Expósito Martínez Javier
Clemente Infantes Pablo

cARds game

Índice:

Descripción general de la aplicación

En un principio la aplicación consiste en crear una nueva experiencia de inmersión de juego al usuario, en algún juego de cartas en el cuál mediante nuestra aplicación el usuario sería capaz de ver tridimensionalmente los modelos de las cartas que está usando e incluso la interacción de estas con el medio y otras cartas cercanas.

También es probable que nuestra aplicación se encargue de la lógica del juego mediante uso de otras cartas, reconocimiento de voz u acciones que representen ataque, defensa, y un par más de opciones de manera que el usuario, no solo observa visualmente el modelo sino que interactúa en tiempo real con la aplicación.

Justificación/Motivación

Problemas a resolver

Uno de los problemas que resuelve sería la falta de dinamismo y realidad en el juego de cartas en la versión real, respecto a la serie en la que está inspirado, es decir, no es la misma experiencia jugar a un juego normal de cartas basado en la serie con respecto a jugar a dicho mismo juego pero con una experiencia que se acerca mucho a como es en dicha serie, es decir se produce una aproximación a cómo es en realidad el juego, que cuando eres un fan, se agradece.

Otro problema que resuelve y quizás el más importante, sería gracias a que la aplicación lleva la lógica del juego y por tanto, el jugador no tendrá que aprenderse mecánicas, cartas y jugadas que dependiendo de qué tan complejo sea el juego puede ser tedioso y alejar a los usuarios del juego.

Competencia

Investigando un poco encontramos por ejemplo la empresa Tilt Five que está desarrollando un tablero que con ayuda de realidad aumentada, se crea un juego virtual con personajes, construcciones y demás objetos necesarios, sin embargo, no vemos esto como una competencia directa que nos cause problemas en el mercado, debido a que nuestro juego es diferente y enfocado a un público que puede que no esté tan interesado en la propuesta de esta empresa.

Investigando un poco más, resulta que hay un prototipo similar al nuestro sobre pokemon, pero actualmente solo ha sido desarrollado por un fan, además que nuestro juego puede captar mayor atención debido a la interacción de las cartas, no solo con el resto, como se hace en el prototipo de pokemon sino también con el tablero.

En tercer lugar tenemos como principal competencia directa a BigAR, que sobre 2018, estaba ya desarrollando realidad aumentada para el juego de cartas Magic: The Gathering, sin embargo, hasta donde se ha visto, sólo aporta información adicional o información sobre la carta y en vez de una simple imagen, un video o cinemática pero no es un modelo tridimensional del personaje lo cuál llama más la atención ni tampoco parece ser que la app controle la lógica del juego, por tanto pensamos que nos podemos imponer en el mercado.

Opiniones de expertos/clientes acerca de la idea

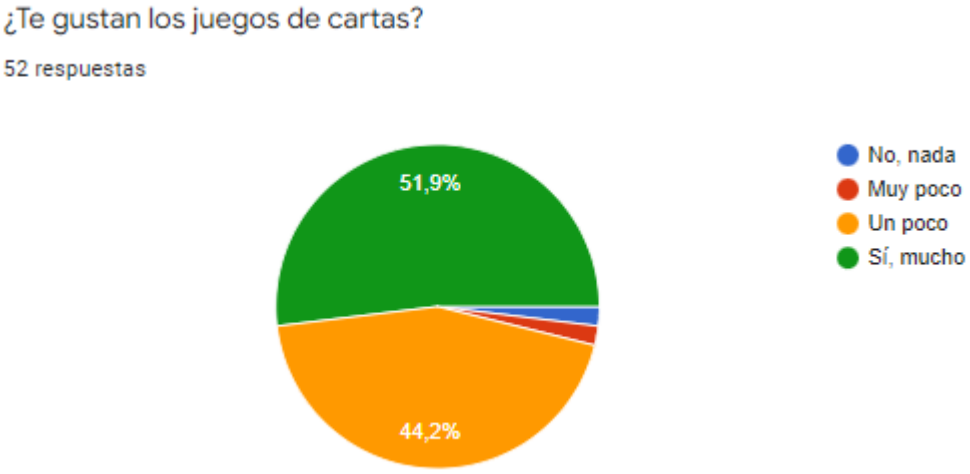
Preguntas encuesta:

1. ¿Conoces la realidad aumentada?

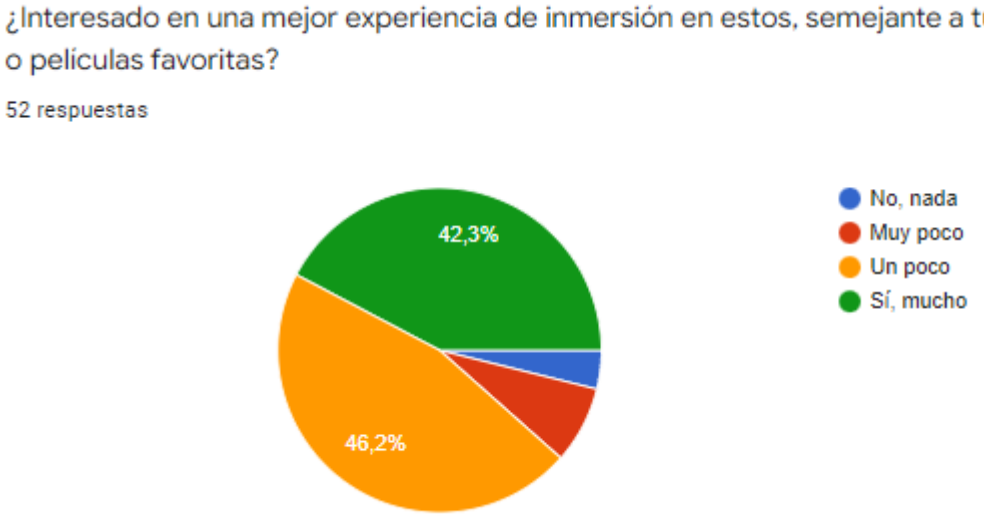
2. ¿Tienes algún tipo de interés por la AR o te llama la atención?

3. ¿Crees que el uso de AR es simple o complejo?

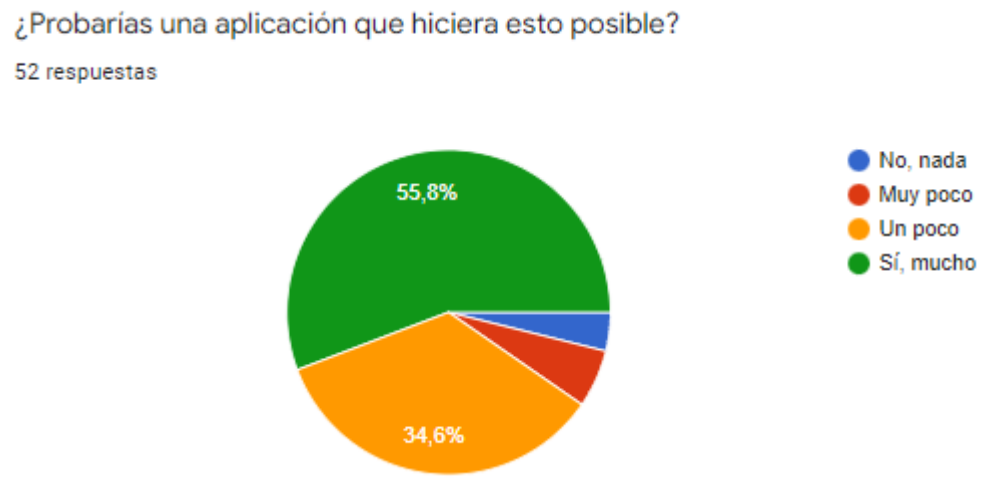
4. ¿Te gustan los juegos de cartas?



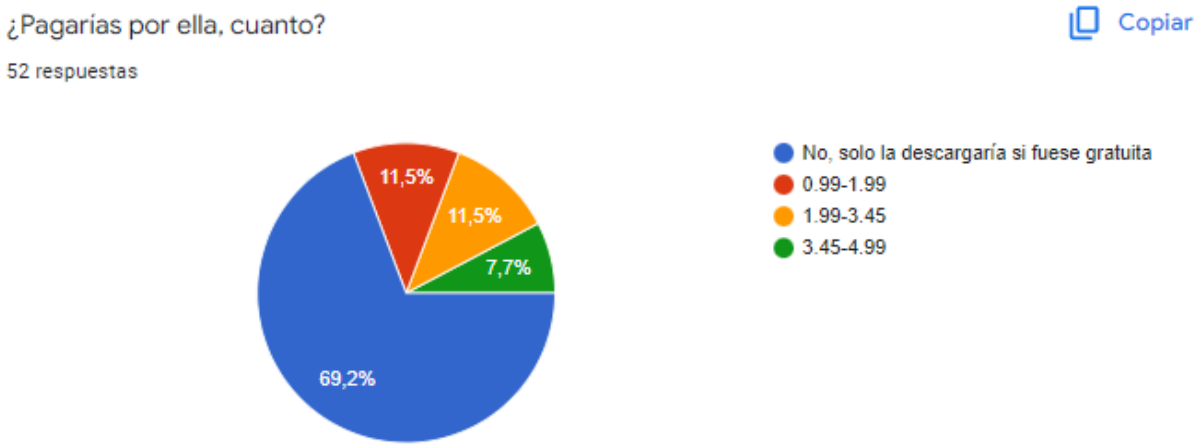
5. ¿Interesado en una mejor experiencia de inmersión en estos, semejante a tus series o películas favoritas?*Foto junto con bibliografía*



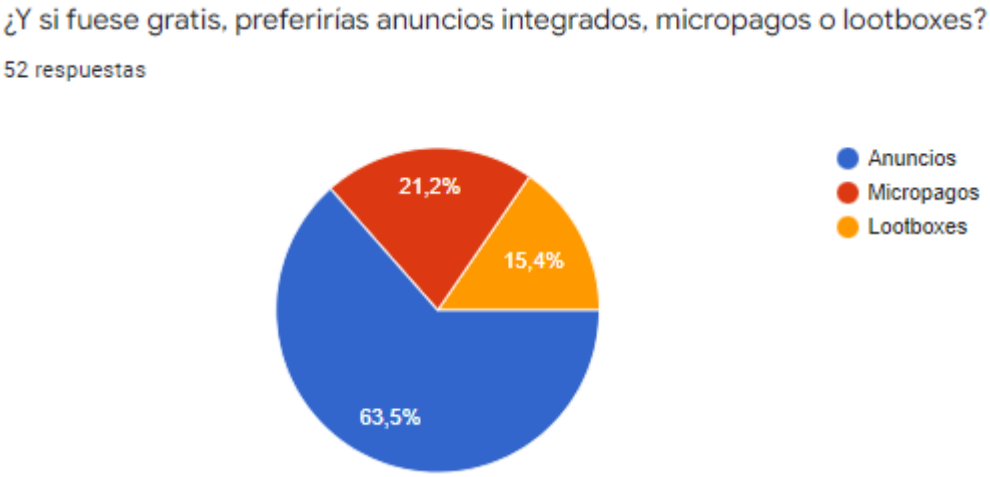
6. ¿Probarías una aplicación que hiciera esto posible?



