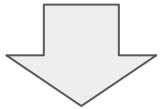
目次

1. 目的

・アイディアの元

　釣りが好きな家族の、「釣りのゲームがしたい」という言葉



・作るもの

　カメラを通して実際にジェスチャーで操作できる釣りのゲーム

・ターゲット

　釣り・ゲームが好きな人

・できること

　エンターテイメントとしてのゲームの提供

2. 実装

環境

・Windows 11

・Python 3.11.9

・Webカメラ

Pythonライブラリ

・Mediapipe

・OpenCV(cv2)

・Pillow

・Numpy

2. 実装

ファイル構成

main.py	ゲームの起動
game.py	ゲームのコア機能
camera.py	カメラの入力
pose.py	骨格検知を行う
debug.py	デバッグ用の出力
asset	画像素材、ゲームデータなど

2. 実装

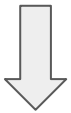
threading(標準ライブラリ)を使って

・カメラ入力

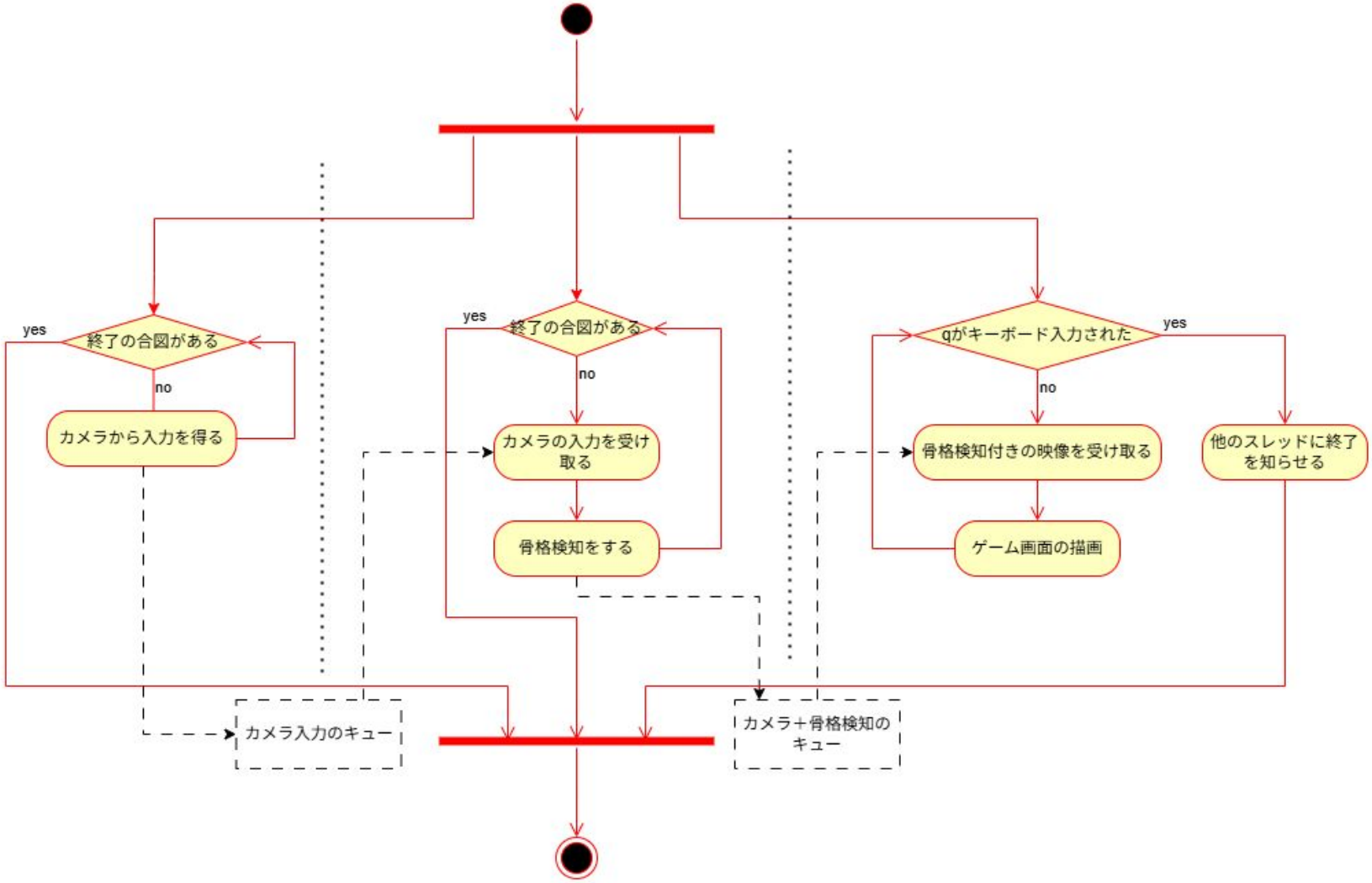
・骨格検知

・ゲームの処理と描画

の3つのスレッドに分ける



重い骨格検知の処理とゲームのメイン処理を分けて処理できるように


終了の合図がある
yes
no
カメラから入力を得る
カメラ入力のキュー
終了の合図がある
yes
no
カメラの入力を受け取る
骨格検知をする
qがキーボード入力された
yes
no
骨格検知付きの映像を受け取る
ゲーム画面の描画
他のスレッドに終了を知らせる
カメラ＋骨格検知のキュー

