dice riguardo al valore e il seme delle carte individuate.
Per classificarle correttamente, dunque, si è deciso di effettuare un confronto tra le features selezionate dal Cascade durante una detection sulla foto indicata dalla variabile globale

directory di un oggetto Carta e un particolare tipo di immagini, chiamate Target, che sono di piccole dimensioni (37x70) e che sono interamente costituite dal close-up di una determinata carta.
Prima di effettuare il confronto, però, viene eseguita un'equalizzazione dell'istogramma sulle features (ridimensionate a 37x70) e sulle foto Target, entrambe aperte in bianco e nero, in modo da calibrare il contrasto e accentuare le loro caratteristiche salienti[21, 22]. In seguito, viene calcolata la distanza euclidea tra le tutte le coppie possibili di features e foto Target[23, 24], tenendo sempre traccia del confronto che ha generato la distanza più piccola: terminato il processo, si usa quella distanza minima per predire il seme e il numero dell'oggetto Carta, e aggiornare le variabili fotoZoom e fotoRettangolo.



Esempi di recognition corrette

Questa è la teoria dietro alla nostra soluzione; ora si elenca una serie di considerazioni scaturite durante l'implementazione pratica della stessa:
- Il programma, di default, contiene 40 foto Target, ognuna rappresentante una carta differente. Per ottenerle si è deciso di scannerizzare un mazzo di carte piacentine, per poi estrarre, ridimensionare e salvare ogni carta a parte grazie ad un programma di fotoritocco. Questo significa che solo quel tipo di carte viene supportato durante la recognition, a meno che le foto Target non siano sostituite con foto che rappresentano un altro tipo di carte italiane. Questa operazione non richiederebbe nessun alterazione nel codice del programma.

Esempi di foto Target

- Dato che questa soluzione si basa sul confronto tra due immagini, è preferibile, se non necessario, che il Cascade sia settato in modo da cogliere le features il più precisamente possibile, così che al loro interno siano contenute solo ed esclusivamente le carte da analizzare, senza piccoli pezzi di scenario che potrebbero compromettere il calcolo della distanza euclidea. Durante il paragrafo 3.3 si era infatti anticipato che come si setta un Cascade nel momento del suo utilizzo è altrettanto importante a come si decide di allenarlo. I parametri che possono essere alterati, infatti, includono il fattore con cui l'immagine è ridimensionata dopo che la finestra di individuazione è stata passata su tutta la foto, il numero minimo di neighbors, che essenzialmente aumenta la qualità delle rilevazioni a discapito della loro quantità più lo si alza, e la dimensione degli oggetti che si vuole trovare[6]. Quest'ultimo è di gran lunga il parametro più importante, in quanto ha direttamente a che fare con il contenuto della foto che finisce all'interno di una feature. Per creare delle features più precise, si è deciso dunque di effettuare la object detection due volte a foto, modificando leggermente la dimensione degli oggetti richiesti tra una rilevazione e l'altra. Le dimensioni scelte sono 60x50 e 30x40. Per compensare l'inevitabile incremento di tempo speso durante la detection, l'immagine viene automaticamente ridotta prima del processo, in modo da garantire tempi sempre relativamente brevi.
- Purtroppo questa strategia penalizza le carte che non vengono posizionate in modo dritto, poichè le features eventualmente rilevate conterrebbero pezzi di scenario a causa delle proporzioni sfavorevoli della finestra di individuazione.

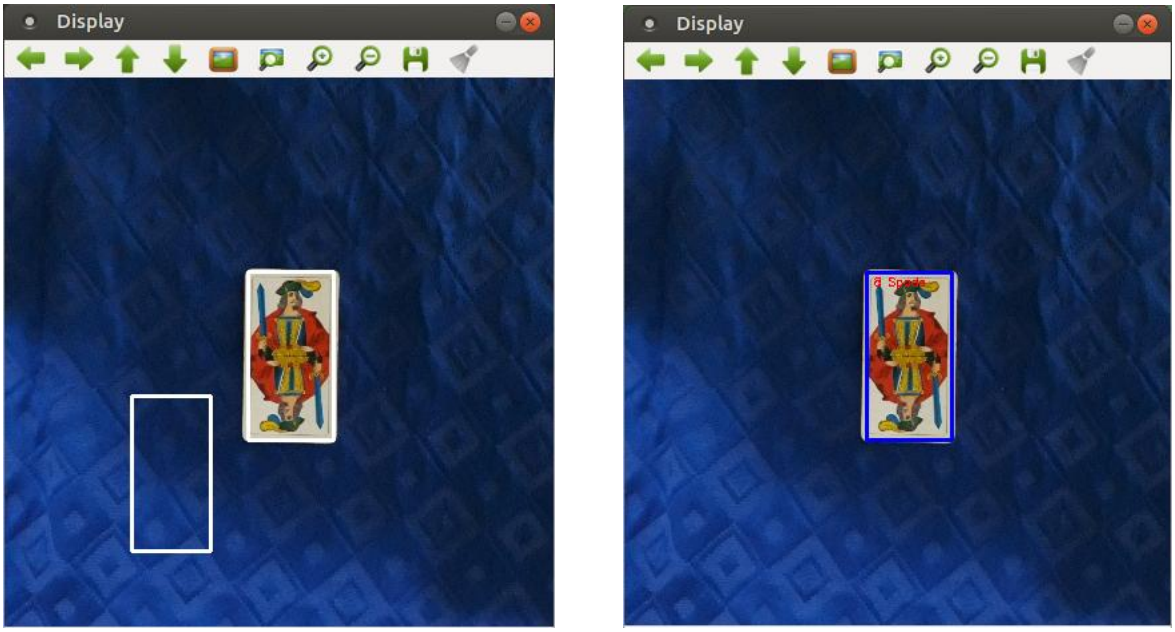
Esempio di recognition fallita su una carta laterale

- Il programma capisce il numero e il seme di una foto Target grazie al nome della sua directory, che deve essere di tipo "x.jpg", dove 'x' è un numero che va da 1 a 40: questo significa che è possibile insegnare informazioni sbagliate se i nomi dei files non riflettono il

contenuto delle foto Target. L'ordine stabilito è:

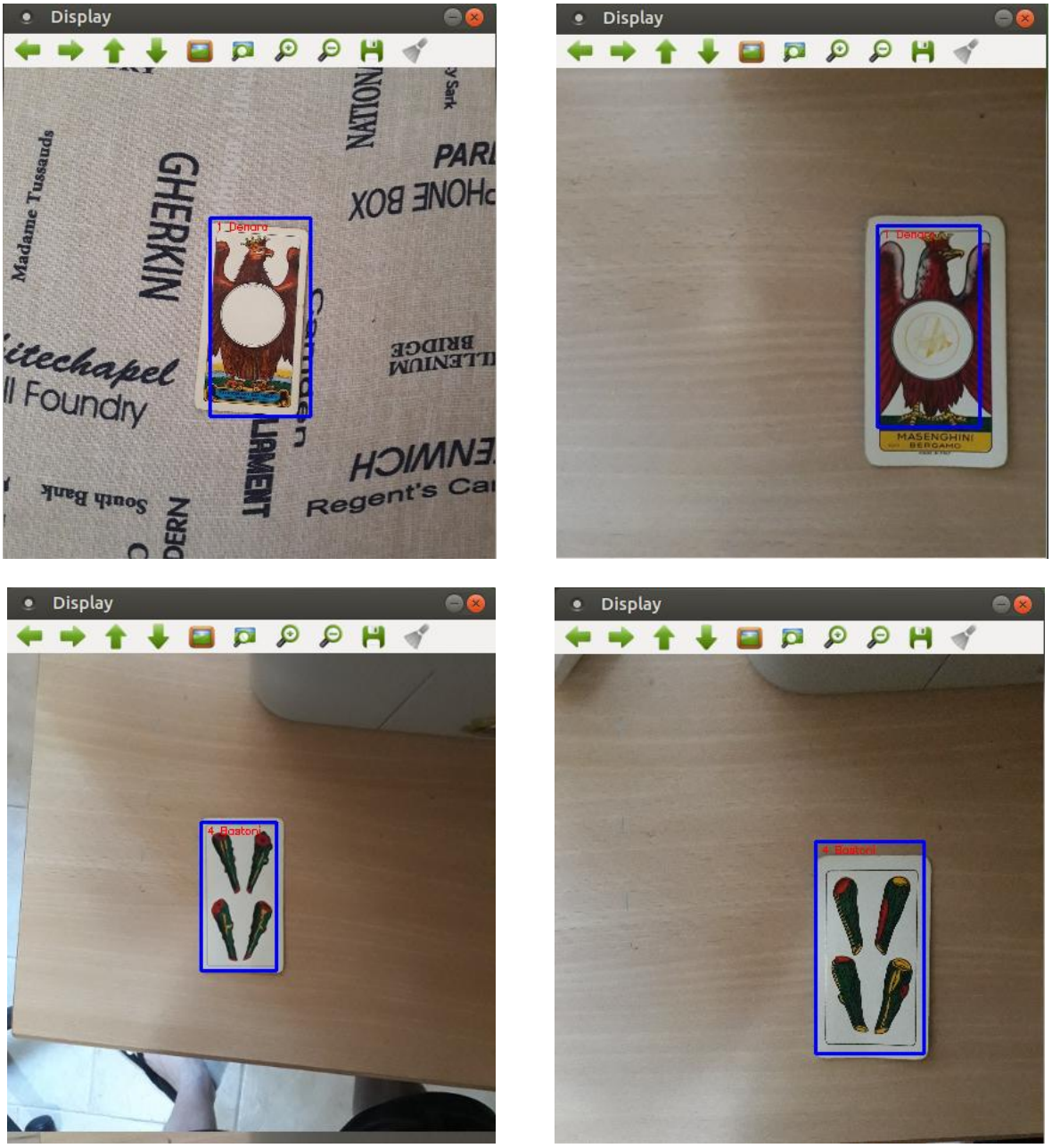
Valore x	Numero	Seme
1 <= x <= 10	x	Denara
11 <= x <= 20	x-10	Spade
21 <= x <= 30	x-20	Coppe
31 <= x <= 40	x-30	Bastoni

- Il programma opera sulla base che ad ogni oggetto Carta è associata una foto differente: nel caso quindi dovessero apparire, ad esempio, due carte in una data immagine, anche se queste verrebbero individuate entrambe durante la fase di detection, nella recognition vincerebbe solo una coppia feature-Target dalla distanza minima, quindi la Carta verrebbe aggiornata con i parametri della carta appartenente alla feature vincitrice, ignorando completamente l'altra.
- In caso di features contenenti falsi positivi, o carte di un tipo differente da quelle Target, si hanno due possibili risultati: se quella feature è l'unica ad essere stata rilevata, allora all'oggetto Carta saranno ugualmente assegnati un numero e un seme, molto probabilmente errati, in quanto esiste sempre una coppia feature-Target dalla distanza minima, anche se alta. Una possibile soluzione sarebbe scartare tutte quelle distanze che superano una certa soglia, ma poi si penalizzerebbero quelle features che contengono carte valide, ma che sono sfocate, e che quindi vengono identificate correttamente anche con distanze minime relativamente alte. Se invece sono state rilevate più features, e almeno una di queste contiene una carta valida, allora il problema non si pone, poiché vincerebbe una coppia feature-Target relativa a quella carta.