7. ¿Pagarías por ella, cuanto?



8. ¿Y si fuese gratis, preferirías anuncios integrados o micropagos?



Conclusiones de estudio de mercado:

Hay un gran desconocimiento respecto a la Realidad Aumentada, pero mucho interés. Además la selección de un juego de cartas parece acertada, y también según las encuestas el público estaría dispuesto a probarlo aunque crean que podría ser muy complejo.

Respecto al pago hay más predisposición a que sea gratuita que de pago, aunque un 36% en caso de que fuese gratuita estaría dispuesta a pagar, luego pensamos que puede llegar a tener una buena acogida en el mercado.

Charla con expertos:

Al final dejaremos las charlas con los expertos para ampliación
Aquí recogemos algunas de las opiniones que más destacadas e interesantes nos han parecido:
Las cartas deben tener un diseño básico, no con texto o texto pequeño para que no resulte incómodo.
Para hacer más cómodo la interacción entre los usuarios con el juego, en vez de cartas de ataque o interacción por voz, hacerlo por zonas, tipo en el primer tercio zona de ataque, en el segundo tercio recuperar vida, tercer tercio defender. Y para controlar el tiempo un temporizador que de pie al combate.

Software/Hardware a utilizar

Para el hardware hemos pensado en un smartphone o tablet con Android como sistemas operativos, ya que al ser un juego de cartas, es lo que más sentido tiene.

Hemos estudiado varias herramientas para informarnos de cuál se adapta mejor a nuestras necesidades, estas son algunas de ellas:

<u>Software:</u>

- ARCore Google
- ARToolKit
- AR.js
- DroidAR
- **AR Unity3D (Vuforia)**
- **Biblioteca Opencv**

La dos últimas opciones son las que más posibilidades tienen (**AR Unity3D con Vuforia Biblioteca Opencv**).

AR Unity3D con Vuforia

Vuforia cuenta con un seguimiento robusto y rendimiento en una variedad de hardware (incluyendo dispositivos móviles y monitores de realidad mixta como las Microsoft HoloLens). La integración de Unity en Vuforia nos puede permitir crear aplicaciones y juegos de visión para Android e iOS utilizando herramientas bastante simples.

Vuforia ofrece las siguientes características:

- Reconocimiento de texto.
- Reconocimiento de Imágenes.
- Rastreo robusto. (el marcador fijado no se perderá tan fácilmente incluso cuando el dispositivo se mueva).
- Detección y rastreo simultáneo de marcadores.
- Detección Rápida de los marcadores.

Biblioteca Opencv

OpenCV es una biblioteca libre de visión artificial originalmente desarrollada por Intel. Desde que apareció se ha utilizado en una gran cantidad de aplicaciones, y hasta 2020 se la sigue mencionando como la biblioteca más popular de visión artificial.

Además puede usarse bajo licencia BSD, lo cual significa que podemos hacer uso de esta para proyectos comerciales,investigación,...

Opencv ofrece las siguientes características:

- Características 2D y 3D
- Estimación de pose de cámara
- Reconocimiento facial
- Reconocimiento de gestos
- Interacción persona-computadora
- Robótica móvil
- Comprensión de movimientos
- Reconocimiento de objetos
- Segmentación
- Estereoscopía
- Structure from motion (SFM)
- Tracking
- Realidad aumentada

Después de investigar por internet las ventajas de cada una, lo más probable es que usemos Opencv para nuestro proyecto, ya que hay bastante documentación en internet, ejemplos prácticos así como es la herramienta que estamos aprendiendo a usar en las clases.

Informe de progreso

Aprendizaje del entorno de desarrollo, bibliotecas, etc…

Para empezar, tal y como hemos visto en clase, la principal biblioteca usada ha sido **OpenCV,** cuya instalación se realiza con **pip install opencv-contrib-python** que incluye el módulo aruco (**ArUco Marker Detection**).

Una vez instalado, hemos aprendido a detectar los arUco markers, obtener y almacenar sus esquinas, sustituirlos por imágenes diferentes dependiendo de la id, a calibrar la cámara, sacar ejes de coordenadas en un arUco Marker y así dibujar líneas y por tantos objetos simples tridimensionales sobre estos, e incluso calcular distancias para un futuro desarrollo de la aplicación.

Otra librería usada es **Pyrender** con la que actualmente estamos aprendiendo cómo renderizar modelos 3D más complejos sobre los marcadores (aún estamos aprendiendo sobre esta; actualmente hemos conseguido renderizar un modelo de ejemplo, cuando detecta un marcador concreto).
Su instalación es simple, se realiza mediante **pip install pyrender**.

Por otro lado, ajeno a esto; mencionar que hemos aprendido a usar DroidCam para poder usar como cámara la de nuestro móvil y así tener una mayor versatilidad y facilidad para mostrar todo.

Prototipo de la aplicación(v1)

Main()

Creamos una variable cap que será la que active la webcam o cámara correspondiente (debido al uso de droidCam, 1 es mi webcam y 0 la del móvil, pero por defecto la webcam es 0).

Luego con la función **loadAugImages(path)** cargamos en el diccionario/map augDics las imágenes que se usarán para reemplazar los marcadores.
Posteriormente dentro del bucle while definimos la variable img que es básicamente lo que se visualiza a través de la cámara o la imagen del tablero con todos los marcadores en caso de descomentar cv2.imread(“tablero.png”) y comentar el otro img y cap.

Con la función **findArucoMarkers(img)** buscamos los marcadores que se hallen en img.
Si encuentra alguno, recorre el vector arucoFound , tal que si encuentra una id en dicho vector que se corresponda con alguna key de nuestro diccionario augDics, se procede a cambiar el marcador por la imagen y traza las líneas correspondientes con la función augmentAruco y se guarda en img.

También se ha creado una variable contador que se va reseteando cada cierto tiempo, que se usará más tarde.

Posteriormente cv2.imshow para visualizarlo.

```
#### MAIN ####

def main():
    cap=cv2.VideoCapture(1)
    augDics=loadAugImages("camposTablero")
    contador=0
    while True:
        sccuess,img=cap.read()
        #img=cv2.imread("tablero.png")
        arucoFound = findArucoMarkers(img)

        # Hacemos bucle en los marcadores y los aumentamos y mostramos la imagen
        if len(arucoFound[0]) != 0:
            for bbox, id in zip(arucoFound[0], arucoFound[1]):
                if int(id) in augDics.keys():
                    img = augmentAruco(bbox, id, img, augDics[int(id)],contador)
                    contador=contador+1
                    if contador==4300:
                        contador=0


        cv2.imshow("Image",img)
        cv2.waitKey(1)


if __name__ == '__main__':
        main()
```

Función loadAugImages()

Como dijimos anteriormente esta función carga las imagenes que reemplazarán a los marcadores (usando la librería os) en nuestro diccionario augDics, que contendrá una key que es la id y la imagen a la que corresponde(para ello la imagen se llama igual que la id del marcador que queremos reemplazar).

Luego creamos una lista con los nombres de las imágenes que tenemos en el path correspondiente(myList).

Creamos augDics y recorremos el vector myList, tal que creamos una key por cada imagen haciendo un cast int del nombre(ya que se llama igual que la id del marcador),separando el nombre como tal del formato de la imagen por ejemplo: si la imagen es 23.jpg , pues se separa 23 y jpg y asignamos akey el valor 23.

Luego cargamos la imagen que reemplazará al marcador y le decimos al diccionario augDics que esa es la imagen correspondiente a dicha Key.

```python
#Funcion que carga las imagenes que van a reemplazar a los marcadores

def loadAugImages(path):
    myList=os.listdir(path)
    #print(myList)

    augDics={}
    for imgPath in myList:
        key = int(os.path.splitext(imgPath)[0]) #Saca la key(id) del nombre de la foto diviendo el nombre, pe 23.jpg
        imgAug=cv2.imread(f'{path}/{imgPath}')
        augDics[key]=imgAug
    return augDics
```

Función findArucoMarkers()

Esta función buscará los aruco markers, para ello lo primero es transformar nuestra imagen a color en una imagen a escala de grises.

Luego creamos un atributo key que determinará los atributos del diccionario de la propia biblioteca de Aruco, y posteriormente creamos dicho diccionario.

Ahora usamos las funciones propias de Aruco para detectar y crear los parámetros que luego son usados por la función detectMarkers para encontrar los marcadores que se visualizan en la imagen.
La función **detectMarkers()** devuelve un vector de vectores tal que , en la primera posición se hallan las boundboxs o límites de los marcadores visualizados, en la segunda posición sus ids y en la tercera las ids rechazadas porque no se detectaron correctamente.

Luego tenemos un if (opcional) para visualizar los límites, y retornamos los boundboxs e ids.

```python
#Creamos la funcion que va a buscar los Aruco Markers

def findArucoMarkers(img,markerSize=6,totalMarkers=250,draw=True):
    imgGray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    key=getattr(aruco,f'DICT_{markerSize}X{markerSize}_{totalMarkers}') #Con f'' lo convierto en string
    arucoDict=aruco.Dictionary_get(key)

    #Creamos los parametros
    arucoParam = aruco.DetectorParameters_create()
    bboxs,ids,rejected = aruco.detectMarkers(imgGray,arucoDict,parameters=arucoParam)

    #if draw:
        #aruco.drawDetectedMarkers(img,bboxs)

    return [bboxs, ids]
```

Función augmentAruco()

Esta función es la que reemplaza el marcador por la imagen que queremos además de trazar los límites y zonas del tablero, y renderizar el modelo 3D.

Para ello definimos las esquinas del marcador y las dimensiones/propiedades de la imagen.
Luego buscamos la homografía, creamos imgOut que es la imagen que sustituye al marcador y luego la fusionamos con img ya que si no , solo se visualizaria la imagen del marcador pero el fondo sería negro.

Se llama a la función **pintaBordesTablero()** que precisamente realiza dicha tarea y además hemos programado que cuando detecte el marcador con la id igual a 0 muestre el objeto tridimensional.
Por último la retornamos imgOut.

```
def augmentAruco(bbox, id, img, imgAug, contador,drawId=True):
    #Esquinas del marcador
    tl = (bbox[0][0][0], bbox[0][0][1])
    tr = (bbox[0][1][0], bbox[0][1][1])
    br = (bbox[0][2][0], bbox[0][2][1])
    bl = (bbox[0][3][0], bbox[0][3][1])

    #dimensiones de la foto , altura, anchura y el canal
    h, w, c = imgAug.shape

    pts1 = np.array([tl,tr,br,bl])
    pts2 = np.float32([[0,0], [w,0], [w,h], [0,h]])
    matrix, _ = cv2.findHomography(pts2, pts1)
    imgOut = cv2.warpPerspective(imgAug, matrix, (img.shape[1], img.shape[0]))# img.shape[1] anchura, img.shape[0]
    cv2.fillConvexPoly(img, pts1.astype(int), (0,0,0)) #Le pasamos la imagen de la webcam, el punto 1 como entero
    imgOut = img + imgOut

    #x=tl[0]
    #y=tl[1]
    #cv2.putText(imgOut, str(id), (int(x),int(y)), cv2.FONT_HERSHEY_SIMPLEX,1, (255, 0, 0), 2, cv2.LINE_AA)
    pintaBordesTablero(bbox,id,imgOut,contador)
    if int(id)==0:
        imgOut=renderModelo3D(imgOut,"examples/models/fuze.obj")
    return imgOut
```

Función pintaLinea()

Esta función simplemente es una simplificación de cv2.line() para entender mejor cómo se están dibujando las líneas. (pIni y pFin son los puntos que forman la línea que se está pintando.)

```
#### Funciones relacionadas con pintar ####
def pintaLinea(img,pIni,pFin,color,grosor):
    cv2.line(img,(int(pIni[0]),int(pIni[1])),(int(pFin[0]),int(pFin[1])),color,grosor)
```

Función pintaBordesTablero()

Se encarga de trazar los bordes y líneas que delimitan las zonas del tablero sabiendo que las ids de los marcadores del tablero van por orden de izquierda a derecha, de arriba a abajo. Además de dibujar el timer/cronómetro que se halla en el centro del tablero.