2. 実装

threadingの使い方

```
class Pose(threading.Thread):
    def __init__(self,camera):…

    def finishRun(self):
        self.__finish = True

    def run(self):
        # スレッドで実行するもの
        while not self.__finish: # カメラからフレームをデキューして解析してエンキュー
            if self.cam.frameQueue.empty():
                continue
            frame = self.cam.frameQueue.get()
            results = self.__getPose(frame)
            self.framePoseQueue.put([frame,results])

    def __getPose(self,img):…
```

Threadクラスを継承

runメソッド
　実行する処理

startメソッド
　スレッドの起動

joinメソッド
　スレッドをジョイン

2. 実装

cv2でのカメラ入力

```
cap = cv2.VideoCapture(camera_id)
```

カメラの取得

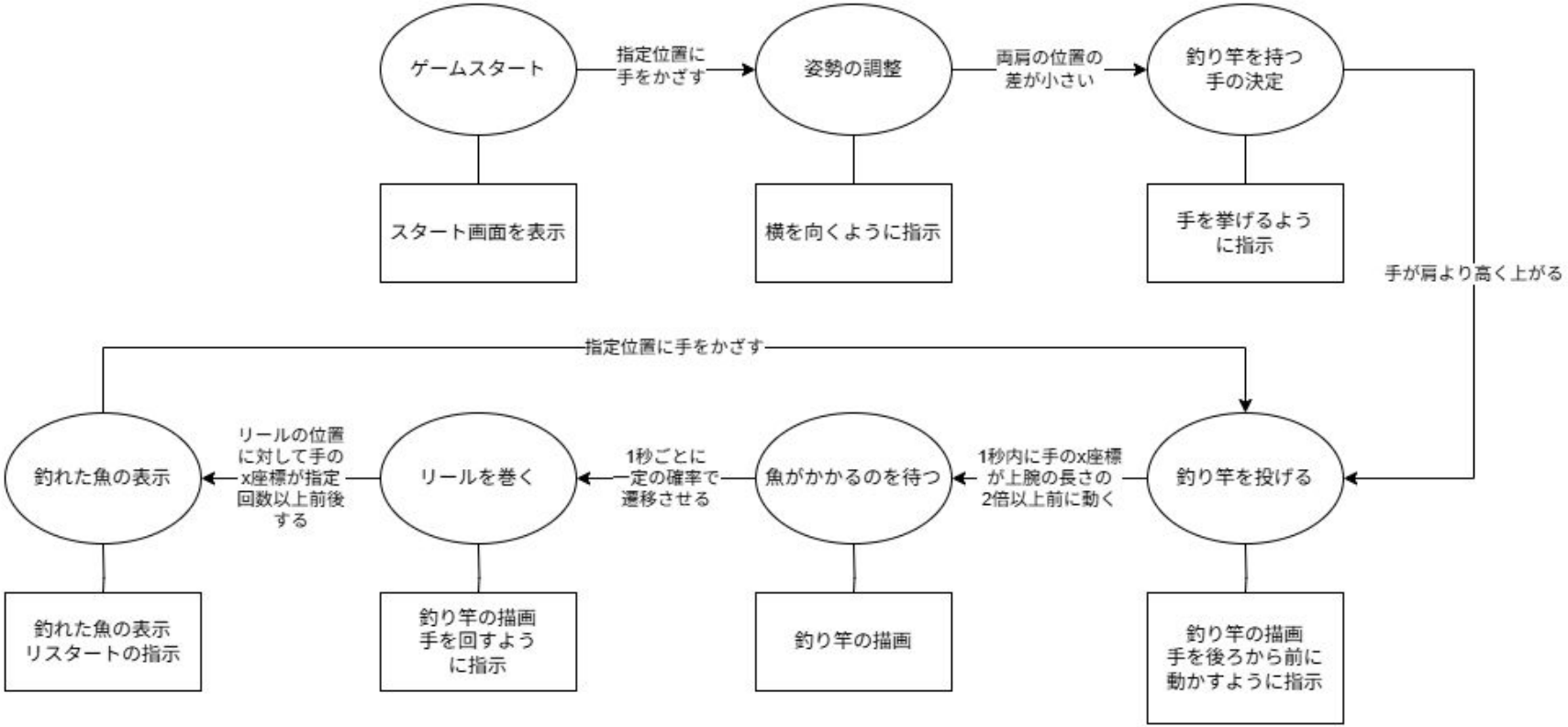
camera_id=0がデフォルトのカメラ

```
ret, frame = cap.read()
```

