POLITECNICO DI TORINO

DIPARTIMENTO DI AUTOMATICA E INFORMATICA

Corso di Image Processing and Computer Vision

Ingegneria Informatica Grafica e Multimedia

Relazione di progetto:

Implementazione del gioco della Morra Cinese

Docente:

Bartolomeo Montrucchio

Studenti:

Bresciani Alessandro, 305903

Ciardo Roberto, 287279

Rovegno Carolina, 306124

Esercitatore:

Luigi De Russis

Anno accademico

2021/2022

Contents

1 Introduzione

La relazione di progetto si basa sull'implementazione del famoso gioco della Morra Cinese. Il progetto si propone l'obbiettivo di realizzare un'applicazione che permetta all'utente di sfidare il calcolatore in una serie di partite uno contro uno, impiegando le competenze acquisite durante il corso di Image Processing and Computer Vision. Il programma sfrutta il tracking della mano del giocatore per riconoscere la mossa effettuata e confrontarla con l'opzione scelta dal computer per decretare il vincitore.
Sono state approfondite queste specifiche funzionalità:

1. rilevazione della mano e dei rispettivi landmarks: estrazione della posizione della mano all'interno della viewport, ricerca dei pixels associati agli specifici punti chiave e conseguente marcatura di questi ultimi;
2. gestione dello stato e aggiornamento dello stesso relativo alla visualizzazione del layout: identificazione della schermata di menu, di gioco e di risultati ottenuti;
3. classificazione della mossa operata dall'utente e conseguente risposta dell'elaboratore;
4. decretazione del vincitore nella schermata di "result".

È necessario l'utilizzo di una webcam integrata nel pc e di una versione di Python 3.10. Per eseguire l'applicazione è necessario eseguire lo script python MorraCinese_TES.py da linea di comando o da IDE.

2 Librerie

2.1 Files e librerie: descrizione

Questo paragrafo illustra le librerie e files che sono stati utilizzati all'interno del progetto trattato.
Per cominciare il progetto è costituito da due file.py:

- HandTrackingModule_TES.py
- MorraCinese_TES.py

Inoltre, sono state utilizzate le librerie principali:

- OpenCV
- MediaPipe

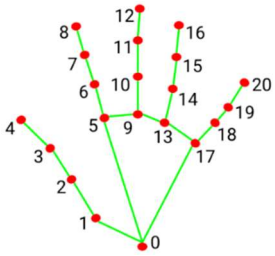
2.1.1 Libreria OpenCV

OpenCV (cv2) è una libreria open source usata nel campo dell'image processing e della computer vision. Comprende più di 2500 algoritmi che permettono di effettuare operazioni di analisi ed elaborazione di immagini e video come il riconoscimento di oggetti (object recognition), di volti (face recognition), la classificazione di movimenti, l'estrazione di modelli 3D.
In questo progetto è stata utilizzata per gestire l'acquisizione delle immagini attraverso la webcam e per la creazione del layout della UI, attraverso la quale l'utente ha la possibilità di interagire con il gioco.

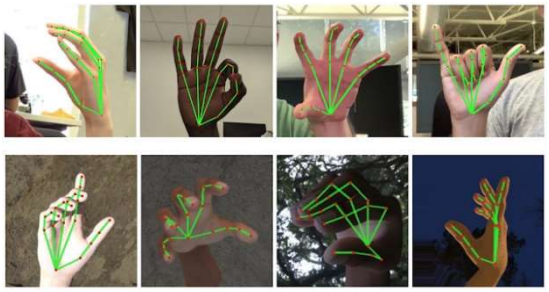
2.1.2 Framework MediaPipe

Mediapipe è un framework sviluppato da Google e messo a disposizione degli utenti. Esso offre soluzioni di Machine Learning multipiattaforma per contenuti in diretta e in streaming quali: riconoscimento facciale, Iris, Hand Detector, Posa, rilevamento di oggetti e molto altro.
Per il progetto realizzato è stato impiegato il modulo MediaPipe Hands, ossia una soluzione di tracciamento di mani e dita ad alta fedeltà. Questa è composta da una serie di moduli che lavorano all'unisono: un modello per il rilevamento del palmo (Palm Detection Model) che restituisce un riquadro orientato (bounding box)

della mano e un modello (Hand Landmark Model) che sfrutta l'apprendimento automatico per dedurre 21 punti di riferimento 3D di una mano attraverso un fotogramma.
Il primo modello fornisce l'immagine della mano ritagliata al secondo modello, ciò aumenta la capacità di predizione delle coordinate dei landmarks. Viene inoltre invocato quando il modello dei punti chiave non è più in grado di identificare la presenza della mano. In questo modo è possibile rilocalizzare il bounding box.
Dopo il rilevamento del palmo, l'Hand Landmark Model esegue una localizzazione precisa dei 21 punti chiave relativi alle nocche della mano e della sua posa; quest'ultimo è robusto anche in presenza di mani parzialmente visibili.
[https://google.github.io/mediapipe/solutions/hands]