Esempio di recognition corretta, nonostante la presenza di un falso positivo

- Va tuttavia notato che una feature contenente una carta di tipo diverso da quelle Target non comporta necessariamente una recognition errata: dopotutto si parla di confronti tra immagini di 37x70. A quelle dimensioni non contano tanto i dettagli precisi, ma piuttosto la forma generale dell'immagine, vista nel suo insieme. Se quindi quella carta appartiene ad un mazzo di carte italiane ragionevolmente simile a quello delle foto Target, la recognition potrebbe avvenire correttamente lo stesso.



Esempi di recognition corrette tra carte uguali di mazzi differenti

- La recognition termina con il lancio di un eccezione se non dovesse essere trovata nessuna feature, sia perché non c'è nessuna carta nella foto o sia perché si tratta di falso negativo, o se dovesse verificarsi un problema durante l'apertura dei files.

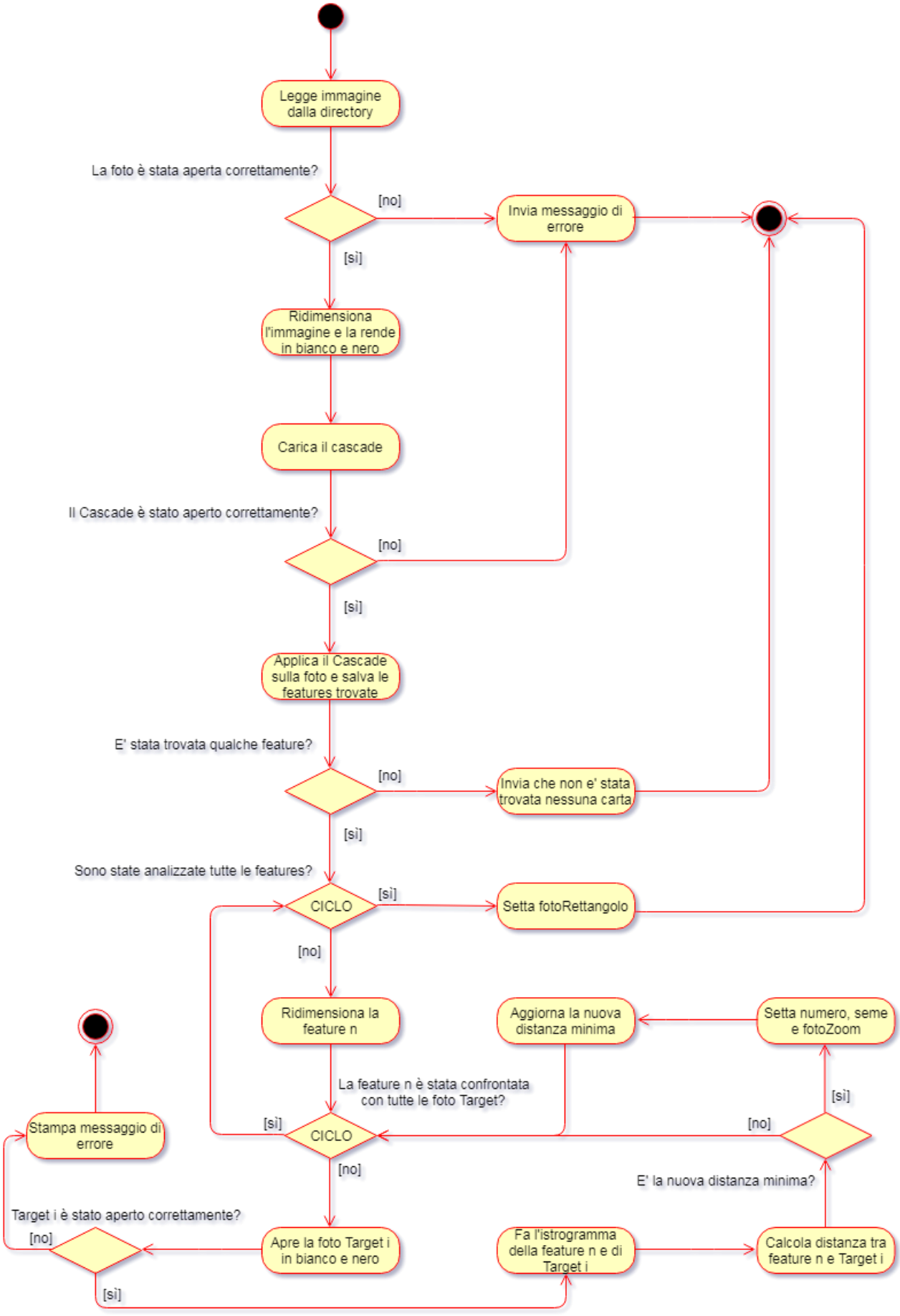


Diagramma attività della funzione **riconosci** in **Carta.cpp**

4.4 AI: le tattiche della CPU

L'altra più grande problematica del progetto è stata come realizzare una CPU avversaria che fosse in grado di tenere testa al giocatore: Scopa, come altri giochi a carte, è basato in larga parte sulla fortuna, ma questo non vuol dire che non vadano applicate delle strategie precise per aumentare le proprie possibilità di vittoria, come cercare di pescare più denari possibili o carte dal numero 6 e 7 per realizzare la primiera più alta. Strategie che, dunque, vanno insegnate alla CPU.

Quando arriva il suo turno, la CPU scorre tutte le carte nella sua mano alla ricerca di quelle che potrebbero fargli pescare qualcosa dal tavolo.

Dunque, per ogni carta in mano "i":

- la CPU, come da regolamento, controlla prima se sia possibile pescare carte singole dallo stesso numero di "i", altrimenti cerca quelle combinazioni di carte la cui somma è la stessa di "i". Per far ciò il programma usa una funzione ricorsiva che estrae via via le carte dal tavolo per inserirle in un mazzo provvisorio, per poi calcolarne la somma e confrontarla al numero di "i". Eventualmente questo mazzo provvisorio conterrà ogni singola permutazione delle carte in tavola, in modo da coprire tutte le somme possibili, mentre il tavolo sarà ripristinato alla sua forma originale. Infine, al termine dei confronti, tutte le possibili prese vengono salvate in una lista, a prescindere che siano composte da una o più carte.
- Scorrendo la lista, la CPU analizza le possibili pescate, assegnando a ciascuna un punteggio che riflette il suo contenuto: tra i fattori considerati all'interno di una possibile presa ci sono il numero di carte, il numero di denari, quanto quelle carte sono utili per la primiera, se è possibile fare Scopa e la presenza del 7 Bello. Va anche notato che il punteggio viene assegnato tenendo traccia del progresso della partita: se quindi, ad esempio, il giocatore ha già pescato 6 carte di denari, allora le carte di quel genere perderanno di valore e saranno trattate come tutte le altre, in quanto la CPU ha già perso il punto per il numero di denari. Ogni punteggio viene confrontato con quello più alto: se il record viene battuto, la CPU individua una nuova migliore possibile presa e aggiorna il punteggio più alto.

Dopo aver analizzato tutte le carte in mano, la CPU ha due alternative:

- Se è possibile pescare qualcosa, allora la CPU tira quella carta corrispondente alla possibile presa migliore, e pesca quelle carte, terminando il suo turno.
- Altrimenti, dato che non può pescare nulla, tira la carta che ha meno valore strategico. Per far ciò la CPU assegna un punteggio ad ogni carta della sua mano, usando un sistema di valutazione differente rispetto a quello delle possibili pescate. Generalmente si preferisce che la carta lanciata sia di numero basso, in modo da conservare quelle più alte