カメラから映像を取得

映像の取得をループし、time.sleepでループの間隔を開けることでFPSを制限

ゲームのメイン処理の遷移



2. 実装

cv2のテキスト描画は日本語を使えない

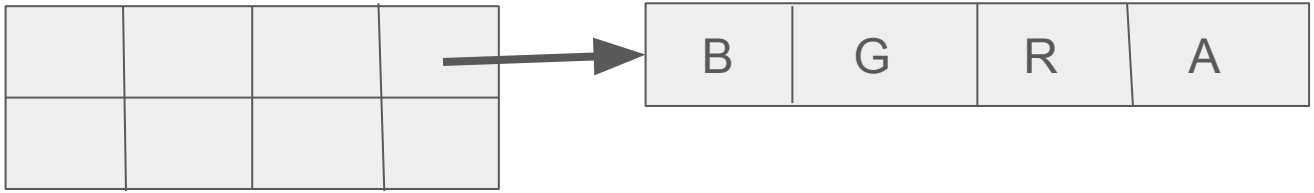
代わりにPillowを使って処理

```
def writeText(self,frame,pos,text,fontSize,color,font_path
="C:/Windows/Fonts/msgothic.ttc",anchor = "mm"):
    img_pil = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
    font = ImageFont.truetype(font_path, fontSize)
    draw = ImageDraw.Draw(img_pil)
    draw.text(pos, text, font=font, fill=color,anchor=anchor)
    frame = cv2.cvtColor(np.array(img_pil), cv2.COLOR_RGB2BGR)
    return frame
```

Pillowの画像に変換→Pillowで描画→OpenCVの画像に戻す

2. 実装

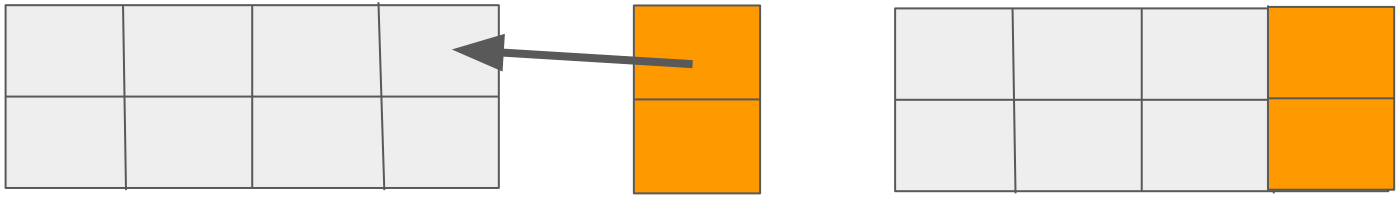
画像の合成



画像はピクセルが保存された配列で扱われる

各ピクセルは3(透明度なし)か4バイト

元の画像の配列の一部を、上に置きたい画像の配列で置き換える



2. 実装

画像の合成 - 透明度の考慮

透明な部分がある場合は直接置き換えるのではうまくいかない

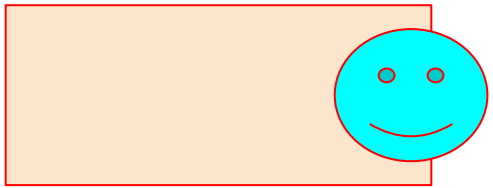
背景　透明度128(半分)

透明度から比率を考えてその分だけ色を足す

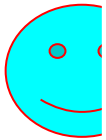
合成後の色 = (1-透明度/255)*元の色 + (透明度/255)*合成する色