La función **boundingrects()** devuelve la esquina superior izquierda, su anchura (w, que usaremos para calcular la longitud inicial del timer) y su altura.
La variable contador ha sido usada para modificar tanto longitud, color y grosor del timer.
También hemos realizado la calibración de la cámara con el script que se nos proporcionó de manera que tenemos acceso a los coeficientes de la matriz y distorsión de la cámara, pudiendo aplicar la función **estimatePoseSingleMarkers()** que sirve para estimar la posición individual de cada marcador. Posteriormente obtenemos las esquinas de los marcadores que serán usadas como puntos de las líneas a trazar.

```python
#### Funciones relacionadas con el diseño grafico del tablero ####
def pintaBordesTablero(bbox,id,imgOut,contador):
    x,y,w,h=cv2.boundingRect(bbox)
    color=(255,255,0)
    #15.6862 = 4000 (contador) / 255 (numero max color)
    colorTimer=(0,200-(contador/15.6862),120+(contador/15.6862))
    grosorTimer=14-(contador/500)
    grosor=3
    num=7*w-(contador/10)
    rvecs, tvecs, markerPoints = cv2.aruco.estimatePoseSingleMarkers(bbox,0.02,camara.cameraMatrix,camara.distCoeffs
    #Eliminamos el error de la matriz de valores de numpy
    (rvecs - tvecs).any()

    #Posiciones en aruco
    tl = (bbox[0][0][0], bbox[0][0][1])
    tr = (bbox[0][1][0], bbox[0][1][1])
    br = (bbox[0][2][0], bbox[0][2][1])
    bl = (bbox[0][3][0], bbox[0][3][1])

    #Bordes de la zona central para que se pinten debajo
    if(int(id)==32):
        pintaLinea(imgOut,tr,br,(0,0,0),grosor)

    if(int(id)==25):
        pintaLinea(imgOut,tl,bl,(0,0,0),grosor)
```

```python
#Borde superior
for i in range(1,9):
    if(i==int(id)):
        pintaLinea(imgOut,tl,tr,color,grosor)

#Borde inferior
for i in range(49,57):
    if(i==int(id)):
        pintaLinea(imgOut,bl,br,color,grosor)
#Alrededor del timer
for i in range(25,33):
    if(i==int(id)):
        pintaLinea(imgOut,tl,tr,color,grosor)
        pintaLinea(imgOut,bl,br,color,grosor)

#Espacios intermedios
for i in range(2,26,2):
    if(i==int(id)):
        pintaLinea(imgOut,tr,br,color,grosor)
for i in range(34,58,2):
    if(i==int(id)):
        pintaLinea(imgOut,tr,br,color,grosor)

#Borde izquierdo restante
for i in range(1,18,8):
    if(i==int(id)):
        pintaLinea(imgOut,tl,bl,color,grosor)
for i in range(33,50,8):
    if(i==int(id)):
        pintaLinea(imgOut,tl,bl,color,grosor)
```

```
#Pintar el timer
if(int(id)==25):
    pIniSup=(bbox[0][0][0], ((bbox[0][0][1]+bbox[0][1][1]+bbox[0][2][1]+bbox[0][3][1])/4)-100)
    pFinSup=(bbox[0][0][0]+num,((bbox[0][0][1]+bbox[0][1][1]+bbox[0][2][1]+bbox[0][3][1])/4)-100)
    pIniInf=(bbox[0][0][0], ((bbox[0][0][1]+bbox[0][1][1]+bbox[0][2][1]+bbox[0][3][1])/4))
    pFinInf=(bbox[0][0][0]+num,((bbox[0][0][1]+bbox[0][1][1]+bbox[0][2][1]+bbox[0][3][1])/4))
    pintaLinea(imgOut,pIniSup,pFinSup,colorTimer,int(grosorTimer))
    pintaLinea(imgOut,pIniInf,pFinInf,colorTimer,int(grosorTimer))
```

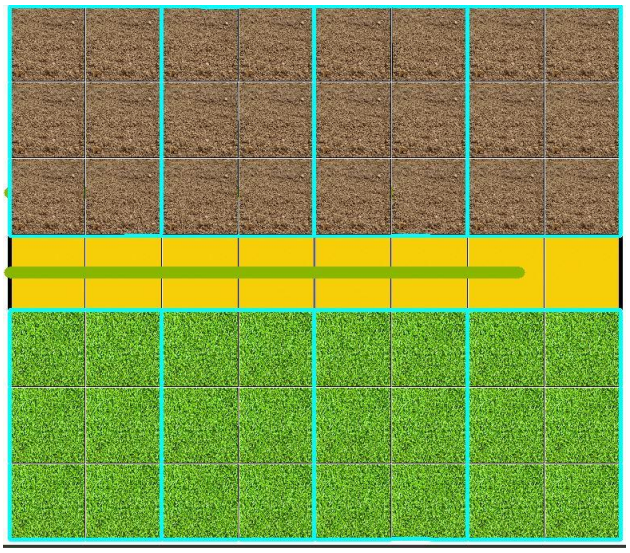
Función renderModelo3D()

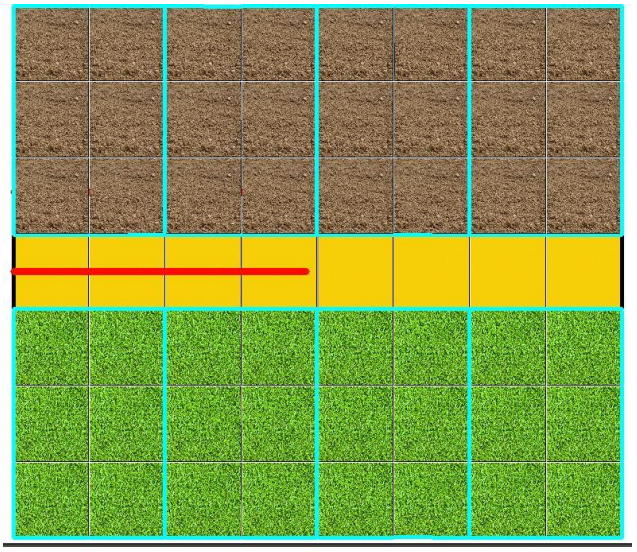
Realiza el renderizado de un modelo tridimensional. Para ello usamos la **librería pyrender**:

1. Cargamos el objeto
2. Creamos una escena y añadimos el objeto
3. Creamos una cámara y su posición
4. La añadimos a la escena
5. Hacemos lo mismo con la iluminación
6. Renderizamos y obtenemos el modelo
7. Lo sumamos a imgOut para que se vea

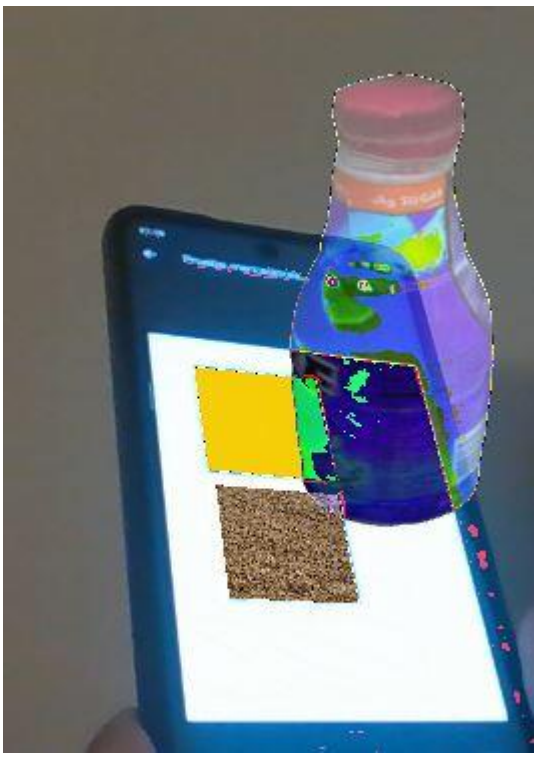
```
#Renderizado del modelo 3D

def renderModelo3D(img,path):
    obj = trimesh.load(path)
    obj_mesh = pyrender.Mesh.from_trimesh(obj)
    scene = pyrender.Scene()
    scene.add(obj_mesh)
    camera = pyrender.PerspectiveCamera(yfov=np.pi / 3.0, aspectRatio=1.0)
    s = np.sqrt(2)/2
    camera_pose = np.array([[0.0, -s,   s,   0.3],[1.0,  0.0, 0.0, 0.0],[0.0,  s,   s,   0.35],[0.0,  0.0, 0.0, 1.0]
    scene.add(camera, pose=camera_pose)
    light = pyrender.SpotLight(color=np.ones(3), intensity=1.0,innerConeAngle=np.pi/16.0,outerConeAngle=np.pi/6.0)
    scene.add(light, pose=camera_pose)
    r = pyrender.OffscreenRenderer(640, 480)
    color, depth = r.render(scene)
    imgOut=img+color
    return imgOut
```

Resultados del prototipo





(La imagen de la página siguiente fue tomada cuando todavía no se habían realizado las zonas intermedias, fue tomada mediante el uso de droidCam, sirve para apreciar el timer tridimensionalmente)



Prototipo de la aplicación(v2)