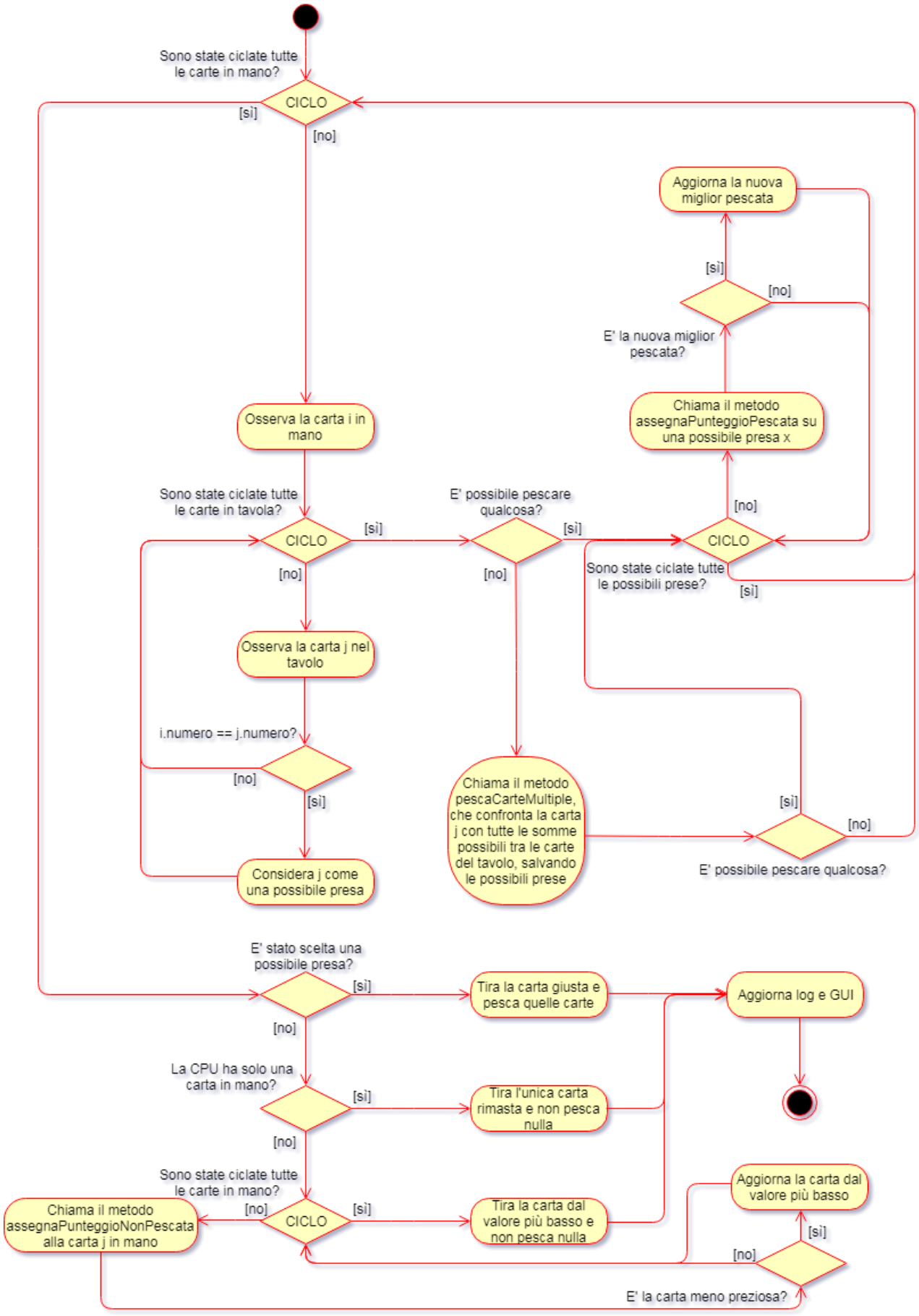


Diagramma attività di **mossaCPU** in **ScopaUtil.cpp**

per i turni successivi, nella speranza che per allora ci sarà qualcosa di pescabile. Un nuovo fattore che viene considerato è l'evenienza che la somma delle carte nel tavolo sia inferiore a 10: la CPU, se possibile, cercherà di lanciare una carta che, una volta sommata a quelle del tavolo, renderà quella somma maggiore di 10, impedendo al giocatore ogni possibilità di fare Scopa il turno seguente. Al termine del processo, la CPU tira la carta meno preziosa e conclude il suo turno. Ovviamente se possiede solo una carta in mano, tira direttamente quella, senza stare a calcolare i punteggi.

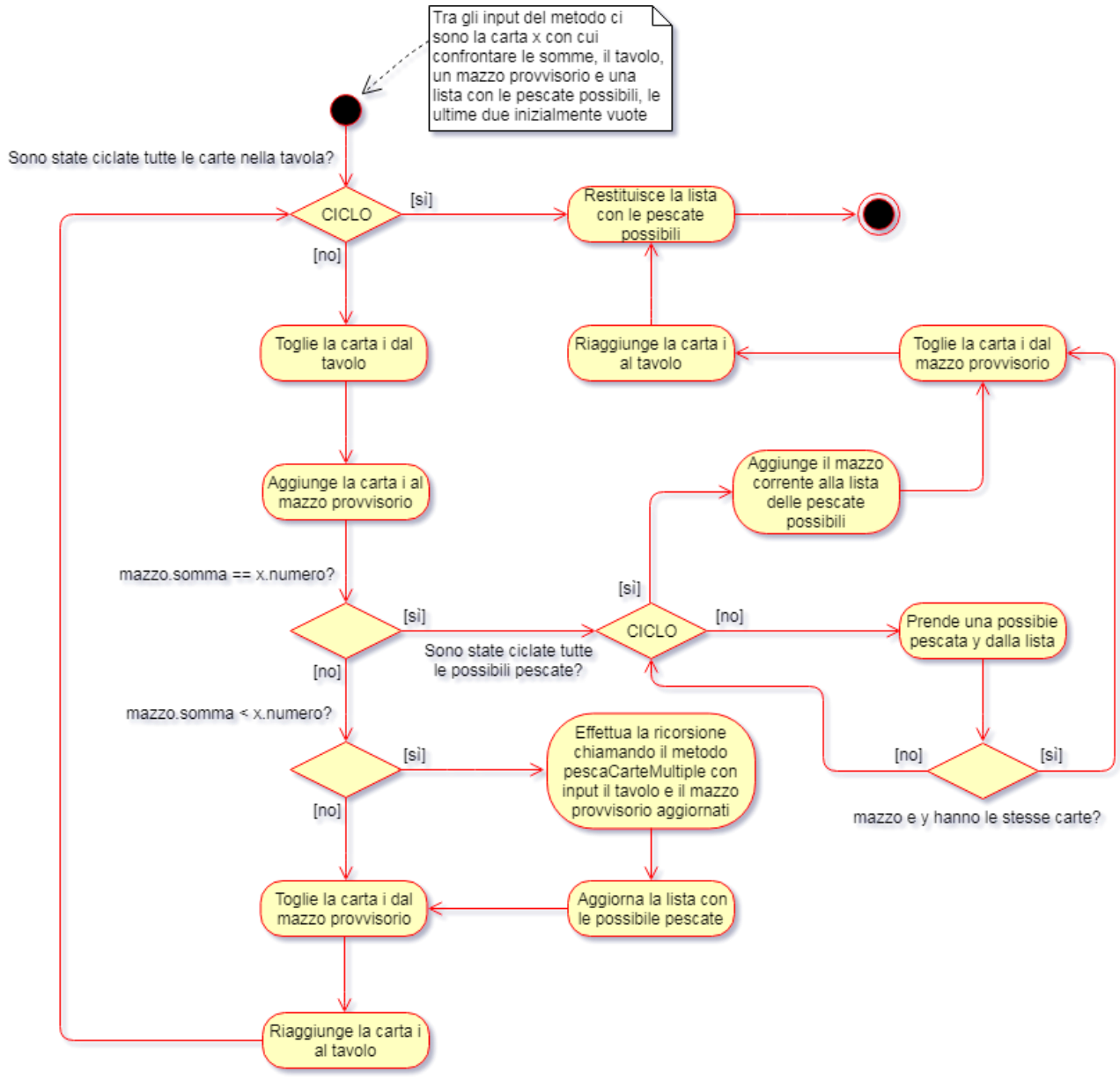
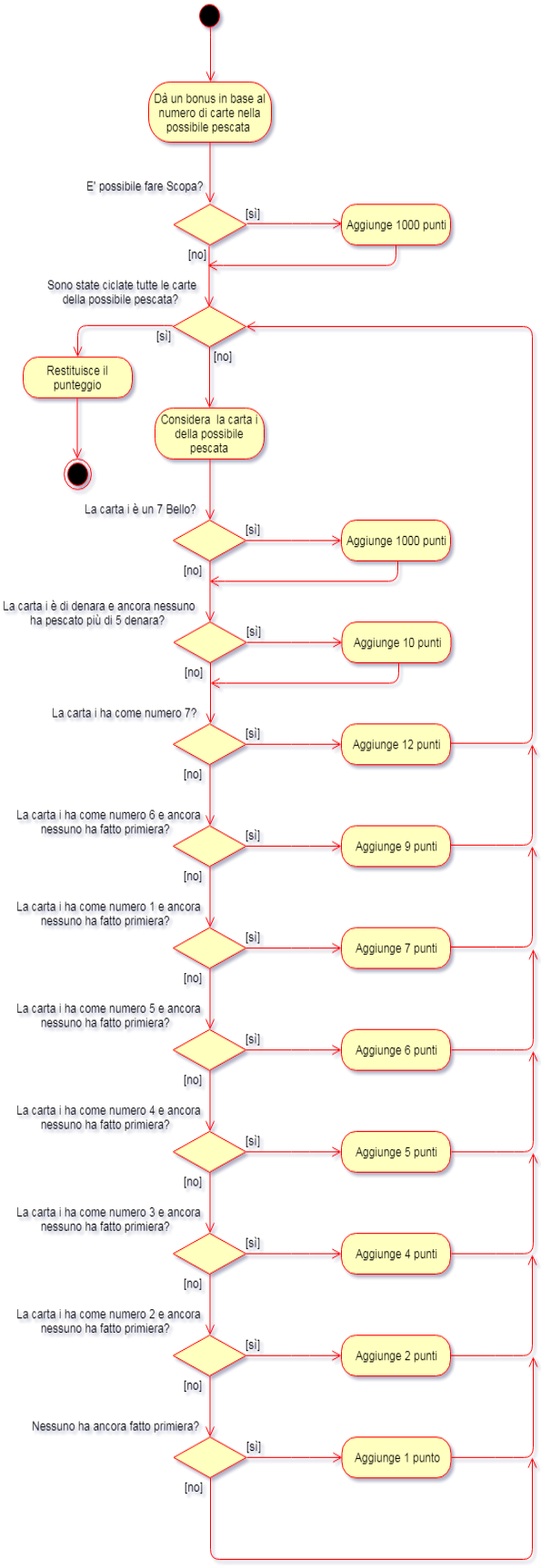
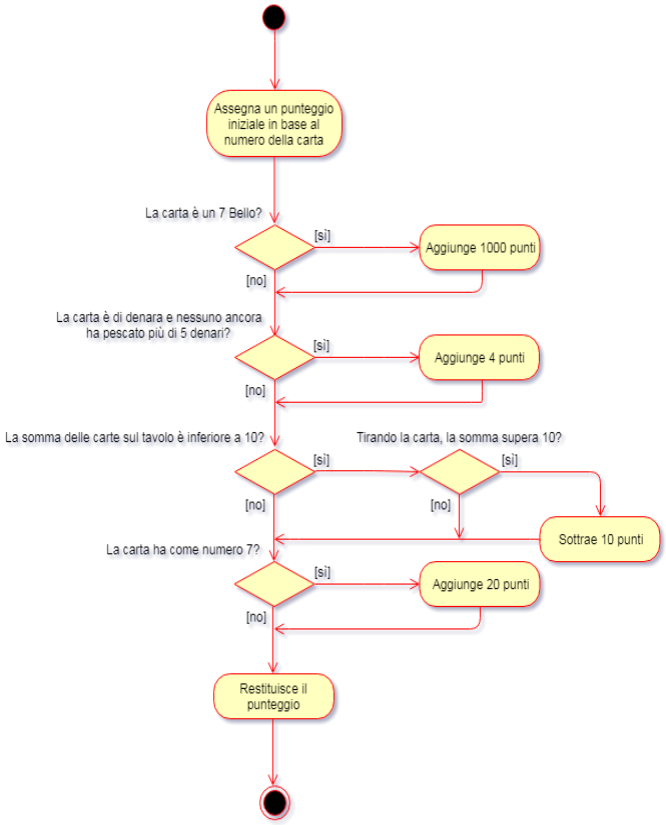


Diagramma attività di **pescaCarteMultiple** in **ScopaUtil.cpp**





Diagrammi attività di **assegnaPunteggioPescata** e **assegnaPunteggioNonPescata** in **ScopaUtil.cpp**

4.5 Assegnazione e integrità delle carte

Nel paragrafo 4.2 si era anticipato che il riconoscimento delle carte avviene nel momento in cui vengono sottratte dal mazzo di gioco per essere assegnate al tavolo o a uno dei due giocatori, mentre nel paragrafo 4.3 si sono descritte le eccezioni che il programma lancia nel caso le foto inserite dall'utente non contengano carte o siano soggette a casi di falso negativo. Esiste tuttavia un'ultima problematica da risolvere: dato che le foto sono scelte dall'utente, si deve garantire che non solo le carte siano presenti, ma che siano tutte diverse, in modo che durante la partita non escano dei doppioni. Per evitare questa evenienza, il programma tiene traccia delle carte già uscite grazie ad un array di boolean di 40 elementi, dove ogni casella contiene un flag che indica se la carta assegnatole è già uscita o no.