2. 実装

画像の合成 - はみ出しの考慮



重ねたい画像がはみ出たとき、インデックスエラーが起きる



重ねたい画像の幅≦元の画像の幅- 重ね始めるx座標

重ねたい画像の高さ≦元の画像の高さ- 重ね始めるy座標

に制限する(切り取り)

2. 実装

画像の合成 - はみ出しの考慮2

重ねたい画像が左か上にはみ出したときに、切り取り方が代わる

画像を反転→合成の処理→もう一度反転

2. 実装

画像の反転はOpenCVの関数として用意されている

左右反転

```
img = cv2.flip(img,1)
```

上下反転

```
img = cv2.flip(img,0)
```

2. 実装

おまけ

ゲームを面白くする要素として

室内で釣りをする→魚ではなく既にできた料理が釣れたら面白いのではないか？

と思い、釣れるものを魚料理に設定(ムニエル、焼き鮭、寿司…)

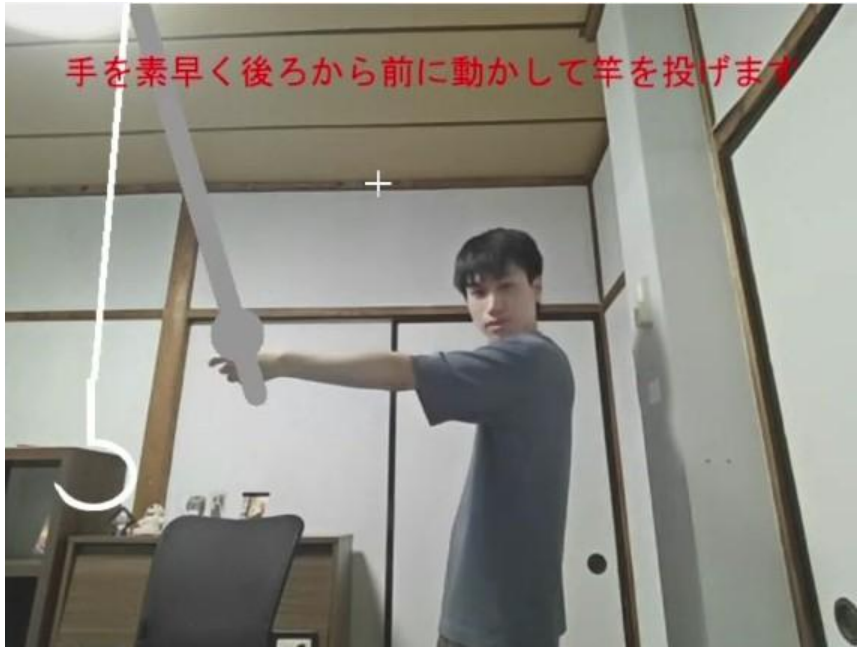
3. 結果

実行方法

```
pip install mediapipe, opencv-python, pillow, numpy

python main.py
```

3.結果



3. 結果

4. 改善点・発展

改善点

・動作が重い

骨格検知の処理が重いため、5FPSで動作している

↓ 改善案

より軽いモデルを使う

または

骨格検知の周期は変えないまま、その間の骨格の位置を補間して動作させる

4. 改善点・発展

改善点2

・モデルの精度

骨格を検知しないときがある

特に、顔が隠れると体全体の骨格を検知しなくなる

↓ 改善方法

モデルの変更

多少は補間によって耐えることもできる

4. 改善点・発展

発展

今回のゲームのような目的で体の位置を取得するのであれば、モーションキャプチャーでも可能

しかし…

モーションキャプチャーを行うには機材が必要(コストや場所の問題)

↓

骨格検知を使うことで、PCとカメラさえあれば似たようなことができる

4. 改善点・発展

発展2

趣味で作ったものですがせっかくなので置いておきます

FaceMeshをゲーム内で使用する例

「VRChat」内でのアバターの表情を操作する外部プログラム

←実際にプログラムを作って動かしたもの
目の閉じ具合、口の開き具合、眉の高さ
をトラッキング

4. 改善点・発展

VSeeFaceのように、フェイストラッキングを行えるフリーソフトなどは存在する

自作をすることでより自由な表情のカスタムが可能

手順としては

Pythonで骨格検知・データの整形

→OSCでパラメータの送信(UDP9000番ポート)

→VRChatで受け取り、パラメータにあったアニメーションの再生

OSCはUDPをベースにしたリアルタイム通信用のプロトコル

事前にUnityでアバターにアニメーションを作り、パラメータと対応させておく必要がある