Capstone Design Project 2

Dance learning web application with pose estimation and deep learning

Department of Computer Science at the College of IT
School of Computer Science and Engineering

Polska Gurom:
Bartosz Han, 2021429001
Marcel Kućmierowski, 2021429010
Grzegorz Siudak, 2021429003
Wiktoria Stachera, 2021429012

Supervised by Professor Seungho Kim and Professor Kyong Kim

December, 2023

Kyungpook National University

Table of contents

Project description

Today, many people are interested in dance and decided to practice it with tutorials that can be found on the Internet. However, these tutorials do not provide the necessary feedback to continue training properly. The idea of the project is to create an application whose primary purpose is to provide a review of the performance of the user-dancer compared to a given recorded video, using the recording of the user's dance and image processing. This application aims to allow dancers to get computed-generated feedback, a missing part of video tutorials, and an excellent opportunity to improve their skills quickly. With our application, dancers can become self-learners while still having objective performance feedback. Dancers can achieve more with our application than with only video tutorials.

The proposal statement

In today's digital age, dance challenges and online tutorials have gained immense popularity across social media platforms. However, there currently needs to be more in providing constructive feedback on one's dance performance. This project aims to develop an application that utilizes pose estimation technology to score dance performances based on accuracy compared to the original video.

Defining requirements and specifications

Features of a good software

Today's projects aim to meet the software life cycle process fully. Based on Software Engineering by Ian Sommerville, we revised our work and checked for current practices that helped us improve our approach. Our program provides overall correctness of output and compliance with user requirements. It is easy to maintain and propose changes. It provides reliability, security and safety, efficiency (including resource efficiency), ease of use, and ergonomics.

Development & management

First, it is good to establish which development method to choose. We were thinking about waterfall and iterative (Figure *1*). We decided that iterative (and incremental) design better suited our perspective. Each person could work almost independently, the drawbacks would not cost much time and changes, and we could show partial results to the client (professor at cyclical meetings). As the main idea behind this process states, developing a system took place through repeated cycles and small increments improved by earlier parts of program versions. We could not foresee all the capabilities or features needed, so our program was evolving through constant iteration. This approach allowed us to communicate freely with all team members and clarify our doubts.

We were working on the project in the scope of two university courses: Capstone Design Project 2 and Problem-solving-based Engineering Training. We had a few meetings with the Professor at the Problem-solving-based Engineering Training class. We discussed our ideas, considered the Professor's opinions, and presented gradual results. We concluded our

meetings by showing the end product and discussing the next steps. The Professor suggested we develop the project further during the winter break.

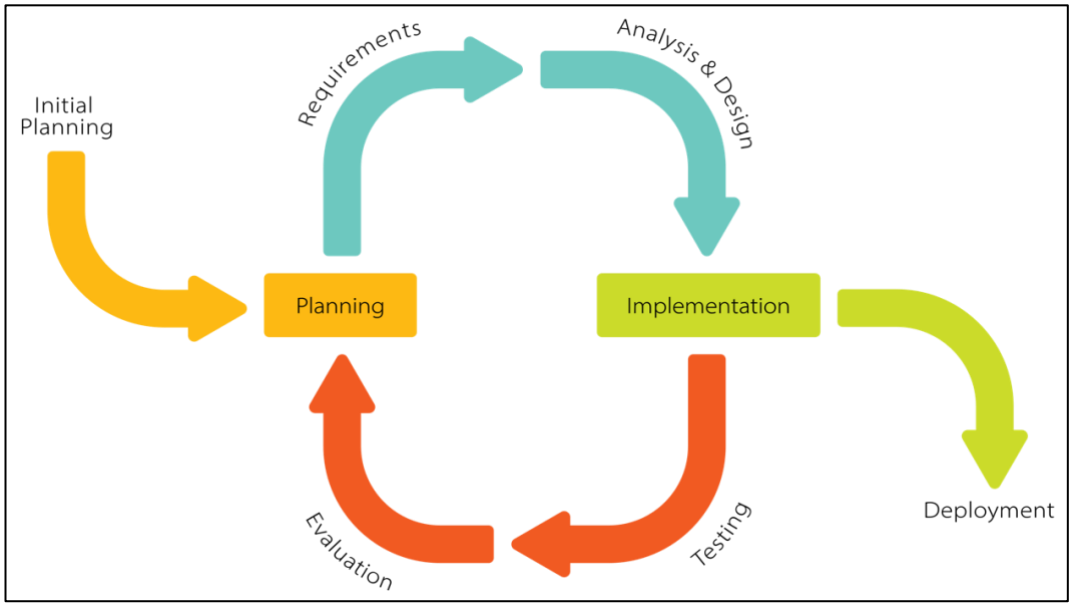


Figure 1: Iterative development model (by Krupadeluxe, wikipedia.org)

Technologies

Our project is a fully independent app as SaaS (Software as a Service). Users do not have to worry about data, O/S, servers, or storage – it is all adapted to work as a final product for the user site. We managed all of them and packed them into a single end product.