Nel caso quindi una recognition su una carta appena estratta dal mazzo si concluda con successo, si usa il suo ID (ottenuto in base al suo valore e seme) per accedere alla sua specifica casella e attivare il flag se è la prima volta che quella carta esce, o lanciare un'eccezione se il flag era già attivo.

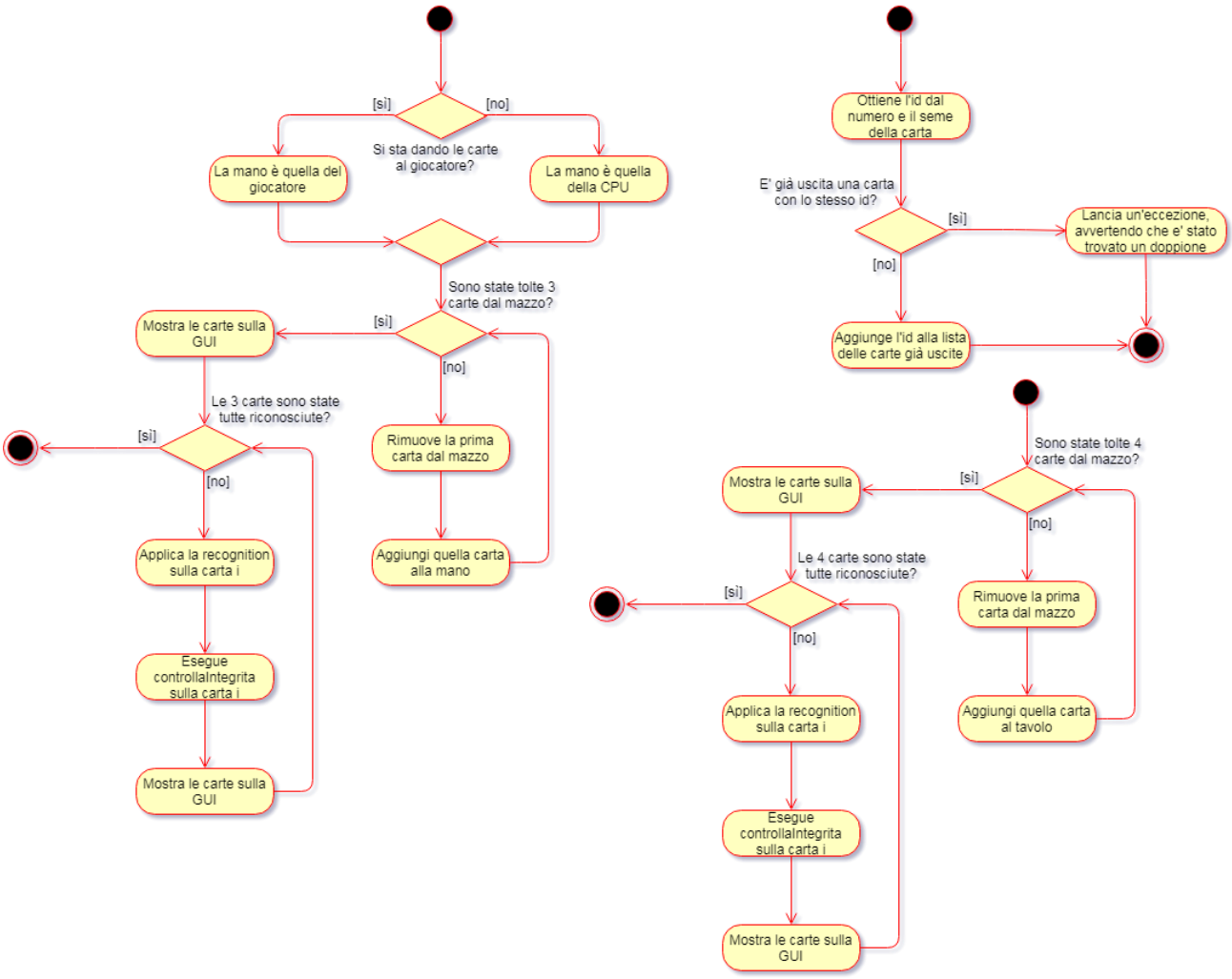


Diagramma attività dei metodi **daiCarteMano**, **daiCarteTavolo** e **controllaIntegrita** in **ScopaUtil.cpp**

Si può inoltre notare dagli schemi UML che le carte vengono stampate sulla GUI ancor prima che vengano riconosciute: questo perché quando un oggetto Carta viene creato inizialmente con la sola directory, gli viene assegnata una foto provvisoria che mostra una carta interamente coperta da un effetto pixellato che cela il suo contenuto, che, si ricorda, non è ancora noto fino a quando non avviene la recognition.
Mostrando dunque un'intera mano (o tavolata) di carte, inizialmente occultata, per poi rivelarla elemento dopo elemento, si crea una specie di animazione che mostra i progressi della recognition.
Si può considerare questo passaggio una breve anticipazione sulla creazione della GUI, trattata nel prossimo paragrafo.

Esempio di progressione di una recognition

Si ricorda inoltre che per evitare la presenza improvvisa di un doppione durante la partita, è sempre possibile effettuare un test sull'integrità delle foto direttamente dal menù principale.

5. La GUI interattiva

Uno degli obiettivi alla base del progetto è sempre stata la creazione di una GUI interattiva, una scelta ovvia dato che si ha a che fare con un gioco di carte, dove l'aspetto visivo è fondamentale. Tuttavia, come si è notato nel paragrafo introduttivo, mentre OpenCV offre degli strumenti che permettono la creazione di un'interfaccia grafica, questi sono assai limitati rispetto alle opzioni fornite da altre librerie: non è infatti possibile implementare pulsanti interattivi, labels e tant'altro.
Ciò nonostante, è sempre possibile trovare soluzioni alternative che portino a risultati ugualmente soddisfacenti.

5.1 Il menù principale

Il primo passo è stato la creazione di un menù principale che accogliesse l'utente dopo aver avviato il main. Le opzioni a disposizione sono:
- **Gioca**: avvia il gioco.
- **About**: stampa sul terminale alcune informazioni sullo scopo del programma.
- **Test Carte**: analizza le foto all'interno delle cartelle CarteA, CarteB e CarteC alla ricerca di eventuali problemi.
- **Carte Coperte?**: permette all'utente di scegliere se giocare a carte coperte o a carte scoperte.
- **Exit**: esce dal programma.

Il menù principale

Per realizzare il menù, è necessario innanzitutto inizializzare una finestra che lo delimiti; questa viene generata con tre parametri specifici che dettano le sue caratteristiche[25]:
- **CV_WINDOW_AUTOSIZE**: la finestra assume le dimensioni dell'immagine al suo interno.
- **CV_WINDOW_KEEPRATIO**: la finestra mantiene le proporzioni dell'immagine al suo

interno.
- **CV_GUI_NORMAL**: la finestra perde la barra degli strumenti che altrimenti viene visualizzata di default quando si apre una foto con OpenCV.

Una volta generata la finestra, è possibile assegnarle la foto contenente il menù: a livello tecnico, infatti, il menù principale non è nient'altro che un'immagine in cui i pulsanti sono stati aggiunti alla foto stessa grazie ad un programma di fotoritocco; va notato che questi pulsanti si sarebbero potuti implementare con le funzioni di OpenCV a tempo di esecuzione, disegnando rettangoli, riempiendo il loro interno con un colore unico e scrivendoci sopra il testo. Tuttavia il risultato finale non sarebbe stato esteticamente piacevole quanto la soluzione finale adottata, per non parlare delle difficoltà nel posizionamento dei pulsanti stessi quando non si sta lavorando con gli occhi sulla foto, ma su un codice astratto.
A prescindere dalle soluzioni adottate, questi pulsanti sono puramente estetici, in quanto, si ricorda, non è possibile implementare pulsanti interattivi in OpenCV.
Quello che OpenCV permette, tuttavia, è l'invio di eventi quando il mouse interagisce con un'immagine: catturandone uno è possibile determinare quale parte della foto è stata interessata e se è stato premuto qualche pulsante[25].
Questo significa che è possibile delimitare quelle parti dell'immagine che corrispondono ai pulsanti, in modo che vengano eseguite istruzioni precise quando viene premuta una zona al loro interno, proprio come si comporterebbe un pulsante vero e proprio.
Alla finestra viene dunque assegnata una funzione CallBack che si occupa di catturare e gestire gli eventi, e che viene automaticamente eseguita in un processo parallelo: questa funzione è di tipo void e, non potendo ritornare alcun tipo di risultato, si limita al cambiamento di una serie di variabili globali; il processo originale, invece, scorre continuamente alcune di queste variabili, in attesa che qualcuna di esse sia alterata in seguito ad un evento.
Il processo originale controlla anche che la finestra contenente il menù non sia stata chiusa con il pulsante "x" in alto a destra: contrariamente a quanto ci si potrebbe aspettare, chiudere una finestra in quel modo non solo non termina l'esecuzione del programma, ma non invia nessun tipo di evento: è dunque compito del processo originale richiedere la terminazione del programma dopo aver notato la chiusura della finestra.

Ora si analizza cosa succede di preciso quando l'utente preme uno dei pulsanti:
- Quando l'utente decide di giocare, il processo che gestisce gli eventi aggiorna quella specifica variabile globale che si occupa di segnalare questa intenzione. Il processo originale si accorge del cambiamento, chiude la finestra con il menù e ritorna al main l'opzione di gioco scelta dal giocatore.