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP



2.1.3 Installazione librerie

Per poter installare Mediapipe, occorre scrivere da terminale:

```
(mp_env)$ pip install mediapipe
(mp_env)$ python3
```

In alternativa, cercare all'interno di Python Packages la libreria Mediapipe e subito dopo installarla.

2.1.4 Altre Librerie

Come ulteriore libreria esterna abbiamo utilizzato, come da Laboratori, la libreria **Matplotlib**.
Quest'ultima è stata fondamentale per la visualizzazione e la gestione della finestra di gioco.

3 Struttura del programma

3.1 Modulo di Hand Tracking: descrizione

Il modulo di handTracking (file handTrackingModule_TES.py) è costituito da una classe base all'interno di una logica ad oggetti.

3.1.1 Parametri e Costruttore della classe

Il Costruttore setta tutti i parametri per l'individuazione della mano dell'utente, fondamentali per l'avvio dell'applicazione; si appoggia al framework di Mediapipe. I parametri trattati sono:

- **Mode:**
  Se impostato su FALSE, la soluzione considera le immagini di input come un flusso video. Tenterà di rilevare le mani nelle prime immagini di input e, in caso di rilevamento riuscito, localizzerà ulteriormente i punti di riferimento della mano. Nelle immagini successive, una volta rilevate tutte le mani max_num_hands e localizzati i corrispondenti punti di riferimento, tiene semplicemente traccia di tali landmarks senza invocare un altro rilevamento fino a quando non perde traccia di nessuna delle mani. Ciò riduce la latenza ed è ideale per l'elaborazione di fotogrammi video. Se

impostato su TRUE, il rilevamento delle mani viene eseguito su ogni immagine di input, ideale per l'elaborazione di un batch di immagini statiche, possibilmente non correlate. Predefinito su FALSE.

- **maxHands**:
  Numero massimo di Landmarks da rilevare. Predefinito su 2.
- **detectionCon**:
  Valore di confidenza minimo ([0.0, 1.0]) dal modello di rilevamento della mano affinché il rilevamento sia considerato riuscito. Predefinito su 0.5.
- **trackCon:**
  Valore di confidenza minimo ([0.0, 1.0]) dal modello di tracciamento dei punti di riferimento affinché i landmarks della mano siano considerati tracciati correttamente, altrimenti il rilevamento della mano verrà richiamato automaticamente nell'immagine di input successiva. Impostandolo su un valore più elevato è possibile aumentare la robustezza della soluzione, a scapito di una latenza più elevata. Ignorato se static_image_mode è TRUE, dove il rilevamento delle mani viene eseguito semplicemente su ogni immagine. Predefinito su 0.5.

3.1.2 Metodi utilizzati

All'interno della classe sono definiti due metodi fondamentali per la ricerca e la visualizzazione della mano dell'utente:

1. **Metodo *findHands(…)* :**
   Tramite questo metodo viene effettuato il process dell'immagine acquisita, su di essa vengono rilevate le mani presenti e ritorna un oggetto che viene costruito tramite la libreria di Mediapipe. Solamente se vengono rilevate le mani all'interno dell'immagine, vengono segnati i landmarks acquisiti.
2. **Metodo *findPosition(…)* :**
   Attraverso questa funzione viene processato il risultato ritornato dal metodo precedente in modo da costruire una lista (*lmList*), facilmente fruibile, costituita da una serie di Array. Ognuno di essi è costituito dall' id del singolo landmark e dalle coordinate del suo centro in pixels.