En esta versión ya “definitiva” hemos añadido las siguientes funciones.

```
def estimadorDelBoard(bboxs,ids,arucoDict,img,matrixUndistort,imgAug):
    longitud = .025 #longitud del aruco en metros
    separacion = .001 #separacion entre estos
    num_marc_X=8 #Numero marcadores eje X
    num_marc_Y=7 #Numero marcadores eje Y
    id_primer=1 #id del primer marcador
    board=cv2.aruco.GridBoard_create(num_marc_X, num_marc_Y,longitud,separacion,arucoDict,id_primer)
    success, rvec, tvec = cv2.aruco.estimatePoseBoard(bboxs, ids, board, camara.cameraMatrix,camara.distCoeffs,None,None)
    if success != 0:
        distancia = int(100*np.linalg.norm(np.array(tvec),ord=2))#esto es distancia a la camara
        proyecciones, _ = cv2.projectPoints(np.array([[0.104,0.095,0.0]]), rvec,tvec,matrixUndistort, camara.distCoeffs)


        #Amos a hacer ahora el dibujo del tablero
        proyeccionestl, _ = cv2.projectPoints(np.array([[-0.01,0.19,0.0]]), rvec,tvec,matrixUndistort, camara.distCoeffs)
        proyeccionestr, _ = cv2.projectPoints(np.array([[0.208,0.19,0.0]]), rvec,tvec,matrixUndistort, camara.distCoeffs)
        proyeccionesbr, _ = cv2.projectPoints(np.array([[0.208,0.0,0.0]]), rvec,tvec,matrixUndistort, camara.distCoeffs)
        proyeccionesbl, _ = cv2.projectPoints(np.array([[-0.01,0.0,0.0]]), rvec,tvec,matrixUndistort, camara.distCoeffs)


        tl=(proyeccionestl[0][0][0],proyeccionestl[0][0][1])
        tr=(proyeccionestr[0][0][0],proyeccionestr[0][0][1])
        br=(proyeccionesbr[0][0][0],proyeccionesbr[0][0][1])
        bl=(proyeccionesbl[0][0][0],proyeccionesbl[0][0][1])
```

```
        h, w, c = imgAug.shape
        pts1 = np.array([tl,tr,br,bl])

        pts2 = np.float32([[0,0], [w,0], [w,h], [0,h]])
        matrix, _ = cv2.findHomography(pts2, pts1)
        imgOut = cv2.warpPerspective(imgAug, matrix, (img.shape[1], img.shape[0]))# img.shape[1] anchura, img.shape[0] altu
        cv2.fillConvexPoly(img, pts1.astype(int), (0,0,0)) #Le pasamos la imagen de la webcam, el punto 1 como entero y el
        imgOut = img + imgOut
        cv2.putText(imgOut, str(distancia)+"cm",((proyecciones[0][0][0]).astype(int),proyecciones[0][0][1].astype(int)),0,
        cv2.drawFrameAxes(imgOut, matrixUndistort, camara.distCoeffs, rvec, tvec,longitud*10 )
```

Función estimadorDelBoard()

Con esta función definimos el tamaño del tablero y dibujamos en el tablero usando **estimatePoseBoard()**, el cual crea un eje de coordenadas cuyo **origen** es la esquina superior izquierda si el tablero es horizontal, por tanto en nuestro caso es la **esquina inferior izquierda** . Una vez hecho esto con la función **projectPoints()** y sabiendo los vectores de traslación y rotación, podemos **asignar los puntos que serán las esquinas de la imagen que se dibuja en el tablero**, en vez de usar las esquinas de un marcador.

Además con **estimatePoseBoard()** adquirimos la ventaja de que **aunque un marcador sea tapado, el tablero seguirá siendo visible**. Las últimas dos líneas son opcionales para ver la distancia respecto a la cámara y el eje de coordenadas.
En resumen, hacemos lo que hace augmentAruco(), solamente que en vez de usar estimatePoseSingleMarkers usamos estimatePoseBoard().

Nueva función augmentAruco()

La función augmentAruco() ha sido redefinida de la siguiente forma:

```
def augmentAruco(bbox, id, img, imgAug,matrixUndistort):

    #dimensiones de la foto , altura, anchura y el canal
    h, w, c = imgAug.shape
    #Dimensiones del tablero IRL

    rvecs, tvecs, markerPoints = cv2.aruco.estimatePoseSingleMarkers(bbox,0.02,camara.cameraMatrix,camara.distCoeffs)
    if rvecs is not None:
        for i in range(len(rvecs)):
            distancia = int(100*np.linalg.norm(np.array(tvecs[i]),ord=2))#esto es distancia a la camara
            proyecciones, _ = cv2.projectPoints(np.array([[0.0,0.0,0.0]]), rvecs[i],tvecs[i],matrixUndistort, camara.distCo
            proyeccionestl, _ = cv2.projectPoints(np.array([[-0.01,0.01,0.0]]), rvecs[i],tvecs[i],matrixUndistort, camara.d
            proyeccionestr, _ = cv2.projectPoints(np.array([[0.01,0.01,0.0]]), rvecs[i],tvecs[i],matrixUndistort, camara.di
            proyeccionesbr, _ = cv2.projectPoints(np.array([[0.01,-0.01,0.0]]), rvecs[i],tvecs[i],matrixUndistort, camara.d
            proyeccionesbl, _ = cv2.projectPoints(np.array([[-0.01,-0.01,0.0]]), rvecs[i],tvecs[i],matrixUndistort, camara.

            #cv2.putText(img, str(id),((proyeccionestl[0][0][0]).astype(int),proyeccionestl[0][0][1].astype(int)),0, 1, (0,

            tl=((proyeccionestl[0][0][0]),proyeccionestl[0][0][1])
            tr=((proyeccionestr[0][0][0]),proyeccionestr[0][0][1])
            br=((proyeccionesbr[0][0][0]),proyeccionesbr[0][0][1])
            bl=((proyeccionesbl[0][0][0]),proyeccionesbl[0][0][1])

            pts1 = np.array([tl,tr,br,bl])

            pts2 = np.float32([[0,0], [w,0], [w,h], [0,h]])
            matrix, _ = cv2.findHomography(pts2, pts1)
            imgOut = cv2.warpPerspective(imgAug, matrix, (img.shape[1], img.shape[0]))# img.shape[1] anchura, img.shape[0]
            cv2.fillConvexPoly(img, pts1.astype(int), (0,0,0)) #Le pasamos la imagen de la webcam, el punto 1 como entero y
            imgOut = img + imgOut
            #cv2.putText(imgOut, str(distancia)+"cm",((proyecciones[0][0][0]).astype(int),proyecciones[0][0][1].astype(int)
            #cv2.drawFrameAxes(imgOut, matrixUndistort, camara.distCoeffs, rvecs[i], tvecs[i],0.015)
            #cv2.putText(imgOut, str(id),((proyecciones[0][0][0]).astype(int),proyecciones[0][0][1].astype(int)),0, 1, (0,2
```

Como hemos mencionado anteriormente, el proceso es similar que con la anterior función solo que ahora el **origen del eje de coordenadas es el centro del marcador debido a usar estimatePoseSingleMarkers()**. En caso de que hubiese algún error o no detecte el board o marcador respectivamente imgOut = img , y los vectores son nulos para evitar que el programa falle.

Ambas funciones retornan sus vectores de traslación,rotación y la imagen resultado de dibujar el tablero y el terreno del marcador.

```
else:
    imgOut=img
    tvecs=None
    rvecs=None

return imgOut,tvecs,rvecs
```

Funciones CaldulodDistancia() y CalculoCoordenadas()
Con la primera función calculamos la distancia entre el tablero y las respectivas cartas usando sus vectores de traslación tomando como referencia, punto de origen, la cámara (aunque a efectos prácticos no ha sido usada).

La segunda función devuelve las coordenadas de la carta respecto al origen del tablero (esquina inferior izquierda) para así saber en qué zona se halla la carta. El resto de parámetros extras son usados para mostrar una línea entre el origen de tablero y la carta (opcional, está comentado).

```
def CalculodDistancia(tvecTablero,tvecCarta):
    x1,y1,z1=-tvecTablero[0][0],-tvecTablero[1][0],-tvecTablero[2][0]
    x2,y2,z2=-tvecCarta[0][0][0],-tvecCarta[0][0][1],-tvecCarta[0][0][2]

    distanciax,distanciay,distanciaz=x1-x2,y1-y2,z1-z2
    distancia=np.sqrt(distanciax * distanciax + distanciay * distanciay + distanciaz * distanciaz)
    print(distancia)
    return distancia


def CalculoCoordenadas(img,tvecTablero,tvecCarta,rvecCarta,rvecTablero,matrixUndistort):
    x1,y1,z1=-tvecTablero[0][0],-tvecTablero[1][0],-tvecTablero[2][0]
    x2,y2,z2=-tvecCarta[0][0][0],-tvecCarta[0][0][1],-tvecCarta[0][0][2]
    dx,dy,dz=x2-x1,y2-y1,z2-z1
    #print("[",dx,",",dy,",",dz,"]")
```

El nuevo main()

Ahora la función main() es mucho más amplia:

```python
#MAIN
def main():
    cap=cv2.VideoCapture(0)
    augDics=loadAugImages("camposTablero")
    contador=0
    if cap.isOpened():
        hframe = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
        wframe = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        matrix, roi = cv2.getOptimalNewCameraMatrix(camara.cameraMatrix, camara.distCoeffs, (wframe,hframe), 1, (wframe,hfr
        roi_x, roi_y, roi_w, roi_h = roi
        color_steve=[0,0,255]
        color_kid=[0,0,255]
        kid=-1
        steve=-1
        ganador=-1
        while True:
            ret,img=cap.read()
            if ret:
                #img=cv2.imread("tablero.png")
                #img = cv2.undistort(img, camara.cameraMatrix, camara.distCoeffs, None, matrix)
                #img = img[roi_y: roi_y+roi_h,roi_x: roi_x+roi_w]


                arucoFoundTablero = findArucoMarkers(img)
                arucoFoundCartas=findArucoMarkers(img,markerSize=5,totalMarkers=250,draw=True)
                #Nuevo

                img,tvecTablero,rvecTablero=estimadorDelBoard(arucoFoundTablero[0],arucoFoundTablero[1],arucoFoundTablero[2
```

En esta primera parte hemos añadido:

- Un contador que se usará para disminuir la longitud, cambiar el color y anchura del timer
- Las operaciones necesarias para obtener el frame rectificado sin la distorsión de la cámara, aunque esta comentado debido a que por problemas con la calibración del ordenador usado no es usado.
- Los colores para saber el resultado del combate.

Ahora hay dos foundArucoMarkers() (que ahora también retorna el diccionario aruco), uno para el tablero y otro para las cartas ya que usan diferentes.
Llamamos a la función estimadorDelBoard()

```python
if len(arucoFoundCartas[0]) != 0:
    for bbox, id in zip(arucoFoundCartas[0], arucoFoundCartas[1]):
        if int(id) in augDics.keys():
            img,tvecCarta,rvecCarta = augmentAruco(bbox, id, img, augDics[int(id)],matrix)
        else:
            img,tvecCarta,rvecCarta = augmentAruco(bbox, id, img, augDics[1],matrix)

        idKi_adi=int(id) in range(0,6)

        if tvecTablero is None:
            if tvecCarta is not None and idKi_adi:
                graphic.Ki_Adi(img,tvecCarta,rvecCarta,matrix,kid)
            else:
                graphic.steve_ataque(img,tvecCarta,rvecCarta,matrix,steve)
        else:
```

Aquí tenemos para poder ver los muñecos aunque no veamos el tablero, ya que si vemos el tablero la interacción será distinta.

Básicamente hemos programado que si me detecta al menos una bbox de un marcador, llame a la función augmentAruco(), con un if else por si la id del aruco, no coincide con ninguna imagen, que ponga por defecto una para que no haya problema.