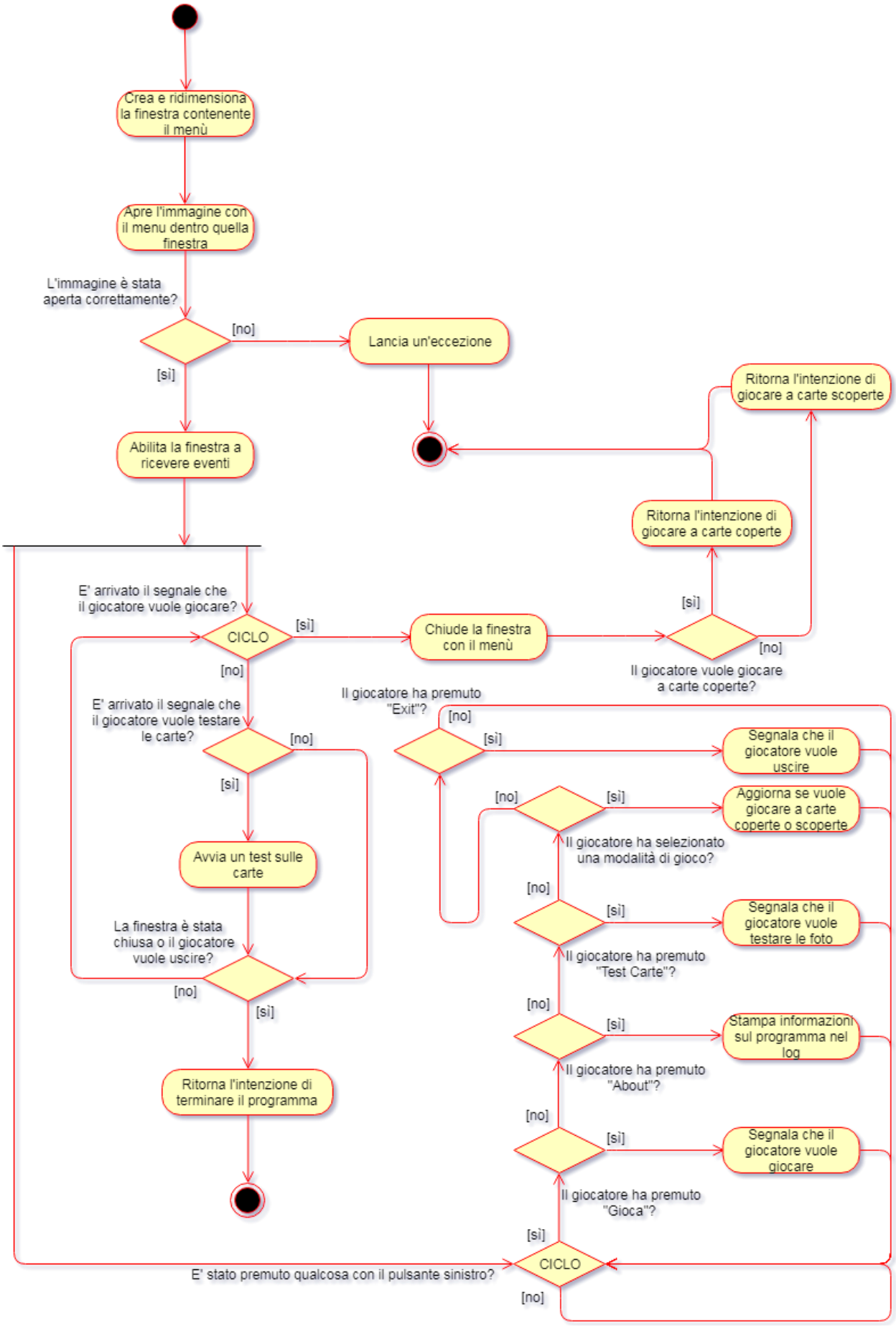


Diagramma attività delle funzioni **menu** e **CallBackMenu** in **ScopaUtil.cpp**

- Di default, l'opzione di gioco selezionata è quella a carte scoperte. L'utente può però scegliere la sua opzione preferita cliccando su "Sì" o "No" sotto la casella "Carte Coperte?": in questo modo si aggiorna una variabile globale che salva le preferenze del giocatore, mentre una freccia nera sulla GUI riflette visivamente l'opzione attualmente attiva. E' da notare che il cambiamento della freccia nera non è dovuta ad un'animazione: in realtà esistono due immagini del menù principale che come unica differenza hanno il posizionamento della freccia. Quando si altera un'opzione, viene dunque semplicemente scambiata l'immagine all'interno della finestra.

La freccia menzionata sopra

La mano della CPU quando è coperta

- Nel caso l'utente volesse testare le foto, il processo originale chiama, in seguito all'alterazione della variabile globale corrispondente, una funzione che si occupa di creare carte con le immagini date in input, per poi sottoporle ad un processo di recognition. Ciò avviene dietro le quinte, mentre sul terminale vengono stampati i progressi e sulla GUI viene mostrato un avviso che indica lo svolgimento del test in corso. Per tutta la durata del test, la capacità di catturare gli eventi viene disattivata. Terminato il processo, la variabile globale viene ripristinata, altrimenti il processo originale entrerebbe in un loop infinito.

Nota: esiste anche la versione di questa schermata con il pulsante "Sì" selezionato

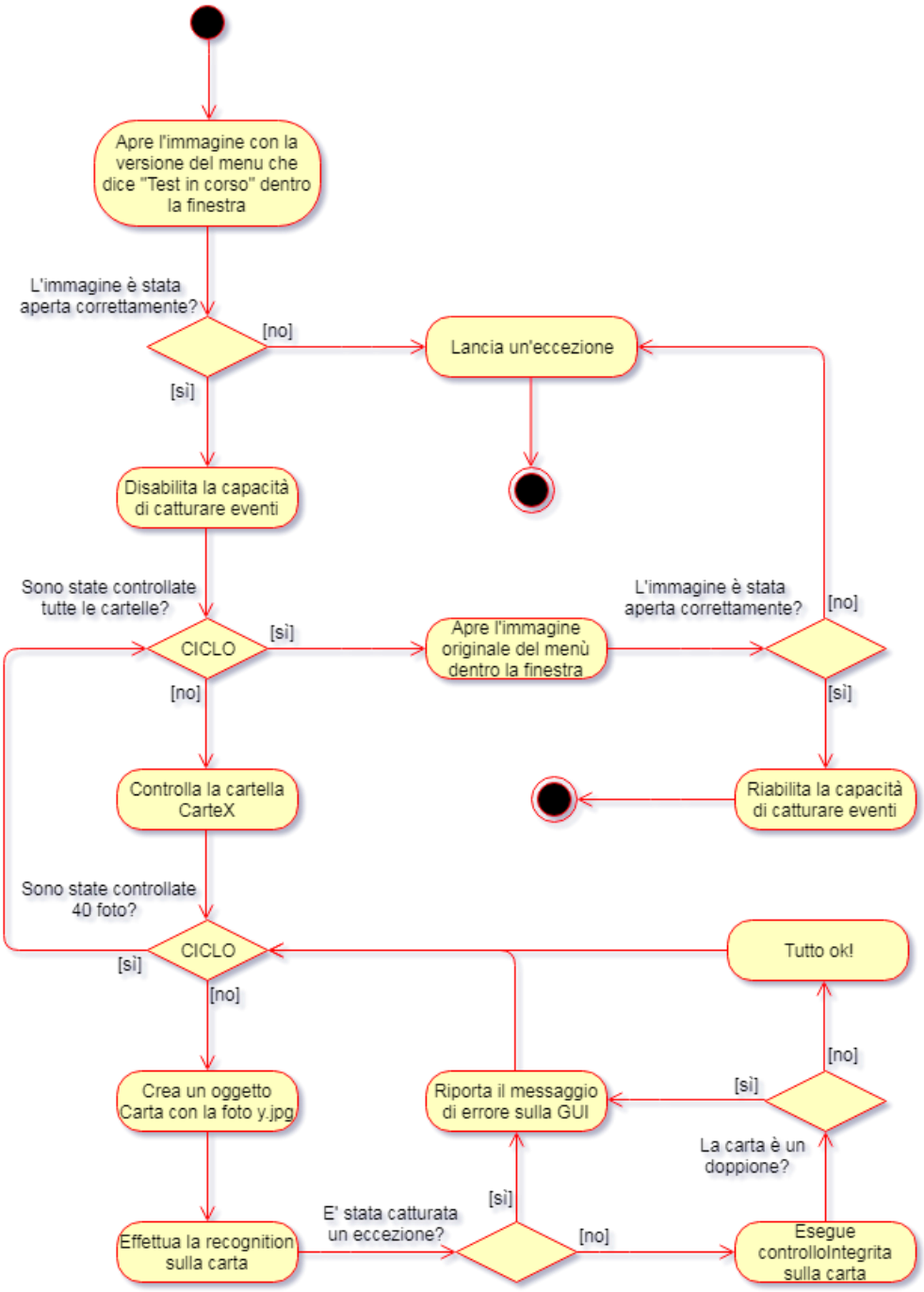


Diagramma attività di **controllaCarte** in **ScopaUtil.cpp**

- Premendo “About” non viene alterata nessuna variabile globale: vengono semplicemente stampate sul terminale alcune informazioni riguardo al programma.
- Premendo “Exit”, invece, si ha un processo simile a quello che avviene quando si preme “Gioca”, con la differenza che il menù ritorna l’intenzione dell’utente di uscire, anziché di giocare.