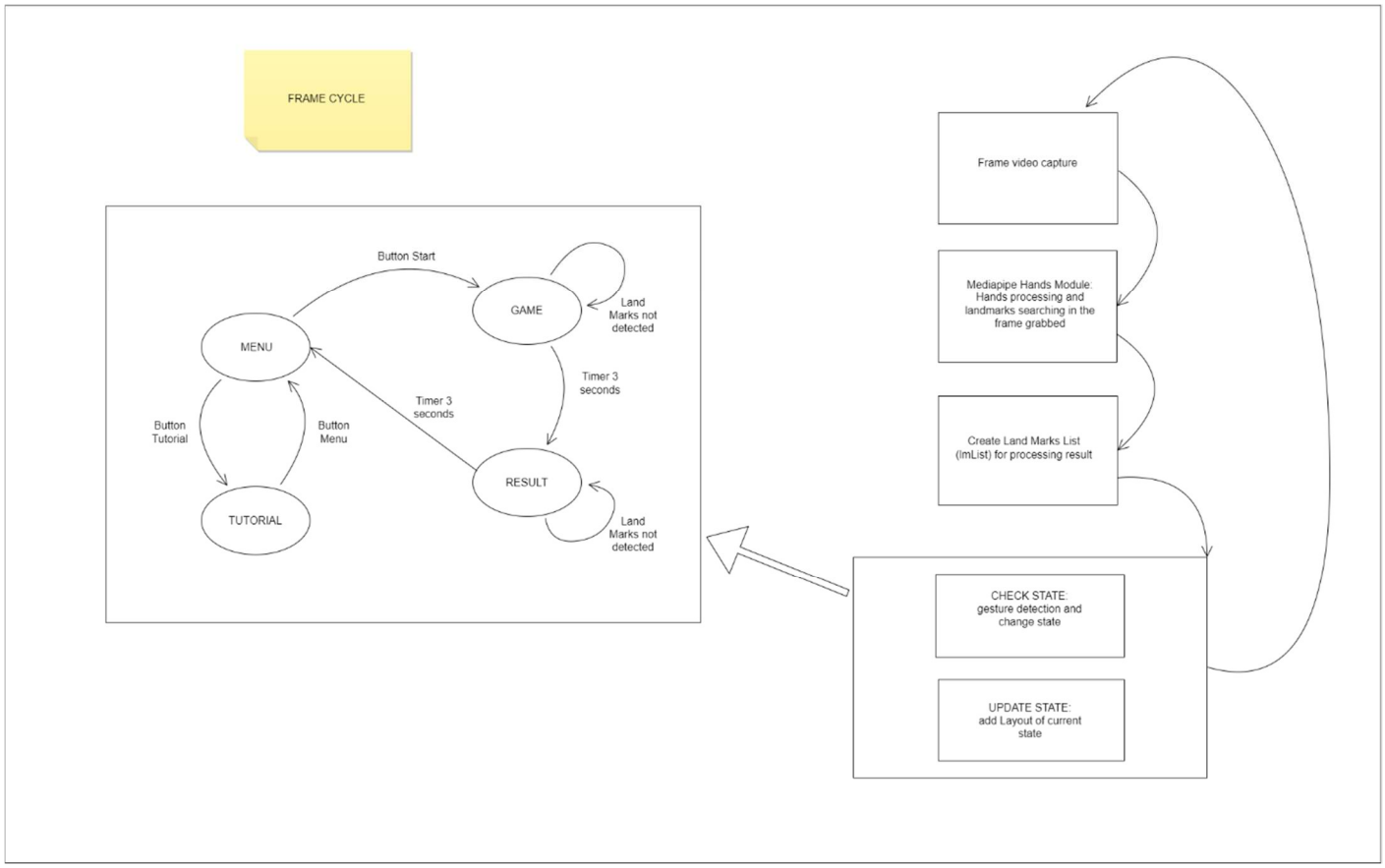
Inoltre, tramite questa funzione, se viene passato il parametro *draw = true*, avviene il disegno all'interno della schermata di cerchi corrispondenti ai landmarks.

3.2 Diagramma di Stato dell'applicazione

Di seguito è riportato il diagramma del ciclo di un frame.

Per iniziare, viene acquisito il frame tramite la funzione *cv2.videoCapture(0)* aprendo la connessione con la webcam. Subito dopo viene chiamata la funzione *grab_frame()* a cui viene passata l'istanza della camera e di un detector (paragrafo 3.1), ossia il riferimento acquisito dall'handTrackingModule.