We used Python 3.10 language to program the back-end and Flask 3.0 with JavaScript ECMAScript 2023, CSS3, and HTML5 for the front-end. Testing did not require any units or libraries; we coped with that manually.

Strategy

Must have:
ability to choose and show the video, correct assessing and scoring, showing the score, zero-delay playback

Should have:
showing the score after each move, precise scoring, zero-delay reading poses

Could have:
uploading dances by user, scoreboard, saving scores in the databases, custom background color, zero-delay scoring, safety contracts, YouTube connection

Won't have:
multiplayer, availability for smartphones or tablets

Diagrams

Use case

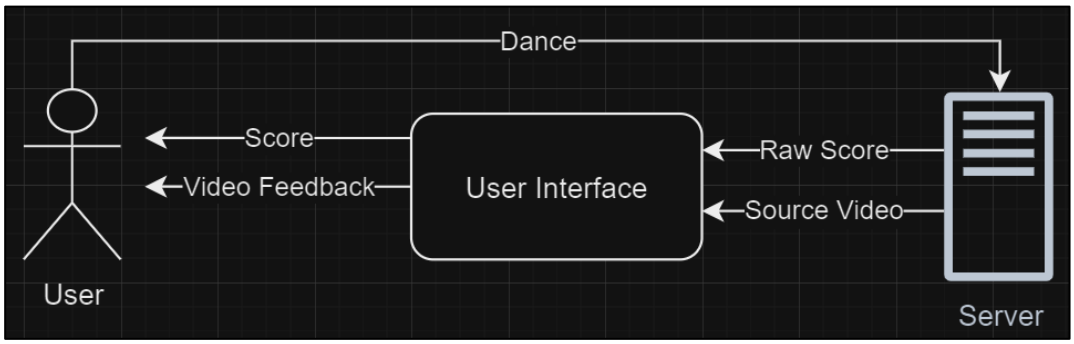


Figure 2: Exemplary use case diagram

Sequence

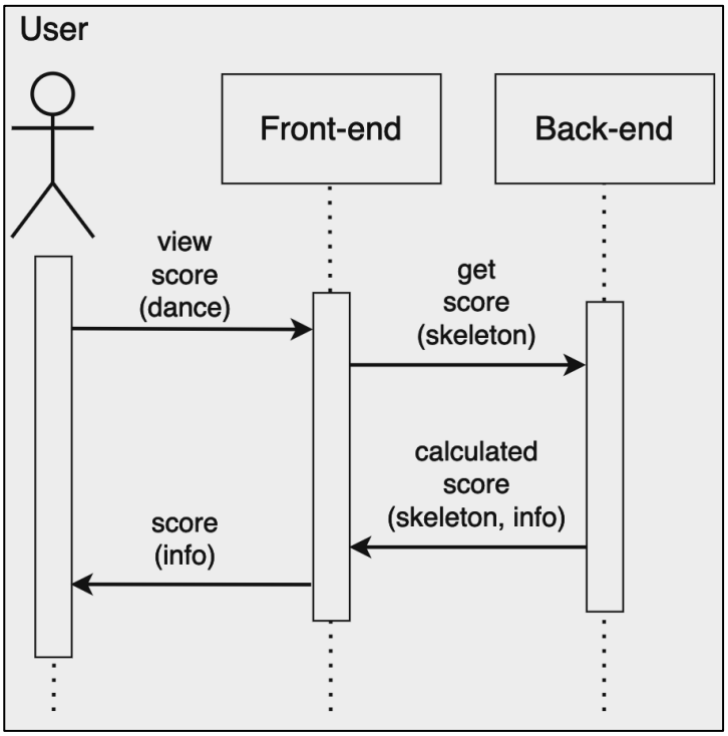


Figure 3: Exemplary sequence diagram

Requirements

Functional

1. The user must be able to choose a dance video.
2. The user must be able to watch a dance video.
3. The user must have access to all uploaded videos.
4. The system must generate a score.
5. The student should be able to dance without system interruptions.
6. The system must be guaranteed full Internet access.

Non-functional and product

1. The system must be available for 22 hours daily for all users.
2. The system can have up to 2 technical breaks of less or equal to 1 hour daily.
3. The system must not have a latency greater than 60 seconds.

Non-functional and external

1. The system must comply with the rules of preserving users' privacy following the GDPR.
2. The system must meet the civil code and law requirements on copyright.

Natural language specification

The web application must be easy to use for each type of user. The interface must be straightforward so that no instruction is needed to handle the program correctly. The system must allow users to dance to their camera and assess that performance. Dance must be adequately scored after each session. Users can choose as many dances as they want and use the system for as long as it is up.