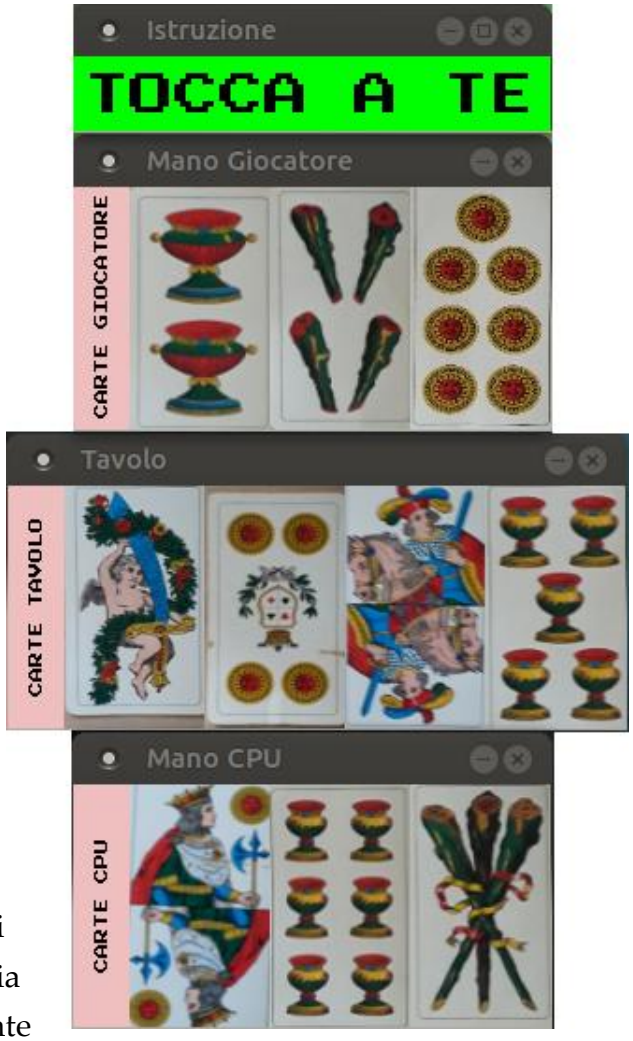
Tutte le foto utilizzate nella creazione della GUI, comprese quelle che contengono le varie versioni del menù principale e le labels, si trovano nella cartella “Grafica”: la loro rimozione comporta delle eccezioni.

5.2 La schermata di gioco e il turno del giocatore

Come si può notare nell'immagine a lato, si è deciso di dividere la schermata di gioco in più finestre: tre contenenti un set di carte, più una che indica se è il turno del giocatore.

In teoria si sarebbe potuta utilizzare una finestra unica, ma OpenCV permette di assegnare una sola foto a finestra. Questo significa che si sarebbero dovute incollare una ad una tutte le carte su uno sfondo, formando una singola immagine che avrebbe dovuta essere ricreata daccapo ad ogni singolo cambiamento.

Dividendo la schermata in questo modo, invece, si alterano solo le finestre che subiscono un cambiamento, lasciando intatte tutte le altre.



La schermata di gioco

Questo significa anche che i set di carte, più le labels che le precedono, sono in realtà delle singole immagini, generate da una funzione di Mazzo.cpp che ridimensiona le fotoZoom, ossia quelle immagini che contengono esclusivamente le carte trovate dopo una object recognition, per poi concatenarle tra loro e con le labels[23].

La scelta di utilizzare le labels è stata essenziale, poiché rendono facilmente comprensibile il contenuto di ogni finestra.

Va anche notato che l'ordine delle finestre mostrato nell'immagine non è affatto casuale: dopo averle generate, con i stessi parametri usati per la creazione di quella contenente il menù, è infatti possibile assegnare loro le coordinate che devono assumere una volta mostrate su schermo[25].

Quando è il turno del giocatore, alle finestre contenenti le carte vengono assegnate delle funzioni CallBack che catturano eventi e che modificano una serie di variabili globali, proprio come si è visto nella creazione del menù. Quando il giocatore tocca una qualsiasi carta, si aggiorna quindi una variabile globale che indica quale carta è stata selezionata e con quale pulsante.

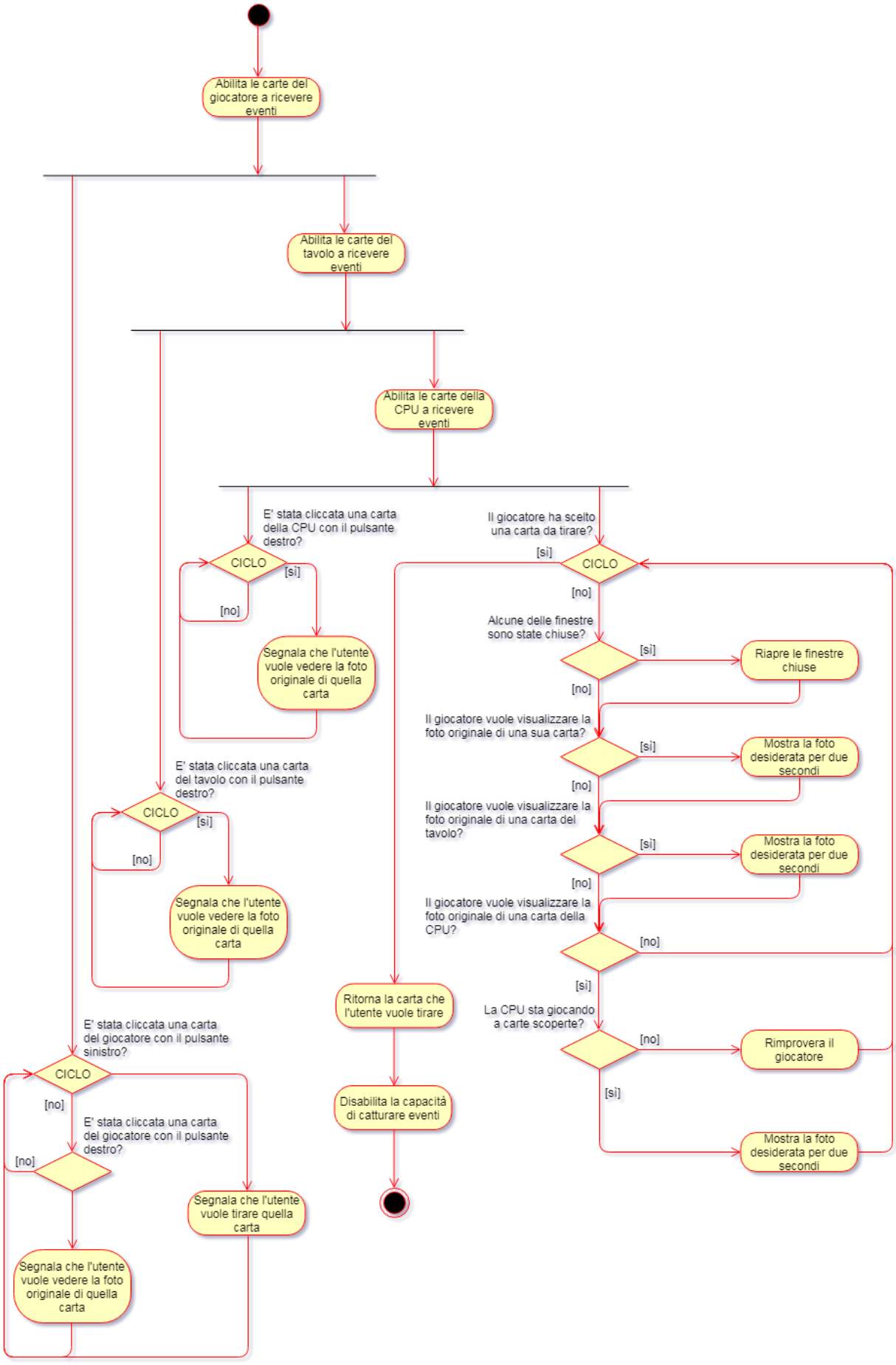
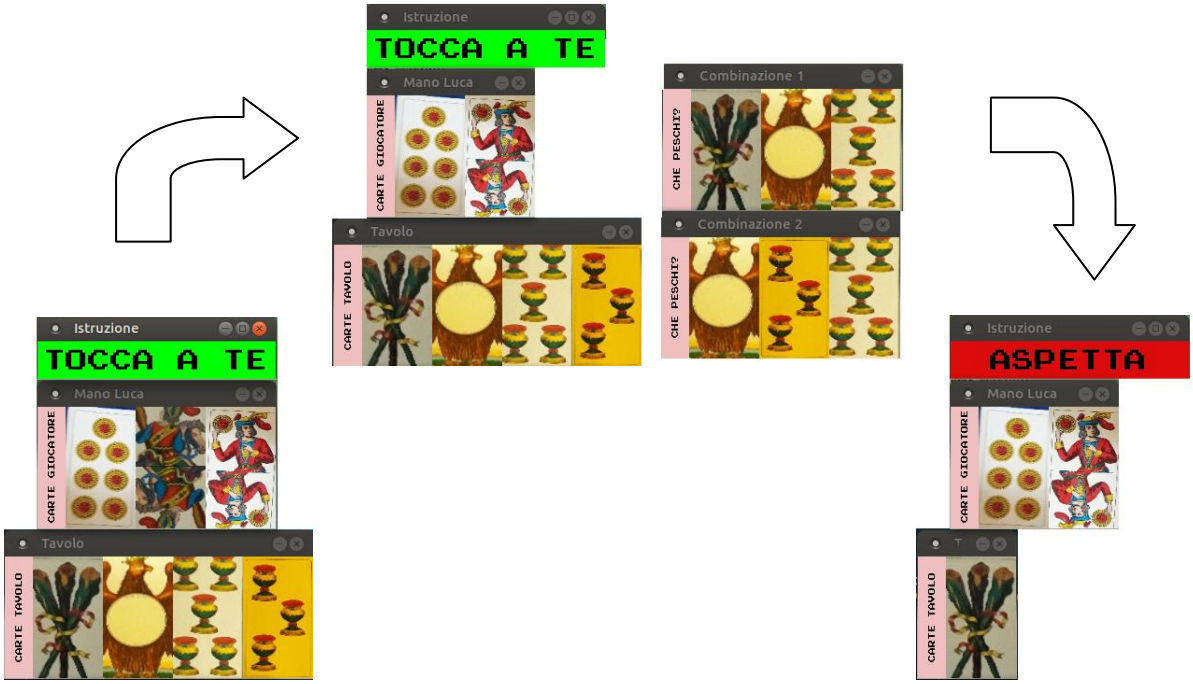