L'applicazione impiega due possibili modi di esecuzione: è possibile giocare in modalità standard, oppure attraverso una modalità di Debug. Ciò è settabile per mezzo della variabile globale booleana *DebugMode.*



3.2.1 Frame video capture

Inizialmente viene acquisito il frame tramite la funzione *cap.read()* che impiega l'istanza della video capture. Quest'ultimo viene successivamente processato per un utilizzo che favorisca l'utente: viene capovolta la visualizzazione e, in seguito, salvata una copia dello stesso.

```
def grab_frame(cap, detector):
    global tTime, cTime, pTime

    ret, frame = cap.read()
    frame = cv2.flip(frame, 1)
    frameCopy = cp.copy(frame)
```

3.2.2 Landmarks searching in the frame grabbed e Create Land marks List (lmList)

Viene rilevata in successione la presenza delle mani e dei landmarks dell'utente attraverso la chiamata all'HandTrackingModule per poi fornire direttamente la lista contenente gli id e le posizioni di questi ultimi.

Il tutto si esaurisce per mezzo della chiamata ai metodi riportati ai paragrafi 3.1.1 e 3.1.2.

```
img = detector.findHands(frame, DebugMode)  # meto
lmList = detector.findPosition(img, 0, DebugMode)
gestureDetection(lmList, img, cap)
```

Per ogni frame viene chiamata la funzione *gestureDetection()* che inizialmente riempie una lista con le posizioni delle singole dita, si tratta della lista *fingers*: 1 per dita alzate o 0 per dita abbassate. La funzione inoltre distingue l'orientamento della mano rilevata identificando la direzione delle dita, verso l'alto oppure verso il basso.

```
fingers = []
# rilevo mano orientata verso l'ALTO (nocca sopra polso)
if lmList[9][2] < lmList[0][2]:
    #rilevo dito aperto/chiuso (riempio fingers[])
    for id in range(1, 5):
        if lmList[tipIds[id]][2] < lmList[tipIds[id] - 2][2]:
            fingers.append(1)
        else:
            fingers.append(0)
```

3.2.3 Macchina a stati finiti : CHECK STATE e UPDATE STATE

Sono stati definiti quattro stati fondamentali dell'applicazione :

```
class State(Enum):
    _Menu = 1
    _Game = 2
    _Result = 3
    _Tutorial = 4
```

CHECK STATE : la logica che definisce l'aggiornamento e la verifica dello stato si basa su due tipologie di eventi possibili:

1) Un evento è quello in cui le dita dell'utente rientrano all'interno di un bounding box.
   Il giocatore dovrà impiegare indice e medio per identificare sulla schermata un puntatore di colore verde. Solamente nel momento in cui quest'ultimo finirà all'interno del riquadro per la selezione voluta, il flusso di gioco continuerà generando un cambiamento di stato.

2) L'altro evento è identificato da un timer.
   Allo scadere di quest'ultimo (3 secondi) si verifica in automatico un cambiamento di stato.
   Tale soluzione è stata impiegata con successo per effettuare il passaggio dallo stato di Result (*State._Result*) allo stato di Menu (*State._Menu*) e da Game (*State._Game*) a Result.

UPDATE STATE: diversamente, la logica che processa l'immagine visualizzata rispetto allo stato corrente:

- Aggiunge le immagini relative alla schermata che deve essere visualizzata tramite opportune funzioni
- Inserisce, se necessario, il puntatore quando vengono rilevate l'indice e il medio della mano del giocatore
- Nel momento in cui si entra in uno stato che sfrutta il Timer, questo verrà aggiornato
- Infine, nel caso in cui la mano non venga rilevata, viene inserita una scritta che avvisa l'utente.

3.3 Descrizione specifica del codice

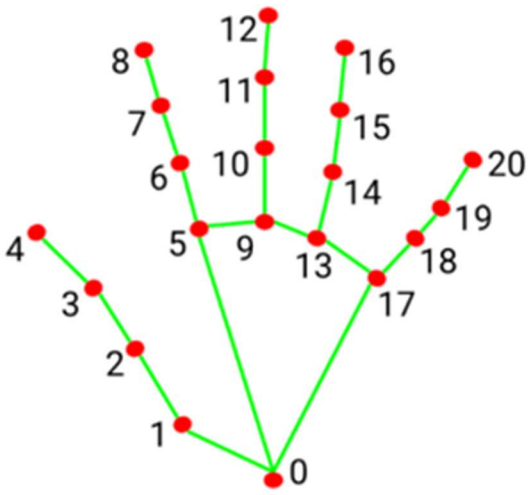
Di seguito è riportata una descrizione approfondita dei moduli principali che compongono l'applicazione e delle scelte implementative.