Establecemos con un booleano que los marcadores con id del 0 al 6 pertenecen a un personaje y el resto al otro, luego ya solo queda decir que si no detecta el tablero pero si una carta, que dependiendo de esas ids pinte un personaje u otro. En caso de que si detecta el tablero:

```
if  tvecCarta is not None and tvecTablero is not None and idKi_adi:
    x,y=CalculoCoordenadas(img,tvecTablero,tvecCarta,rvecCarta,rvecTablero,matrix)
    contador=contador+0.001
    graphic.estado(color_kid,img,rvecCarta,tvecCarta,matrix)
    if contador > 0.2:
        ganador=graphic.combate(kid,steve)
        print(ganador)
        if ganador == 2:
            color_kid=[0,255,0]
        if ganador == 1:
            color_kid=[0,0,255]
        if ganador == 3:
            color_kid=[255,0,0]
        contador=0
```

Si detecta el tablero, las cartas, y pertenece al personaje correspondiente (Kid), calcula las coordenadas de donde está la carta, se pinta la **barra que indica el resultado del combate con graphic.estado()** y cuando el timer se agote, realiza el combate y establece el ganador cambiando los colores de las barras de estado, y el contador del timer se resetea.

```
if(x > 0.0 and x < 0.05 and y > 0.05 and y < 0.075):
    postl=[0.02,0.09]
    postr=[0.03,0.09]
    posbr=[0.03,0.08]
    posbl=[0.02,0.08]
    kid=graphic.Ki_Adi_ataque(img,tvecCarta,rvecCarta,matrix,kid)
    #graphic.dibujarespada(postl, postr, posbr, posbl,0,150,img,rvecTablero, tvecTablero, matrix,(0,0,255))
if(x > 0.0 and x < 0.05 and y > 0.025 and y < 0.05):
    postl=[0.005,0.065]
    postr=[0.045,0.065]
    posbr=[0.045,0.055]
    posbl=[0.005,0.055]
    kid=graphic.Ki_Adi_defensa(img,tvecCarta,rvecCarta,matrix,kid)
    #graphic.cubo(postl, postr, posbr, posbl,50,150,img,rvecTablero, tvecTablero, matrix,(0,255,0))
if(x > 0.0 and x < 0.05 and y > 0.0 and y < 0.025):
    postl=[0.005,0.015]
    postr=[0.045,0.015]
    posbr=[0.045,0.005]
    posbl=[0.005,0.005]
    kid=graphic.Ki_Adi(img,tvecCarta,rvecCarta,matrix,kid)
    #graphic.piramide(postl, postr, posbr, posbl,50,img,rvecTablero, tvecTablero, matrix,(255,0,0))
```

Aquí resumo lo que está repetido 4 veces, dependiendo de la parte del mapa en el que esté, adelante, medio o atrás, estará en modo ataque, defensa o espera respectivamente, la parte comentada era la que dibujaba una espada, una pared o una pirámide, para saber en qué zona estaba, aunque como hicimos diferentes modelos para los diferentes estado de los personajes, ahora la interacción es mucho más natural.

```
if  tvecCarta is not None and tvecTablero is not None and not idKi_adi:
    x,y=CalculoCoordenadas(img,tvecTablero,tvecCarta,rvecCarta,rvecTablero,matrix)
    contador=contador+0.001


    graphic.estado(color_steve,img,rvecCarta,tvecCarta,matrix)

    if ganador == 2:
        color_steve=[0,0,255]
    if ganador == 1:
        color_steve=[0,255,0]
    if ganador == 3:
        color_steve=[255,0,0]
```

```
if(x > 0.0 and x < 0.05 and y > 0.1 and y < 0.125):
    postl=[0.02,0.135]
    postr=[0.03,0.135]
    posbr=[0.03,0.125]
    posbl=[0.02,0.125]
    steve=graphic.steve_ataque(img,tvecCarta,rvecCarta,matrix,steve)
    #graphic.dibujarespada(postl, postr, posbr, posbl,50,150,img,rvecTablero, tvecTablero, matrix,(0,0,255))
if(x > 0.0 and x < 0.05 and y > 0.125 and y < 0.15):
    postl=[0.005,0.155]
    postr=[0.045,0.155]
    posbr=[0.045,0.145]
    posbl=[0.005,0.145]
    steve=graphic.steve_defensa(img,tvecCarta,rvecCarta,matrix,steve)
    #graphic.cubo(postl, postr, posbr, posbl,50,150,img,rvecTablero, tvecTablero, matrix,(0,255,0))
if(x > 0.0 and x < 0.05 and y > 0.15 and y < 0.175):
    postl=[0.005,0.205]
    postr=[0.045,0.205]
    posbr=[0.045,0.195]
    posbl=[0.005,0.195]
    steve=graphic.steve(img,tvecCarta,rvecCarta,matrix,steve)
    #graphic.piramide(postl, postr, posbr, posbl,50,img,rvecTablero, tvecTablero, matrix,(255,0,0))
```

Aquí el símil para Steve.

Para todo el apartado gráfico y para que el código fuese más claro separamos la parte de dibujo de cubo, espada, pirámide, y posteriormente de los muñecos.

Función cubo(), pirámide()

```python
def cubo(postl, postr, posbr, posbl, flotar,alt_cubo,img,rvec, tvec, matrixUndistort,color):

    proyeccionestlc, _ = cv2.projectPoints(np.array([[postl[0],postl[1],0.0]]), rvec,tvec,matrixUndistort, camara.distCoeff
    proyeccionestrc, _ = cv2.projectPoints(np.array([[postr[0],postr[1],0.0]]), rvec,tvec,matrixUndistort, camara.distCoeff
    proyeccionesbrc, _ = cv2.projectPoints(np.array([[posbr[0],posbr[1],0.0]]), rvec,tvec,matrixUndistort, camara.distCoeff
    proyeccionesblc, _ = cv2.projectPoints(np.array([[posbl[0],posbl[1],0.0]]), rvec,tvec,matrixUndistort, camara.distCoeff


    tlc=((proyeccionestlc[0][0][0]),proyeccionestlc[0][0][1])
    trc=((proyeccionestrc[0][0][0]),proyeccionestrc[0][0][1])
    brc=((proyeccionesbrc[0][0][0]),proyeccionesbrc[0][0][1])
    blc=((proyeccionesblc[0][0][0]),proyeccionesblc[0][0][1])

    # Extraemos los puntos de las esquinas en coordenadas separadas

    v1, v2=tlc[0], tlc[1] #Esquina superior izq
    v3, v4=trc[0], trc[1] #Esquina superior dcha
    v5, v6=brc[0], brc[1] #Esquina inferior dcha
    v7, v8=blc[0], blc[1] #Esquina inferior iz

    #Cara inferior

    cv2.line(img,(int(v1),int(v2-flotar)),(int(v3),int(v4-flotar)),color,3) # Linea que une esquina izq con dcha sup
    cv2.line(img,(int(v5),int(v6-flotar)),(int(v7),int(v8-flotar)),color,3) # Linea que une esquina izq con dcha inf
    cv2.line(img,(int(v1),int(v2-flotar)),(int(v7),int(v8-flotar)),color,3) # Linea que une esquina izq sup con izq inf
    cv2.line(img,(int(v3),int(v4-flotar)),(int(v5),int(v6-flotar)),color,3) # Linea que une esquina dcha sup con dcha inf
```

```python
    #Cara superior

    cv2.line(img,(int(v1),int(v2-alt_cubo)),(int(v3),int(v4-alt_cubo)),color,3) # Linea que une esquina izq con dcha sup
    cv2.line(img,(int(v5),int(v6-alt_cubo)),(int(v7),int(v8-alt_cubo)),color,3) # Linea que une esquina izq con dcha inf
    cv2.line(img,(int(v1),int(v2-alt_cubo)),(int(v7),int(v8-alt_cubo)),color,3) # Linea que une esquina izq sup con izq inf
    cv2.line(img,(int(v3),int(v4-alt_cubo)),(int(v5),int(v6-alt_cubo)),color,3) # Linea que une esquina dcha sup con dcha i

    #Caras laterales


    cv2.line(img,(int(v1),int(v2-alt_cubo)),(int(v1),int(v2-flotar)),color,3) #Linea une esquina izq superior de cara sup c
    cv2.line(img,(int(v3),int(v4-alt_cubo)),(int(v3),int(v4-flotar)),color,3) #Linea une esquina dcha superior de cara sup
    cv2.line(img,(int(v5),int(v6-alt_cubo)),(int(v5),int(v6-flotar)),color,3) #Linea une esquina dcha inferior de cara sup
    cv2.line(img,(int(v7),int(v8-alt_cubo)),(int(v7),int(v8-flotar)),color,3) #Linea une esquina izq inferior de cara sup c
```

Aquí hemos visto la función que dibuja el cubo a partir de unas coordenadas y atributos y a continuación igual pero con la pirámide.

```python
def piramide(postl, postr, posbr, posbl, flotar,img,rvec, tvec, matrixUndistort,color):
    #Dibujamos la piramide

    proyeccionestlc, _ = cv2.projectPoints(np.array([[postl[0],postl[1],0.0]]), rvec,tvec,matrixUndistort, camara.distCoeff
    proyeccionestrc, _ = cv2.projectPoints(np.array([[postr[0],postr[1],0.0]]), rvec,tvec,matrixUndistort, camara.distCoeff
    proyeccionesbrc, _ = cv2.projectPoints(np.array([[posbr[0],posbr[1],0.0]]), rvec,tvec,matrixUndistort, camara.distCoeff
    proyeccionesblc, _ = cv2.projectPoints(np.array([[posbl[0],posbl[1],0.0]]), rvec,tvec,matrixUndistort, camara.distCoeff


    tlc=((proyeccionestlc[0][0][0]),proyeccionestlc[0][0][1])
    trc=((proyeccionestrc[0][0][0]),proyeccionestrc[0][0][1])
    brc=((proyeccionesbrc[0][0][0]),proyeccionesbrc[0][0][1])
    blc=((proyeccionesblc[0][0][0]),proyeccionesblc[0][0][1])

    # Extraemos los puntos de las esquinas en coordenadas separadas

    v1, v2=tlc[0], tlc[1] #Esquina superior izq
    v3, v4=trc[0], trc[1] #Esquina superior dcha
    v5, v6=brc[0], brc[1] #Esquina inferior dcha
    v7, v8=blc[0], blc[1] #Esquina inferior iz
    #Cara inferior

    cv2.line(img,(int(v1),int(v2-flotar)),(int(v3),int(v4-flotar)),color,3)
    cv2.line(img,(int(v5),int(v6-flotar)),(int(v7),int(v8-flotar)),color,3)
    cv2.line(img,(int(v1),int(v2-flotar)),(int(v7),int(v8-flotar)),color,3)
    cv2.line(img,(int(v3),int(v4-flotar)),(int(v5),int(v6-flotar)),color,3)
```

```
#Esquinas

cex1, cey1= (v1+v5)//2, (v2+v6)//2
cey2= (v4+v8)//2

cv2.line(img,(int(v1),int(v2-flotar)),(int(cex1),int(cey1-200)),color,3)
cv2.line(img,(int(v5),int(v6-flotar)),(int(cex1),int(cey1-200)),color,3)
cv2.line(img,(int(v3),int(v4-flotar)),(int(cex1),int(cey2-200)),color,3)
cv2.line(img,(int(v7),int(v8-flotar)),(int(cex1),int(cey2-200)),color,3)
```

