Read Vision.md first to understand project goals and objectives

src folder contains the following folders

* dw\_ai -> **ai-paddle-controller**
* dw\_pong-react -> **gameboard**
* dw\_game -> **game-engine**
* dw-visualization-react -> **clocktower-visualizer**
* dw\_motion -> **human-paddle-control**
* dw\_depth\_feed -> **human-visualizer**
* dw\_visualization -> **neural-net-visualizer**
* dw\_net **-> pong-network**

**ai-paddle-controller (dw\_ai)**  is written in python consist of 3 main files:

model.py – create an AI model to training agent

ai\_driver.py – loads three models where state.game\_level (0,1,2) will assign agent (1,2,3)

ai\_subscriber.py – subscribes to mqtt topics to update paddle position and gameboard

def on\_connect(self, client, userdata, flags, rc):

print("Connected with result code " + str(rc))

client.subscribe("puck/position")

client.subscribe("player1/score")

client.subscribe("player2/score")

client.subscribe("paddle1/position")

client.subscribe("paddle2/position")

client.subscribe("game/level")

client.subscribe("game/frame")

\_\_init\_\_.py

**Gameboard (dw\_pong-react)** is written in JavaScript’s runtime Node.JS and is deployed by Node Package Manager ([NPM](https://www.freecodecamp.org/news/what-is-npm-a-node-package-manager-tutorial-for-beginners/)). The file package.json contains the dependencies to launch the browser application. A package fails to launch when one of its dependency versions is out of date.

Gameboard is implemented [using react-three-fiber](https://docs.pmnd.rs/react-three-fiber/getting-started/introduction), a React renderer for three.js. It allows you to create complex 3D scenes using React components, manage interactions and animations using React’s state management and lifecycle hooks, and participate in React’s ecosystem.

In 3DModel, add StageLeftInstruction.js and StageRightInstruction.js

In gameboard/public/models, add StageLeftInstruction.glb and StageRightInstruction.glb

In PongGame.js add ‘game/state’ subscription to change stage image

PongGame.js subscribes to the following topics

'puck/position' -> update all position

'paddle1/position’ -> player1 paddle position (human?) // top player see game\_driver.py

'paddle2/position' -> player2 paddle position (computer?)

'motion/position' -> camera HUMAN detection

case "motion/position":

// converting the input position to -4.8 to 4.8 ??

currentMotionPosition = ((data \* 4.8) \* 2) - 4.8 ???

\*\* why does the gameboard scribe to the motion/position ???

* Gameboard should update score, ball position and paddle position, no reason to subscribe for motion/position.
* Add subscriber ‘game/state’ to change the stage image in Pong

Game-engine folder contains pong.py, player.py, game\_driver.py and game\_subscriber.py modules.

pong.py captures Pong game logic where keyboard inputs are obtained, audio sound is initialized, ball trajectory is calculated, score is kept and paddles are set up. It seems the top paddle is computer controlled and the bottom paddle is human controlled.

player.py defines ControllerPlayer, MotionPlayer, RandomPlayer, BotPlayer, and AIPlayer

game\_subscriber.py publishes and subscribes to the following topics

def emit\_state(self, state, request\_action=False):

(puck\_x, puck\_y), bottom\_x, top\_x, score\_left, score\_right, frame = state

publish("puck/position", payload=json.dumps({"x": puck\_x, "y": puck\_y}))

publish("paddle1/position", payload=json.dumps({"position": bottom\_x}))

publish("paddle2/position", payload=json.dumps({"position": top\_x}))

publish("player1/score", payload=json.dumps({"score": score\_left}))

publish("player2/score", payload=json.dumps({"score": score\_right}))

publish("game/frame", payload=json.dumps({"frame": frame}))

def on\_connect(self, client, userdata, flags, rc):

print("Connected with result code " + str(rc))

client.subscribe("paddle1/action")

client.subscribe("paddle2/action")

client.subscribe("paddle1/frame")

client.subscribe("paddle2/frame")

client.subscribe("motion/position")

client.subscribe("motion/presence")

* Why does should not publish paddle position? Checking of boundary should be a function of a paddle position.

game\_drivers.py polls the agents for actions and advances frames and game state at a steady rate.

agent = AIPlayer(subscriber, top=True)

def main contains the main loop.

Current game\_driver supports 3 states: waiting = 0, ready = 1, running = 2

# this should always be the case before starting any level - idle animations only when level == 0 and game state is waiting

subscriber.emit\_game\_state(0) # 0 waiting, 1 ready, 2 running

I would like to another state: gameover = 3. When a player leaves a playing area, computer or a player wins after lever 3, will transit to gameover state. When a player moves a paddle to the left of the screen, the game will reset and transit to waiting state

pong.py contains the Pong game logic

def step(self, bottom\_action, top\_action, subscriber, frames=3 motion\_position=None):

advance the game step until the game is over when frames > limit or no player is deteted

class Paddle accepts action left, right, none, absolute

class Ball controls the ball trajectory path, bouncing angle

def draw\_rect and render draw ball on a screen

human-paddle-control folder contains motion\_driver.py and motion\_subscriber.py modules.

Motion\_driver.py translate a human’s position obtained from a camera into paddle position. The center mass of a the largest blob determines the horizontal position.

class MotionDriver

def get\_human

why must this routine read 50 frames to determine a blob

self.subscriber.emit\_depth\_feed(self.depth\_feed) // publish (“depth/feed”)

class motion\_loop – detect human presents and his position

for counter in range(0,50):

c\_value = self.get\_human()

if c\_value == 0.5:

# c\_value is our dummy value for not seeing a human blob

# continue loop back at waiting for interaction

has\_bad\_values = True

arrayVals = np.append(arrayVals, c\_value)

The above code shows that get\_human reads 50 frames to detect a blob, then the motion\_loop read 50 human blob to publish ‘motion/presence’ true and ‘motion/position’

This code seems to induce input latency to the system. If a blob is detected, it should be published via topic ‘depth/feed’ depth image and ‘motion/presence’ to update the paddle position.

human-visualizer folder display depth images from realsense D435 to a browser

depthfeed.py spins up a server from the ‘visualizer’ folder

visualizer/depth.js subscribes to mqtt (port 9001) topic ‘depth/feed’ and render the image to a browser (port 8000)

neutral-net-visualizer folder displays a clock face showing AI’s emotion

visualization\_driver.py spins up a server from the ‘visualizer’ folder

visualizer.js subscribes to mqtt topics ‘game/level’ and ‘ai/activation’

heap.js is a heap sort

Wait (0)

Running (2)

Move Paddle Right

GameOver

No Paddle Movement

Move Paddle Left

Move Paddle Right To Play

Move Paddle Left To Reset