3.3.1 Riconoscimento della mossa

Per poter proseguire nel gioco è necessario interagire con la schermata per mezzo di due dita della mano, ossia l'indice e il medio. Per prima cosa è stato necessario rilevare quando le dita dell'utente fossero aperte oppure chiuse; viene di seguito riportata la logica implementata all'interno della funzione *gestureDetection():*

```
if len(lmList) != 0:
    fingers = []
    # rilevo mano orientata verso l'ALTO (nocca sopra polso)
    if lmList[9][2] < lmList[0][2]:
        #rilevo dito aperto/chiuso (riempio fingers[])
        for id in range(1, 5):
            if lmList[tipIds[id]][2] < lmList[tipIds[id] - 2][2]:
                fingers.append(1)
            else:
                fingers.append(0)

    # rilevo mano orientata verso l'BASSO (nocca sotto polso)
    else:
        #rilevo dito aperto/chiuso (riempio fingers[])
        for id in range(1, 5):
            if lmList[tipIds[id]][2] > lmList[tipIds[id] - 2][2]:
                fingers.append(1)
            else:
                fingers.append(0)
        #orientation = "basso"
```



Partendo dal fatto che l'orientamento dell'immagine, in pixels, è rivolto dall'alto verso il basso abbiamo utilizzato, da *lmList* (vedi 3.1.2), i landmarks di Id 9 e di Id 0 proposti da Mediapipe per comprendere se la mano fosse rivolta verso l'alto o verso il basso.

Dunque, se la coordinata y del pixel, ottenuta tramite *lmList[**9**][2]* è minore della coordinata ottenuta con *lmList[**0**][2],* dove il primo indice identifica uno specifico Id, mentre il secondo la coordinata y di quel landmark, allora la mano è rivolta verso l'alto, viceversa si identifica una mano rivolta verso il basso.

Per poter distinguere i vari casi di posizionamento della mano è stato impiegato un vettore *fingers* nel quale è salvato lo stato attuale delle dita (alzate o abbassate) tramite i valori 1 e 0. Per il riempimento dell'array si utilizza un vettore *tipIds* contenente i soli ids dei polpastrelli. Abbiamo acceduto alla lista dei landmarks confrontando l'id del polpastrello corrente con *id-2* (es. per l'indice 8, (8-2)).

Allo stesso modo, per il riconoscimento della mossa, è stata impiegata la seguente metodologia: a seconda della posizione delle dita è stata assegnata una variabile (*currentGesture*) che tenga traccia della gesture corrente, successivamente questa è stata usata per identificare la logica di scelta dell'utente stesso. La scelta del PC avviene ogni volta assegnando un valore randomico alla variabile *pcChoice*.

```
if fingers[0] == 0 and fingers[1] == 0 and fingers[2] == 0 and fingers[3] == 0:
    currentGesture = 0 # sasso
elif fingers[0] == 1 and fingers[1] == 1 and fingers[2] == 1 and fingers[3] == 1:
    currentGesture = 1 # carta
elif fingers[0] == 1 and fingers[1] == 1 and fingers[2] == 0 and fingers[3] == 0:
    currentGesture = 2 # forbice
else:
    currentGesture = -1
```

```
playerChoice = currentGesture
pcChoice = random.randint(0, 2)
```

Ugualmente per poter disegnare il puntatore associato all'indice e al medio è stato costruito un cerchio e, identificandone il centro, è stato disposto tra le due dita.

```
if len(lmList) != 0:
    x1, y1 = lmList[tipIds[1]][1:]
    x2, y2 = lmList[tipIds[2]][1:]
    cx, cy = (x1+x2) // 2, (y1+y2) // 2
    if abs(x2-x1) <= 90 and abs(y2-y1) <= 110:
        cv2.circle(img, (cx, cy), 40, (0, 255, 0), cv2.FILLED)
```

3.3.2 Bounding box

I bounding box all'interno dell'applicazione sono fondamentali per scatenare l'evento di click: quando l'indice e il medio entrano in quest'ultimo, avviene un cambio di stato.