Función dibujarespada()

Poco que explicar, se llama 3 veces a la función cubo con sus respectivas coordenadas y parámetros para dibujarlo

```
def  dibujarespada(postl, postr, posbr, posbl,flotar,alt_cubo,imgOut,rvec, tvec, matrix,color):

    color=(0,0,255) # b g r
    flotar=50
    alt_cubo=200
    cubo(postl, postr, posbr, posbl, flotar,alt_cubo,imgOut,rvec, tvec, matrix,color)

    postlizq=[postl[0]-0.01,postl[1]]
    postrizq=[postr[0]-0.01,postr[1]]
    posbrizq=[posbr[0]-0.01,posbr[1]]
    posblizq=[posbl[0]-0.01,posbl[1]]
    color=(0,0,255) # b g r
    flotar=80
    alt_cubo=100
    cubo(postlizq, postrizq, posbrizq, posblizq, flotar,alt_cubo,imgOut,rvec, tvec, matrix,color)

    postldcha=[postl[0]+0.01,postl[1]]
    postrdcha=[postr[0]+0.01,postr[1]]
    posbrdcha=[posbr[0]+0.01,posbr[1]]
    posbldcha=[posbl[0]+0.01,posbl[1]]
    color=(0,0,255) # b g r
    flotar=80
    alt_cubo=100
    cubo(postldcha, postrdcha, posbrdcha, posbldcha, flotar,alt_cubo,imgOut,rvec, tvec, matrix,color)
```

Funciones Ki_Adi(), Ki_Adi_ataque() y Ki_Adi_defensa()

Básicamente cada función pinta un modelo del personaje dependiendo de su estado, que depende de la zona en la que esté. Lo mismo haremos con el otro personaje. Al igual que la espada, nuestros modelos se llaman en varias llamadas a la función cubo() con sus respectivos atributos.

```python
def Ki_Adi(img,tvec,rvec,matrixUndistort,kid):

    kid=0

    #Pierna izq

    postl=(-0.01,0.005)
    postr=(-0.002,0.005)
    posbr=(-0.002,-0.005)
    posbl=(-0.01,-0.005)

    cubo(postl, postr, posbr, posbl, 0,18.75,img,rvec, tvec, matrixUndistort,(0, 44, 110))

    #Pierna drch
    postl=(0.002,0.005)
    postr=(0.01,0.005)
    posbr=(0.01,-0.005)
    posbl=(0.002,-0.005)

    cubo(postl, postr, posbr, posbl, 0,18.75,img,rvec, tvec, matrixUndistort,(0, 44, 110))

    #Cuerpo
    postl=(-0.01,0.005)
    postr=(0.01,0.005)
    posbr=(0.01,-0.005)
    posbl=(-0.01,-0.005)

    cubo(postl, postr, posbr, posbl, 18.75,43.75,img,rvec, tvec, matrixUndistort,(0, 44, 110))
```

```python
148     #Brazo izq
149     postl=(-0.02,0.005)
150     postr=(-0.01,0.005)
151     posbl=(-0.02,-0.005)
152     posbr=(-0.01,-0.005)
153
154     cubo(postl, postr, posbr, posbl, 31.25,43.75,img,rvec, tvec, matrixUndistort,(30, 111, 202))
155
156     #Brazo drch
157     postl=(0.01,0.005)
158     postr=(0.02,0.005)
159     posbr=(0.02,-0.005)
160     posbl=(0.01,-0.005)
161
162     cubo(postl, postr, posbr, posbl, 31.25,43.75,img,rvec, tvec, matrixUndistort,(30, 111, 202))
163
164     #Cabeza
165     postl=(-0.01,0.01)
166     postr=(0.01,0.01)
167     posbr=(0.01,-0.01)
168     posbl=(-0.01,-0.01)
169
170     cubo(postl, postr, posbr, posbl, 43.75,112.5,img,rvec, tvec, matrixUndistort,(180, 208, 252))
171     return kid
```

Aquí está el Kid_Adi básico, el que usamos para el modo espera, y también el kid=0, para saber su estado de cara al combate.

```python
def Ki_Adi_ataque(img,tvec,rvec,matrixUndistort,kid):

    kid = 2

    #Pierna izq

    postl=(-0.01,0.005)
    postr=(-0.002,0.005)
    posbr=(-0.002,-0.005)
    posbl=(-0.01,-0.005)

    cubo(postl, postr, posbr, posbl, 0,18.75,img,rvec, tvec, matrixUndistort,(0, 44, 110))

    #Pierna drch
    postl=(0.002,0.005)
    postr=(0.01,0.005)
    posbr=(0.01,-0.005)
    posbl=(0.002,-0.005)

    cubo(postl, postr, posbr, posbl, 0,18.75,img,rvec, tvec, matrixUndistort,(0, 44, 110))

    #Cuerpo
    postl=(-0.01,0.005)
    postr=(0.01,0.005)
    posbr=(0.01,-0.005)
    posbl=(-0.01,-0.005)

    cubo(postl, postr, posbr, posbl, 18.75,43.75,img,rvec, tvec, matrixUndistort,(0, 44, 110))
```

```python
    #Brazo izq
    postl=(-0.02,0.005)
    postr=(-0.01,0.005)
    posbl=(-0.02,-0.005)
    posbr=(-0.01,-0.005)

    cubo(postl, postr, posbr, posbl, 31.25,43.75,img,rvec, tvec, matrixUndistort,(30, 111, 202))

    #Brazo drch
    postl=(0.01,0.025)
    postr=(0.02,0.025)
    posbr=(0.02,0.005)
    posbl=(0.01,0.005)

    cubo(postl, postr, posbr, posbl, 31.25,46.75,img,rvec, tvec, matrixUndistort,(30, 111, 202))

    #Espada laser

    postl=(0.005,0.03)
    postr=(0.015,0.03)
    posbr=(0.015,0.025)
    posbl=(0.005,0.025)

    cubo(postl, postr, posbr, posbl, 46.75,55,img,rvec, tvec, matrixUndistort,(0,64,128))
    cubo(postl, postr, posbl, posbr, 55,90,img,rvec, tvec, matrixUndistort,(255,0,0))
```

```
    #Cabeza
    postl=(-0.01,0.01)
    postr=(0.01,0.01)
    posbr=(0.01,-0.01)
    posbl=(-0.01,-0.01)

    cubo(postl, postr, posbr, posbl, 43.75,112.5,img,rvec, tvec, matrixUndistort,(180, 208, 252))

    return kid
```

Aquí el Kid_Adi ataque igual que el básico, pero con el brazo estirado y una espada láser.

```
def Ki_Adi_defensa(img,tvec,rvec,matrixUndistort,kid):

    kid=1

    #Pierna izq

    postl=(-0.01,0.005)
    postr=(-0.002,0.005)
    posbr=(-0.002,-0.005)
    posbl=(-0.01,-0.005)

    cubo(postl, postr, posbr, posbl, 0,18.75,img,rvec, tvec, matrixUndistort,(0, 44, 110))

    #Pierna drch
    postl=(0.002,0.005)
    postr=(0.01,0.005)
    posbr=(0.01,-0.005)
    posbl=(0.002,-0.005)

    cubo(postl, postr, posbr, posbl, 0,18.75,img,rvec, tvec, matrixUndistort,(0, 44, 110))

    #Cuerpo
    postl=(-0.01,0.005)
    postr=(0.01,0.005)
    posbr=(0.01,-0.005)
    posbl=(-0.01,-0.005)
```

```
#Brazo izq
postl=(-0.02,0.005)
postr=(-0.01,0.005)
posbl=(-0.02,-0.005)
posbr=(-0.01,-0.005)

cubo(postl, postr, posbr, posbl, 31.25,43.75,img,rvec, tvec, matrixUndistort,(30, 111, 202))

#Brazo drch
postl=(0.01,0.005)
postr=(0.02,0.005)
posbr=(0.02,-0.005)
posbl=(0.01,-0.005)

cubo(postl, postr, posbr, posbl, 31.25,43.75,img,rvec, tvec, matrixUndistort,(30, 111, 202))

#Espada laser
postl=(-0.015,0.03)
postr=(0.015,0.03)
posbr=(0.015,0.025)
posbl=(-0.015,0.025)

cubo(postl, postr, posbr, posbl, 46.75,55,img,rvec, tvec, matrixUndistort,(0,64,128))
cubo(postl, postr, posbl, posbr, 46.75,55,img,rvec, tvec, matrixUndistort,(255,0,0))
```

Y el modo defensa con la espada láser en modo horizontal.

Funciones correspondientes a Steve (el otro personaje).

El Steve es igual, salvo la cabeza más pequeña y los colores.

```
#Palo

postl=(0.005,0.03)
postr=(0.015,0.03)
posbr=(0.015,0.025)
posbl=(0.005,0.025)

cubo(postl, postr, posbr, posbl, 46.75,90,img,rvec, tvec, matrixUndistort,(0,64,128))
```

En el modo ataque tiene un palo.

```
#Brazo izq
postl=(-0.01,0.005)
postr=(0.00,0.005)
posbl=(-0.02,-0.005)
posbr=(-0.01,-0.005)

cubo(postl, postr, posbr, posbl, 31.25,43.75,img,rvec, tvec, matrixUndistort,(100,125,169))

#Brazo drch
postl=(0.0,0.005)
postr=(0.01,0.005)
posbr=(0.02,-0.005)
posbl=(0.01,-0.005)

cubo(postl, postr, posbr, posbl, 31.25,43.75,img,rvec, tvec, matrixUndistort,(100,125,169))

#Palo
postl=(-0.005,0.03)
postr=(0.005,0.03)
posbr=(0.005,0.025)
posbl=(-0.005,0.025)

cubo(postl, postr, posbr, posbl, 46.75,90,img,rvec, tvec, matrixUndistort,(0,64,128))
```

Y en el defensa agarra el palo con las dos manos.

Función timer()

```
timer(img,tvec,rvec,matrix,contador):
timerpizq, _ = cv2.projectPoints(np.array([[-0.01,0.0875,0.0]]), rvec,tvec,matrix, camara.distCoeffs)
timerpdcha, _ = cv2.projectPoints(np.array([[0.2-contador,0.0875,0.0]]), rvec,tvec,matrix, camara.distCoeffs)

izqx,izqy=timerpizq[0][0][0],timerpizq[0][0][1]
dchax,dchay=timerpdcha[0][0][0], timerpdcha[0][0][1]

cv2.line(img,(int(izqx),int(izqy-50)),(int(dchax),int(dchay-50)),(0,255-contador*1250,contador*1250),int(10-contador*30))
```

Luego el timer, una línea flotante centrada en el tablero que va cambiando de color, longitud y anchura conforme quede menos tiempo.