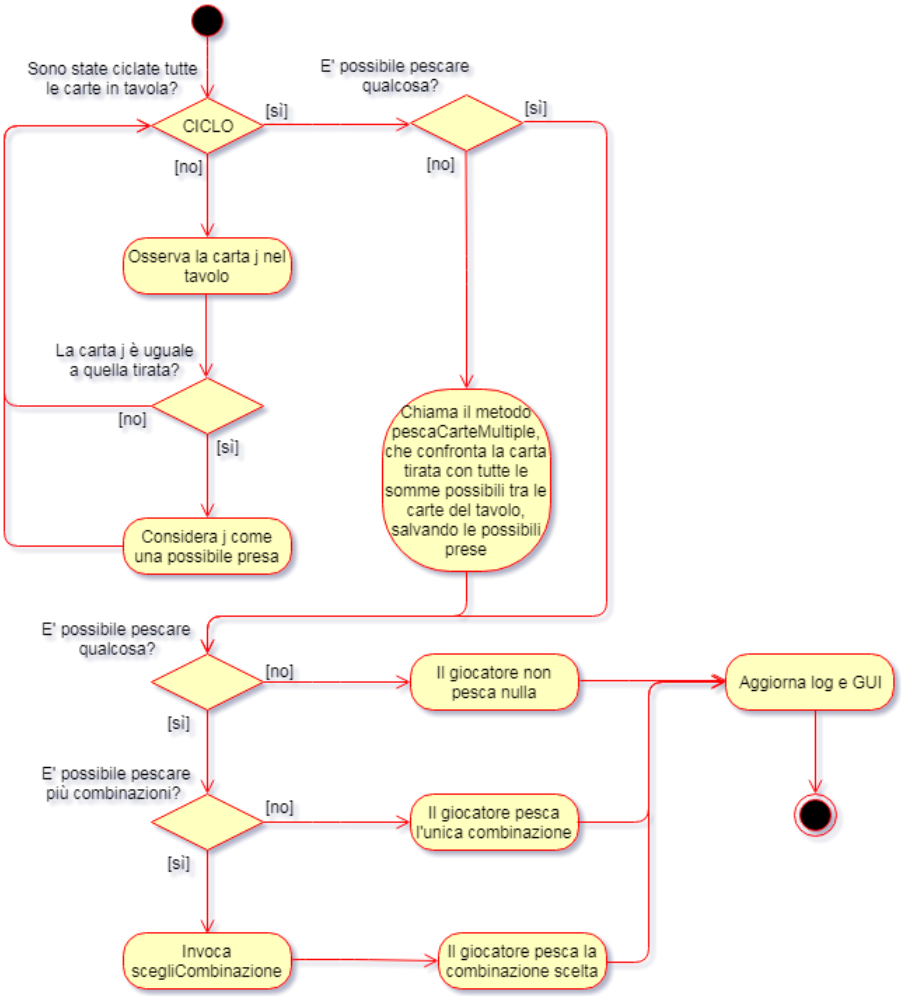
Diagramma attività delle funzioni **toccaCarta**, **CallBackManoGiocatore**, **CallBackTavolo** e **CallBackManoCPU** in **ScopaUtil.cpp**

Il processo originale, accorgendosi del cambiamento, si comporta di conseguenza:
- Se il giocatore ha toccato una qualsiasi carta con il pulsante destro, allora viene mostrata su schermo per un paio di secondo la fotoRettangolo corrispondente, ossia la foto originale data in input in cui la carta al suo interno è contraddistinta da un rettangolo blu.
Ovviamente se viene toccata una carta della CPU, e quest'ultimo sta giocando a carte coperte, la foto non viene visualizzata.
- Se invece il giocatore ha premuto il pulsante sinistro su una carta in suo possesso, allora quella carta viene sottratta dalla sua mano e tirata in gioco, dopodiché le finestre perdono la loro capacità di catturare eventi: ciò avviene perché, a differenza del menù, quest'ultime non vengono chiuse alla fine del processo; il giocatore sarebbe quindi in grado di cliccare sulle carte al loro interno e cambiare le variabili globali anche quando non è il suo turno se non fosse presa questa precauzione.

Un aspetto che non viene mostrato nello schema UML per mancanza di spazio è che le variabili globali vengono resettate ad ogni input, evitando la possibilità di loop infiniti.
Quando il giocatore tira la sua carta, il programma esegue una serie di calcoli, molto simili a quelli già analizzati nel paragrafo 4.4, per determinare se e cosa può pescare: prima vengono effettuati i confronti tra carte singole, poi tra la carta tirata e le somme di tutte le permutazioni delle carte sul tavolo.
Nel caso dovessero esserci più possibili prese per la carta tirata, la decisione sul cosa pescare viene lasciata interamente al giocatore, che deve toccare la combinazione desiderata; altrimenti viene pescata automaticamente l'unica combinazione possibile.
La conclusione del turno viene comunicata dalla finestra più in alto, che diventa rossa.



Esempio in cui il giocatore tira il 9 di bastoni e pesca la combinazione 2



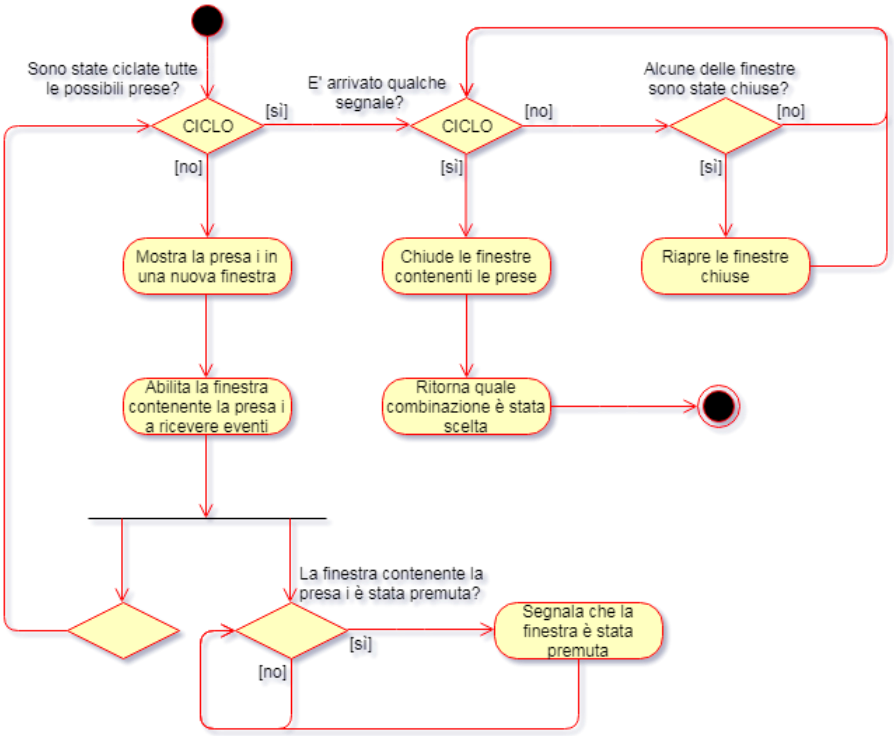


Diagramma attività di **mossaGiocatore**, **scegliCombinazione** e **CallBackCombinazione** in **ScopaUtil.cpp**

5.3 La schermata di fine partita e il calcolo dei punteggi

Terminata una partita, appaiono una serie di finestre che mostrano le carte pescate dal giocatore, quelle pescate dalla CPU e chi ha vinto la partita, mentre sul log c'è una descrizione dettagliata sui punti ottenuti da ciascun giocatore nelle varie categorie.
Il giocatore può quindi decidere se continuare il gioco in un'altra finestra abilitata al ricevere eventi, che funziona esattamente come quelle di tipo simile già esaminate, e che mostra due pulsanti, "Sì" e "No."

La schermata di fine gioco

Per realizzare la schermata, ci sono due funzioni: una che si occupa di calcolare i punteggi per poi stamparli nel log, riportando infine l'esito della partita, l'altra che invece realizza la GUI servendosi dell'esito riportato.
Come si è menzionato in precedenza, il calcolo del punteggio segue il regolamento ufficiale e i punti vengono assegnati in base a:

- Numero di denari
- Numero di scope
- Numero di carte pescate
- Primiera
- Presenza del 7 Bello

Nel caso l'utente prema "no", il programma termina.

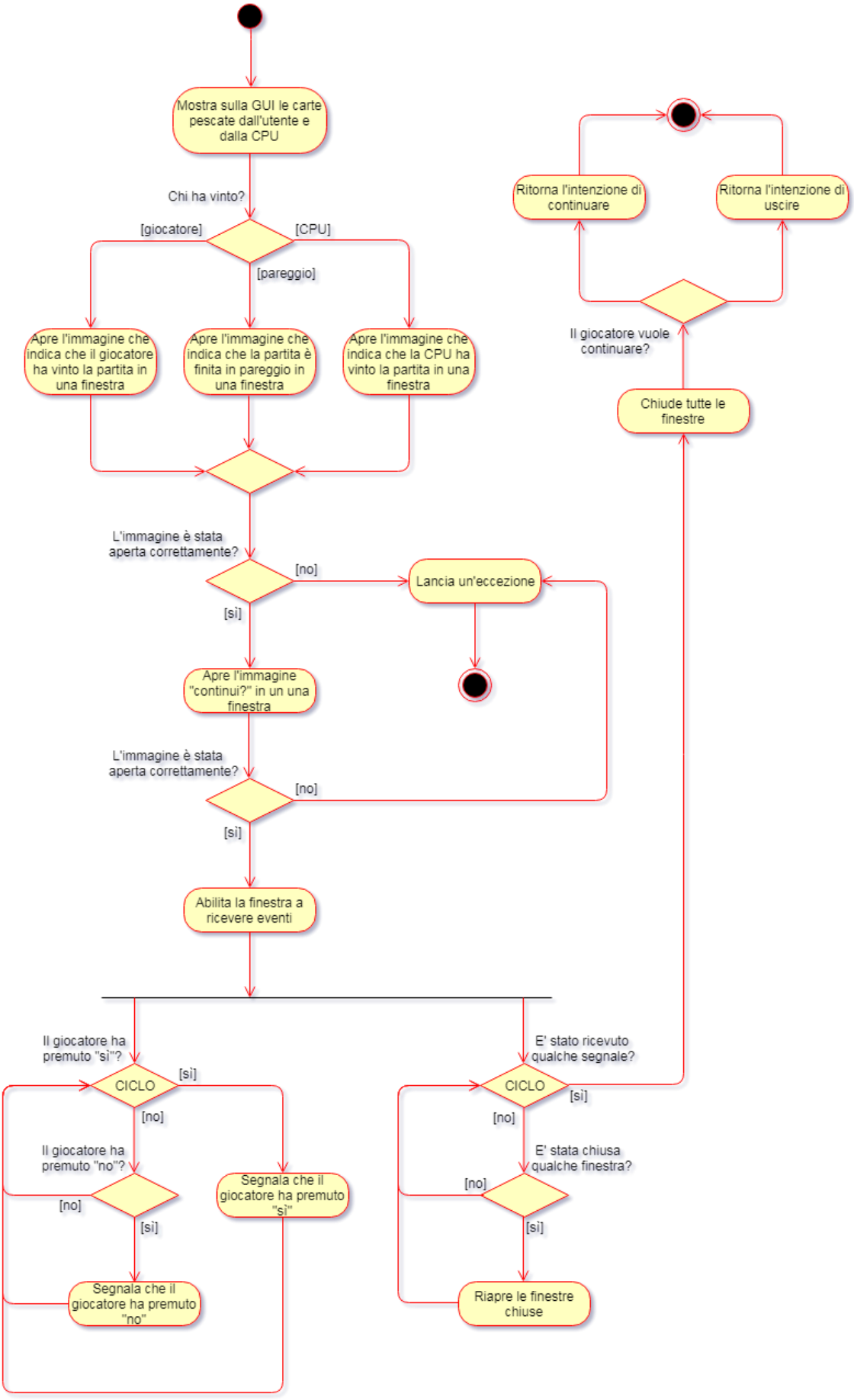


Diagramma attività di **continuaPartita** e **CallBackContinui** in **ScopaUtil.cpp**