Riportiamo di seguito un esempio di implementazione:

```
x1, y1 = lmList[tipIds[1]][1:]
x2, y2 = lmList[tipIds[2]][1:]
cx, cy = (x1 + x2) // 2, (y1 + y2) // 2
# controllo indice alzato
if fingers[0] and fingers[1]:
    # indice su riquadro rosso=Start
    if startPointButtonContinue[1] < cy < endPointButtonContinue[1]:
        if startPointButtonContinue[0] < cx < endPointButtonContinue[0]:
            cTime = 0
            pTime = 0
            tTime = 0
            cState = State._Menu
```

3.3.3 Inserimento di immagini

All'occorrenza sono state inserite delle chiamate a funzione per l'aggiunta delle immagini utili all'interfaccia corrente. Dentro di esse sono stati utilizzati il metodo di *addWeighted(…)* per le figure di background e il metodo di slice del frame per la sostituzione di porzioni di quest'ultimo. A titolo di esempio:

```
def add_immagineTutorial(frame, frameCopy):
    cv2.addWeighted(frame, 0, tutorialImg, 1, 0.0, frame)
    frame[0:180, 0:320] = cv2.resize(frameCopy, (320, 180))
    return frame
```

```
def add_immagineResult(frame, gameResult, pcChoice, playerChoice):
    if gameResult == 0:
        cv2.addWeighted(frame, 0, resultPareggioImg, 1, 0.0, frame)
    elif gameResult == 1:
        cv2.addWeighted(frame, 0, resultVittoriaImg, 1, 0.0, frame)
    elif gameResult == 2:
        cv2.addWeighted(frame, 0, resultSconfittaImg, 1, 0.0, frame)
    elif gameResult == 3:
        cv2.addWeighted(frame, 0, resultMossaNonValidaImg, 1, 0.0, frame)

    if pcChoice == 0:
        pc = sassoImg
    elif pcChoice == 1:
        pc = cartaImg
    elif pcChoice == 2:
        pc = forbiceImg
```

Le immagini sono state precaricate all'inizio del programma e successivamente riscalate (*resize()*) per la finestra prescelta. In base al valore assegnato alla variabile *gameResult* , che si basa sulla relazione tra le scelte intraprese dal computer e dall'utente (valori 0, 1, 2, 3 rispettivamente per pareggio, vittoria, sconfitta, non validità della mossa dell'utente), è stata infine utilizzata una specifica immagine, all'interno dello stato di Result, per entrambi i comportamenti assunti.

3.3.4 Timers

Abbiamo importato una libreria time per effettuare specifici cambiamenti di stato. Ad ogni aggiornamento di frame è stato preso il tempo corrente di sistema per gestire lo scorrere dello stesso relativo dell'applicazione. Vengono aggiornate le variabili *pTime, cTime e tTime*.

```
cTime = time.time()
if pTime == 0:
    pTime = cTime
tTime = tTime + (cTime - pTime)
pTime = cTime
```

4 Come giocare

All'avvio dell'applicazione verrà mostrata la schermata di tutorial dove l'utente potrà trovare le indicazioni sulle varie regole di gioco. Da qui, per passare al menu principale, occorrerà cliccare sul pulsante **Inizia il gioco** spostando il cursore all'interno del relativo bounding box (paragrafo 3.3.2).

Dalla schermata di menu l'utente, cliccando sui relativi bottoni, avrà la possibilità di iniziare la fase di gioco, consultare nuovamente il tutorial oppure chiudere l'applicazione.

Cliccato il pulsante **Inizia il gioco**, si passerà automaticamente alla schermata di gioco e partirà un timer (paragrafo 3.3.4). In questo intervallo di tempo l'utente, avendo cura di posizionare la mano correttamente davanti alla webcam, avrà la possibilità di scegliere la sua mossa.

Allo scadere del timer verrà scattato uno screenshot che mostrerà la mossa scelta dall'utente. Questo sarà visualizzato nella schermata di risultato insieme ad un'immagine (paragrafo 3.3.3) che rappresenta la mossa eseguita dal computer. Una volta decretato il vincitore della sfida (paragrafo 3.3.1), si ritornerà automaticamente alla schermata di menu dove sarà possibile iniziare una nuova partita.

5 Conclusione

Un'altra possibile implementazione dell'applicazione potrebbe prevedere il duello tra due utilizzatori umani, in quanto il framework di Mediapipe permette il riconoscimento multiplo di più mani.

Inoltre, sarebbe anche possibile rendere la scelta effettuata dal computer sfruttando l'utilizzo di una rete neurale, in modo da gestire un'intelligenza artificiale che possa apprendere le mosse effettuate dall'utente nelle varie sessioni di gioco.

Nel nostro caso Mediapipe offre in fase di inferenza un frame rate non eccezionale, un'idea sarebbe impiegare un classificatore molto più leggero così da ottenere una minor latenza.