Función estado()

```
def estado(color,img,rvec,tvec,matrix):
    timerpizq, _ = cv2.projectPoints(np.array([[-0.01,0.0,0.0]]), rvec,tvec,matrix, camara.distCoeffs)
    timerpdcha, _ = cv2.projectPoints(np.array([[0.01,0.0,0.0]]), rvec,tvec,matrix, camara.distCoeffs)

    izqx,izqy=timerpizq[0][0][0],timerpizq[0][0][1]
    dchax,dchay=timerpdcha[0][0][0], timerpdcha[0][0][1]

    cv2.line(img,(int(izqx),int(izqy-150)),(int(dchax),int(dchay-150)),color,int(5))
```

Estado, que muestra el resultado del combate. Implementación muy similar al timer. (no hemos modularizado más las funciones ahorrando código por falta de tiempo)

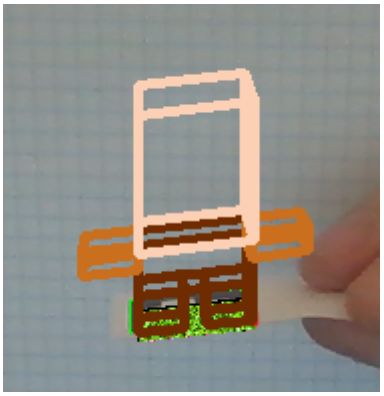
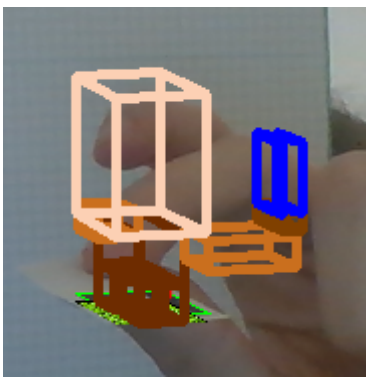
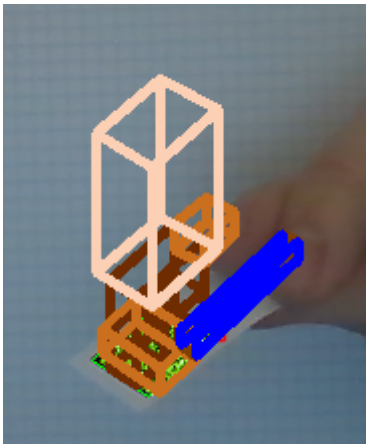
Función combate()

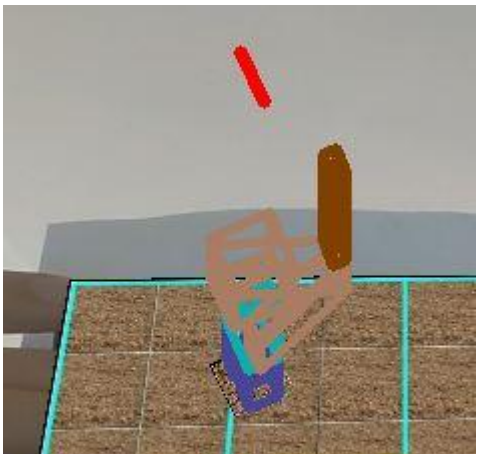
```python
def combate(kid,steve):
    #Ganador 1 = steve, ganador 2 = kid, ganador 3 = empate
    #0 Piramide, 1 Cubo, 2 Espada
    ganador=-1

    empatepapel  = (steve == 0 and kid == 0)
    empatepiedra = (steve == 1 and kid == 1)
    empatetijera = (steve == 2 and kid == 2)

    ganaStevePapel= (steve == 0 and kid == 1)
    ganaStevePiedra= (steve == 1 and kid == 2)
    ganaSteveTijera= (steve == 2 and kid == 0)

    ganaKidPapel= (kid == 0 and steve == 1)
    ganaKidPiedra= (kid == 1 and steve == 2)
    ganaKidTijera= (kid == 2 and steve == 0)

    if empatepapel or empatepiedra or empatetijera:
        ganador = 3
    elif ganaStevePapel or ganaStevePiedra or ganaSteveTijera:
        ganador = 1
    elif ganaKidPapel or ganaKidPiedra or ganaKidTijera:
        ganador = 2

    return ganador
```

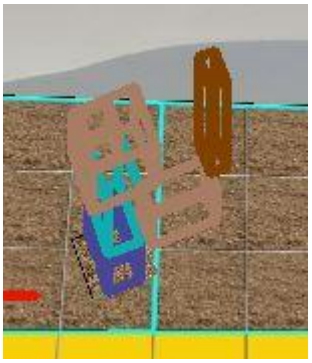
Y la lógica del combate hemos dejado como futuro avance que sea más compleja, ahora hemos dejado un simple Piedra Papel Tijera.

Resultados del prototipo

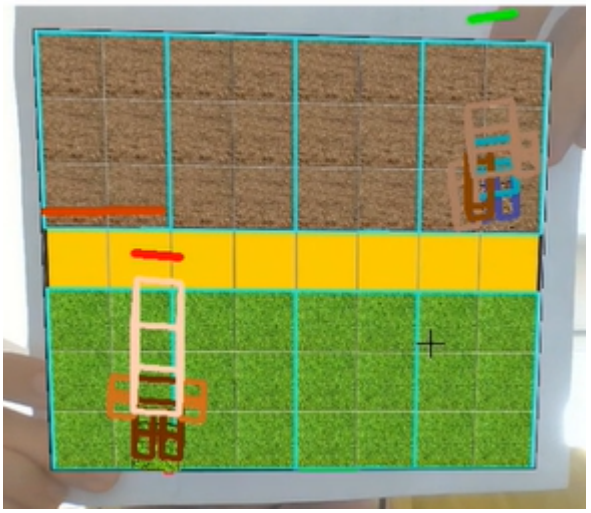
Aquí un primer plano de los diferentes modelos. Primero los modelos Kid básico, ataque y defensa, y luego igual para steve

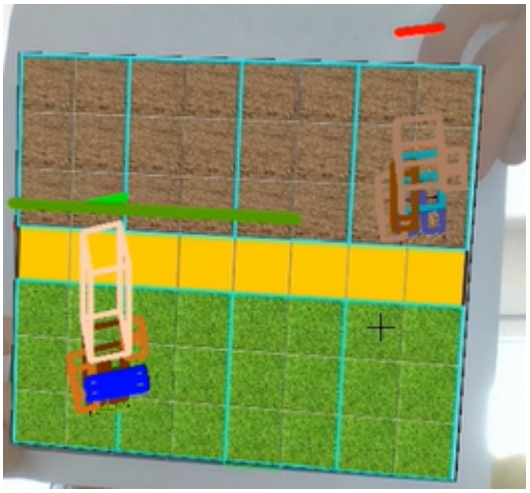




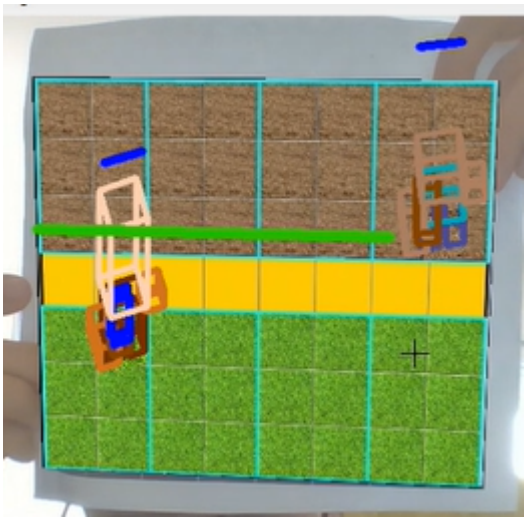


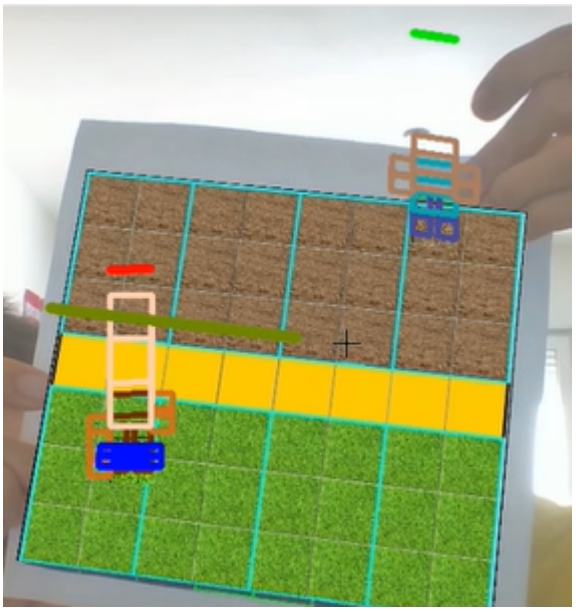


image









Descripción detallada de la aplicación

Tras las encuestas realizadas, el modelo de aplicación que vamos a usar es una aplicación con anuncios integrados.

Mediante nuestra aplicación el usuario dispondrá de marcadores/cartas para ver tridimensionalmente los modelos de las cartas que está usando e incluso la interacción de estas con el tablero y otras cartas cercanas.

Nuestra aplicación también se encargará de la lógica del juego mediante el uso de zonas específicas en el tablero: habrá una zona de cura, ataque, defensa, etc donde la carta actuará de una forma u otra dependiendo de en qué zona esté.
Las interacciones serán mostradas visualmente mediante por ejemplo una barra de vida cuando se hace daño, cuando se cura, etc…

Actualmente ya hemos creado un tablero separado en dos campos, donde los arUco markers son sustituidos por imagenes (para que dicha separación sea clara), hemos trazado líneas para delimitar los bordes y zonas del tablero(cada campo se divide en 4 zonas). En el centro se halla un timer, una línea cuyo grosor y longitud se reducen con el tiempo, además de tornarse rojo cuando queda poco tiempo.
Por otro lado, seguimos trabajando en el renderizado de modelos 3D.

Tras la versión definitiva, el renderizado de modelos 3D usando pyrender se quedó aparte, en lugar de eso hemos realizado nuestros propios modelos 3D, y la interacción con otras cartas por falta de tiempo se queda como propuesta. El resto tal y como se dijo, nuestro modelo interacciona con el tablero y el programa se encarga de la lógica del juego.

Informe Económico

Costes de desarrollo y mantenimiento

En un principio la app solo estaría disponible para sistemas Android, lo cual nos está ahorrando el coste de desarrollo para otros sistemas. Según la investigación que hemos realizado por internet el coste de desarrollo depende de varios factores:

- **Coste/Salario de programadores:** En nuestro caso este factor no influye ya que en un principio, la aplicación será desarrollada por completo entre mi compañero y yo. Aunque a largo plazo no descartamos que si triunfa la aplicación necesitemos de un mayor número de programadores que ayuden al correcto desarrollo y expansión de esta. Debemos tener en cuenta que un programador con poca experiencia puede cobrar sobre los 30€/Hora y esto puede llegar hasta los 100€/Hora si se trata de alguien especializado.