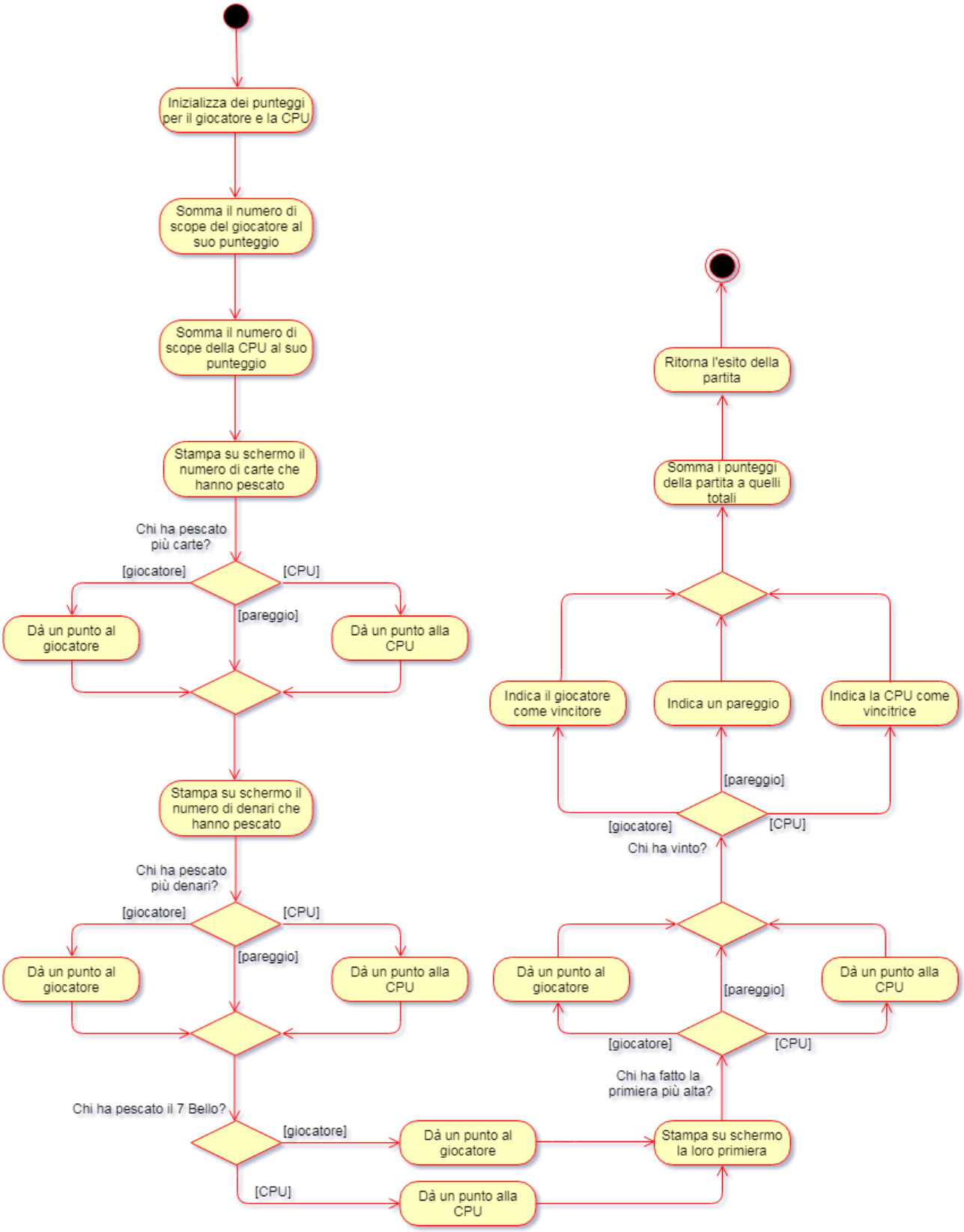


Diagramma attività di **calcolaPunteggio** in **ScopaUtil.cpp**

Per calcolare cose come la primiera, o il numero di denari, calcolaPunteggio invoca una serie di funzioni contenute in Mazzo.cpp, di cui non vengono mostrati i diagrammi attività per la loro relativa semplicità.

6. Brevi conclusioni sul progetto

Con l'implementazione della GUI, la realizzazione del programma è finalmente terminata: si sono dunque aggiunte le 120 foto necessarie per riempire le cartelle CartaA, CartaB e CartaC, con l'intenzione di testarne la validità con la funzione "Test Carte" del menù principale.
Le prime due contengono tutte le carte in un solo setting, rispettivamente una scrivania e un letto, con condizioni di luminosità molto simili, per testare l'efficacia del riconoscimento delle carte senza badare troppo alle incognite che ambientazioni sempre diverse avrebbero inevitabilmente aggiunto al processo di recognition. CarteC, invece, possiede un set di foto più ambizioso e variegato, che sfrutta le potenzialità del Cascade fino al limite.
Dopo averle testate e dopo aver rimpiazzato quella manciata di foto che non avevano dato i risultati sperati, probabilmente per il fatto che erano state scattate sfocate o con condizioni di luminosità avverse, si è potuto provare il gioco stesso nella modalità a carte coperte.
La prestazione dell'IA avversaria è stata sicuramente l'aspetto più interessante da analizzare: fortunatamente le tattiche della CPU si sono rivelate convincenti ed efficaci sin da subito, il che non è affatto scontato nel passaggio dal codice teorico alla pratica stessa. Tuttavia dopo decine di partite si è riscontrata una prevalenza delle vittorie del giocatore, molto lieve, ma sufficientemente grande da farsi notare: più precisamente si è notato che l'utente ha vinto circa il 60% delle partite, mentre il rimanente si è concluso in pareggi o vittore della CPU.
Nonostante questo gap del 10% si possa spiegare come semplice fortuna da parte del giocatore, in quanto, come ogni gioco di carte in circolazione, la qualità delle carte ricevute è altrettanto importante a quella delle tattiche effettuate, è probabile che la motivazione più pertinente sia un'altra: la CPU, per quanto segua diligentemente quelle tattiche che su carta dovrebbero portarlo alla vittoria, non possiede una memoria fotografica che gli permette di ricordarsi di ogni singola carta uscita nel gioco, altrimenti darebbe la sensazione di barare: il giocatore, invece, può potenzialmente ricordarsi di più fattori, e con un occhio rivolto al futuro nel tentativo di ipotizzare quale carte potranno o non potranno uscire nella prossima mano, può implementare una varietà di tattiche meno convenzionali, spesso basate sull'istinto, ma non per questo meno valide e che possono portarlo ad avere la meglio contro la CPU
Durante la fase di prova del gioco sono stati corretti anche tutti i vari bug scaturiti da situazioni in cui l'utente si è comportato in un maniera imprevista, ad esempio chiudendo le finestre della GUI o cliccando le carte anche quando il suo turno era già terminato.

Riferimenti

[1] Cass, Stephen. "The 2015 top ten programming languages." IEEE Spectrum, July 20 (2015).

[2] «OpenCV Official Website,» [Online]. Available: https://opencv.org/.

[3] Culjak, Ivan, et al. "A brief introduction to OpenCV." MIPRO, 2012 proceedings of the 35th international convention. IEEE, 2012.

[4] «Marrtino Robot Official Website,» [Online]. Available: https://sites.google.com/dis.uniroma1.it/marrtino/software?authuser=0.

[5] «OpenCV Face Detection Tutorial,» [Online]. Available: https://docs.opencv.org/3.4/d7/d8b/tutorial_py_face_detection.html.

[6] «OpenCV Cascade Classification,» [Online]. Available: https://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html.

[7] Paul Viola and Michael J. Jones. Rapid object detection using a boosted cascade of simple features. In Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, volume 1, pages I–511. IEEE, 2001.

[8] Haykin Simon, "A comprehensive foundation." Neural networks 2.2004 (2004): 41.

[9] Ruck, Dennis W., Steven K. Rogers, and Matthew Kabrisky. "Feature selection using a multilayer perceptron." Journal of Neural Network Computing 2.2 (1990): 40-48.

[10] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[11] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[12] Perez, Luis, and Jason Wang. "The effectiveness of data augmentation in image classification using deep learning." arXiv preprint arXiv:1712.04621 (2017).

[13] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt and B. Scholkopf, "Support vector machines," in IEEE Intelligent Systems and their Applications, vol. 13, no. 4, pp. 18-28, July-Aug. 1998.

[14] Chih-Wei Hsu and Chih-Jen Lin, "A comparison of methods for multiclass support vector machines," in IEEE Transactions on Neural Networks, vol. 13, no. 2, pp. 415-425, March 2002.

[15] «OpenCV Default Cascades,» [Online]. Available: https://github.com/opencv/opencv/tree/master/data/haarcascades.

[16] «OpenCV Cascade Training Tutorial,» [Online]. Available: https://docs.opencv.org/3.4/dc/d88/tutorial_traincascade.html.

[17] «ImageNet,» [Online]. Available: http://image-net.org/.

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, ImageNet: A Large-Scale Hierarchical Image Database. IEEE Computer Vision and Pattern Recognition (CVPR), 2009.

[19] «OpenCV Boosting,» [Online]. Available: https://docs.opencv.org/2.4/modules/ml/doc/boosting.html.

[20] Hastie Trevor, Tibshirani Robert, and Friedman Jerome. The elements of statistical learning: data mining, inference and prediction. New York: Springer-Verlag, 1(8):371–406, 2001.

[21] «OpenCV Histogram Equalization,» [Online]. Available: https://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/histogram_equalization/histogram_equalization.html.

[22] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, volume 1, pages 886–893. IEEE, 2005.

[23] «OpenCV Operations On Arrays,» [Online]. Available: https://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html.

[24] Liwei Wang, Yan Zhang and Jufu Feng, "On the Euclidean distance of images," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, no. 8, pp. 1334-1339, Aug. 2005.

[25] «OpenCV User Interface,» [Online]. Available: https://docs.opencv.org/2.4/modules/highgui/doc/user_interface.html.