- **Acceso a los datos de la APP:** En nuestro caso nuestra aplicación necesitaría de servidores para poder almacenar tanto los datos personales de los clientes, como

usuario, contraseña, nombre, apellidos, email,... como los modelos 3D de las cartas a usar, el tablero, etc.

Por tanto supondría un coste, aunque más de mantenimiento, ya que para el desarrollo podríamos usar algún servidor local configurado por nosotros mismos para poder salir del paso durante esta fase.

- **Diseños:** En este apartado incluiríamos la necesidad de diseñar un logo para la APP, la interfaz de la aplicación, el diseño definitivo del tablero y modelado 3D de cada personaje perteneciente a una carta.

- **Aplicación nativa Android:** Como hemos dicho anteriormente, en un principio sería para Android por tanto el coste se ve reducido al no tener que hacer un desarrollo híbrido. Desarrollar una app en Android puede tener un coste entre 600€-6000€.

Reuniendo los anteriores factores, en un simulador web hemos obtenido que el coste de desarrollo de nuestra app serían 5500€ aproximadamente:

Modelo de negocio

Nuestro producto es un videojuego en realidad aumentada, cuya ventaja principal es la nula competencia en el mercado, al ser una aplicación totalmente novedosa.

Sobre al sector de la población a la que va dirigido el producto, aunque este, es un producto dirigido a las masas, debido al uso de tecnologías, en un principio podríamos considerar que va dirigido a una población joven o de mediana edad.

La forma de venta de nuestro producto, como ya se ha dicho, es descargando la app de manera gratuita con un modelo de marketing basado en el pago a diversos influencers para que promocionen dicha aplicación.

Respecto al modelo de monetización y generación de beneficios, como hemos visto en las encuestas anteriores, parece que prima una aplicación gratuita con anuncios integrados, por tanto, en un principio seguiríamos dicho modelo.
Aunque no descartamos que en un futuro, debido al éxito de la aplicación u otras causas el modelo, cambiase a ser una aplicación con compras integradas e incluso ser una aplicación de pago, ajustando el precio a las encuestas realizadas (o realizando nuevas encuestas entre los clientes).

Sobre la organización, el principal recurso que necesitamos son servidores para crear una nube donde los clientes almacenen los datos de sus personajes, diseño de tablero y diseño 3D de los personajes, animaciones, etc. Luego nuestros principales socios serán empresas que se dediquen a ofrecer estos servicios. Otra ventaja de esto es que nos permite tener una mayor flexibilidad a la hora de actuar ante cambios inesperados, como por ejemplo podría ser una venta masiva de la aplicación donde si nosotros nos hiciéramos cargo de los servidores tal vez estos colapsaría por no haber previsto una masiva subida de archivos, sin embargo de esta forma , ante esto, simplemente deberíamos pagar más por los servicios de la empresa empleados, pero del resto se debe encargar ella. Además de que ahorramos así costes en mantenimiento,seguridad,etc.

Actualmente pensamos que nuestro proyecto es viable debido al estudio de mercado hecho anteriormente con las encuestas y a la organización explicada anteriormente que dota a nuestro proyecto de gran flexibilidad y ahorro en costes en diversos sectores.

Periodo de amortización

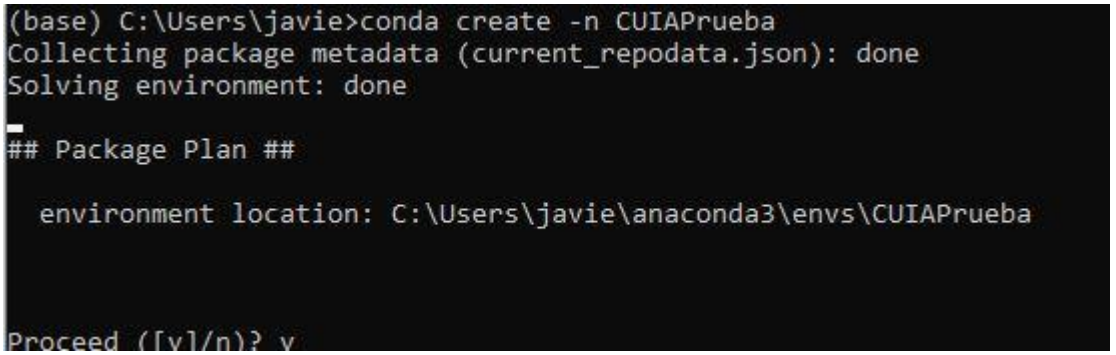
Si partimos del modelo de negocio gratuito, podemos ver que el CPM medio de anuncios en una app Android es 1.5$ que sería 1.2€, es decir de cada 1000 anuncios ganaríamos 1.2€.

Por lo tanto el periodo de amortización sería bastante largo si nuestra app no tiene una buena acogida, en cambio si nuestra app tuviese un público amplio podríamos suponer que en un año se podría amortizar el desarrollo partiendo de un coste aproximado de 5000€.

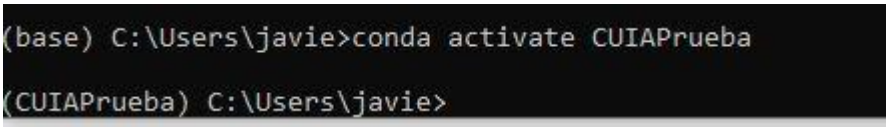
Average Mobile RPMs	
iOS Banners	$0.20 – $2.00
iOS Interstitials	$3.00 – $5.00
Android Banners	$0.15 – $1.50
Android Interstitials	$2.00 – $4.00
General Trend	Down
Source: MonetizePros Aggregation	

Preparación del entorno virtual

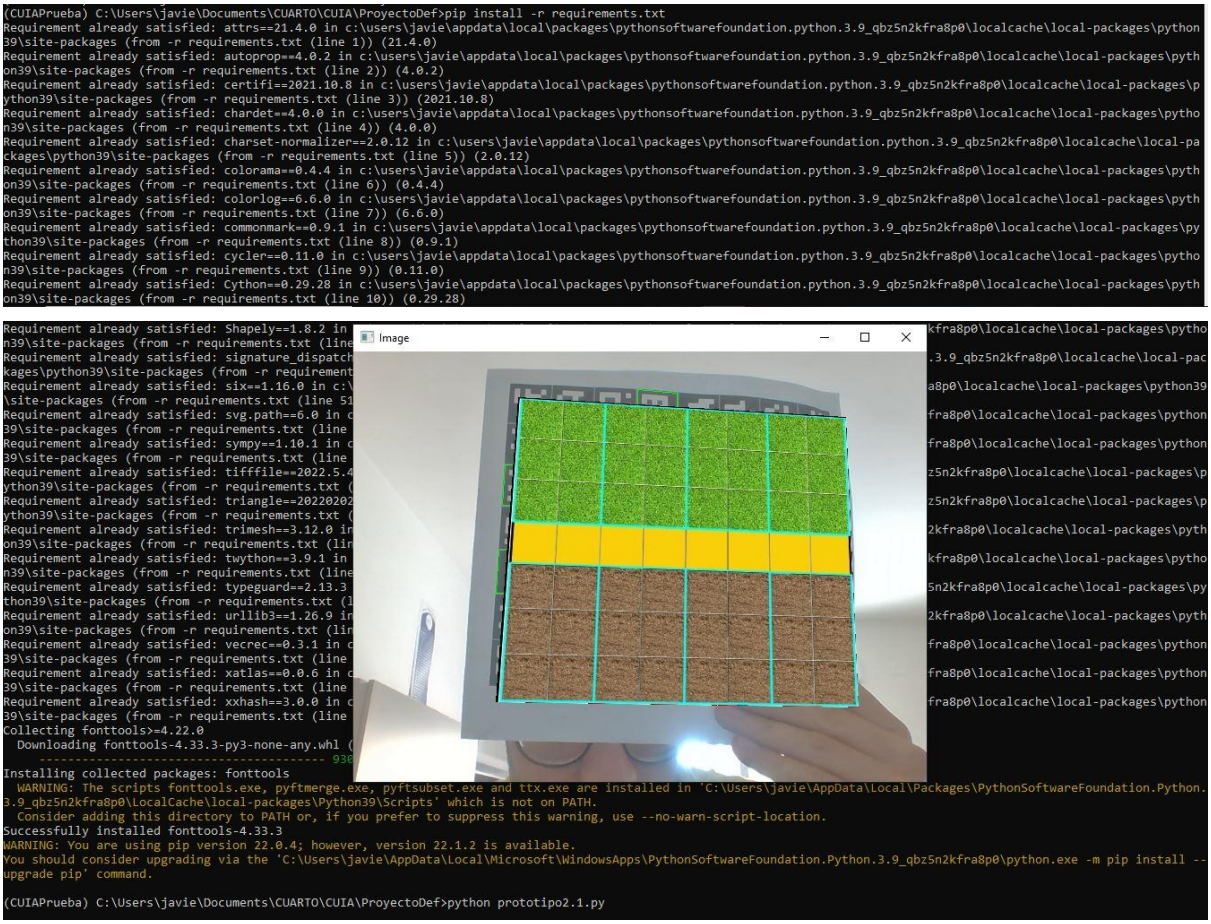
1. Crear un entorno virtual mediante **conda create -n nombre** o de cualquier otra forma.



2. Activar el entorno virtual con **conda activate nombre** o mediante el correspondiente comando de acuerdo a su entorno.



3. Descargar requeriments.txt (estará en el directorio del proyecto) y desplazarse con cd a dicho directorio.
4. Necesitamos tener instalado python, y ahora para instalar las librerías usamos el comando **pip install -r requerimets.txt**



5. Como podemos ver, ya con **python prototipo2.1py** funciona el programa.

Bibliografía

https://docs.unity3d.com/es/2018.4/Manual/vuforia-sdk-overview.html
https://es.wikipedia.org/wiki/OpenCV
https://www.xataka.com/realidad-virtual-aumentada/magic-the-gathering-realidad-aumentada-esta-empresa-quiere-unirlos-para-darle-vida-partidas
https://nintendo.pe/un-fan-de-pokemon-trae-a-la-vida-cartas-tcg-con-la-ayuda-de-la-realidad-aumentada/
https://www.tecnonews.info/noticias/los_juegos_de_mesa_se_enriquecen_con_realidad_aumentada
https://pyrender.readthedocs.io/en/latest/examples/quickstart.html
https://pyrender.readthedocs.io/en/latest/
http://amroamroamro.github.io/mexopencv/opencv_contrib/aruco_create_board_demo.html
https://docs.opencv.org/3.4/db/da9/tutorial_aruco_board_detection.html

https://clips.twitch.tv/SweetHedonisticEagleDeIlluminati-Bb_XTHFunCOyQjP3

https://clips.twitch.tv/KathishShyNarwhalJebaited-0AIlqL9NITsMvhiy

https://www.twitch.tv/videos/1440047782?t=02h17m14s

https://www.twitch.tv/videos/1440092920?t=00h07m34s

https://www.twitch.tv/videos/1440049738?t=02h29m48s

Aquí en la entrevista en llamada tuve fallos técnicos y grabé solo su voz no la mía, y se cortó a mitad de llamada (Video añadido junto a la entrega)