Stories

Wiktoria is an exchange student at KNU. She and her friends are preparing an app for a course project. Her role in the project is to test the given output with dances. She can access the website application as a user – she can open, run, and try it by dancing to the laptop camera. After her performance, the app displays a score. She can assess her moves and, by that, check if the code works correctly. Her insights were possible thanks to a well-optimized program.

Alex is American and is keen on dancing. He enjoys learning new choreographies from YouTube services but finds them mostly unclarified. Alex would like to apply for dance classes, but instructors in his city do not teach choreographies to his favorite pop and hip-hop songs. He cannot afford a console to buy games such as Just Dance or Dance Central. His high school colleagues recommended him the dance learning application. Now, he can dance without spending money whenever he has time or willingness. What he likes the most about the app is that he can choose what videos to train.

Amelia is a dance instructor and a volunteer at the kids' hospital. She wanted a program to work with during her out-of-charge classes at the hospital. Amelia likes the convenience of the portability of the application and the variety of dances. She can easily access the website from the hospital using an Internet connection and her laptop. Amelia does not need any accounts or connected credit cards to provide smiles on the kids' faces. She uses the application weekly for the kids and sometimes plays with her friends for relaxation.

Scenario

Launching the web application and performing chosen dance

1. Initial assumption

A user must open the web application, provide its link, and choose a dance to perform. The user gives the page access to the camera. The user successfully met the requirements of the app and launched it accurately.

2. Normal

After choosing the dance, the user waits for it to load. The user performs a choreography. Evaluation is done automatically. The session can be closed after receiving the score (displayed on the web page).

3. What can go wrong

The system freezes. As a result, no assessment can be made. The user will have to refresh the page. The system can produce odd scores. The user will have to check his environment for any inconvenience.

4. Other activities

The creators may monitor system overload to prevent crashes and delays.

5. System state on completion

The user successfully received his score. The creator did not receive any complaints. The system allowed the user to perform again.

System context

The foundation for our project was the games Just Dance (Figure *4*) and Dance Central (Figure *5*). We played them and knew how they operate. They were primarily made for consoles like Xbox 360, Xbox One, or PlayStation 4. Just Dance is published consecutively yearly, whereas Dance Central created four main games and one for virtual reality. They are trendy but do not meet the PC users' desires (even though Just Dance Now is made solely for personal computers). Users want to choose choreographies independently from the vast library of the YouTube platform. That requirement is challenging enough to become crucial to the game's success. We look forward to adding this component to the game's current state.

We can upload as many choreographies as we want from the YouTube platform, but they must be put in the game before being delivered to the public. The part of choosing whichever dance video to practice by the user is not ready yet – one semester is not adequate for coding that part. We look forward to adding this component to the game's current state.

Figure 4: Just Dance 3 gameplay screenshot

Figure 5: Dance Central 3 gameplay screenshot

Designing

File structure

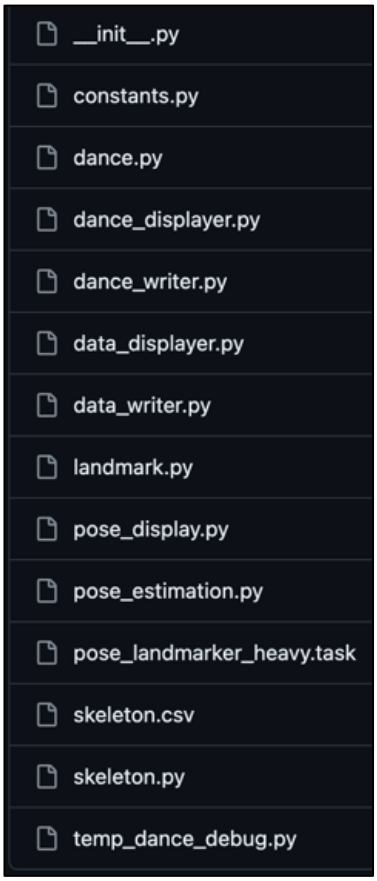


Figure 8: src folder with back-end code

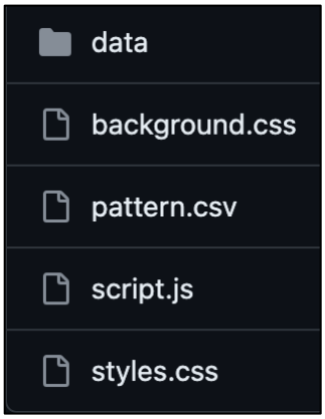


Figure 7: static folder with back-end & front-end code

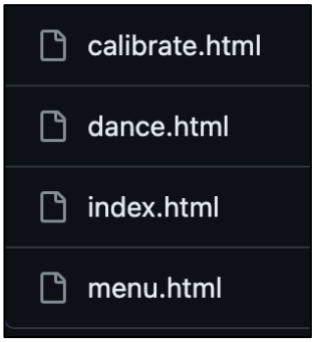


Figure 6: templates folder with front-end code

Back-end

The architecture

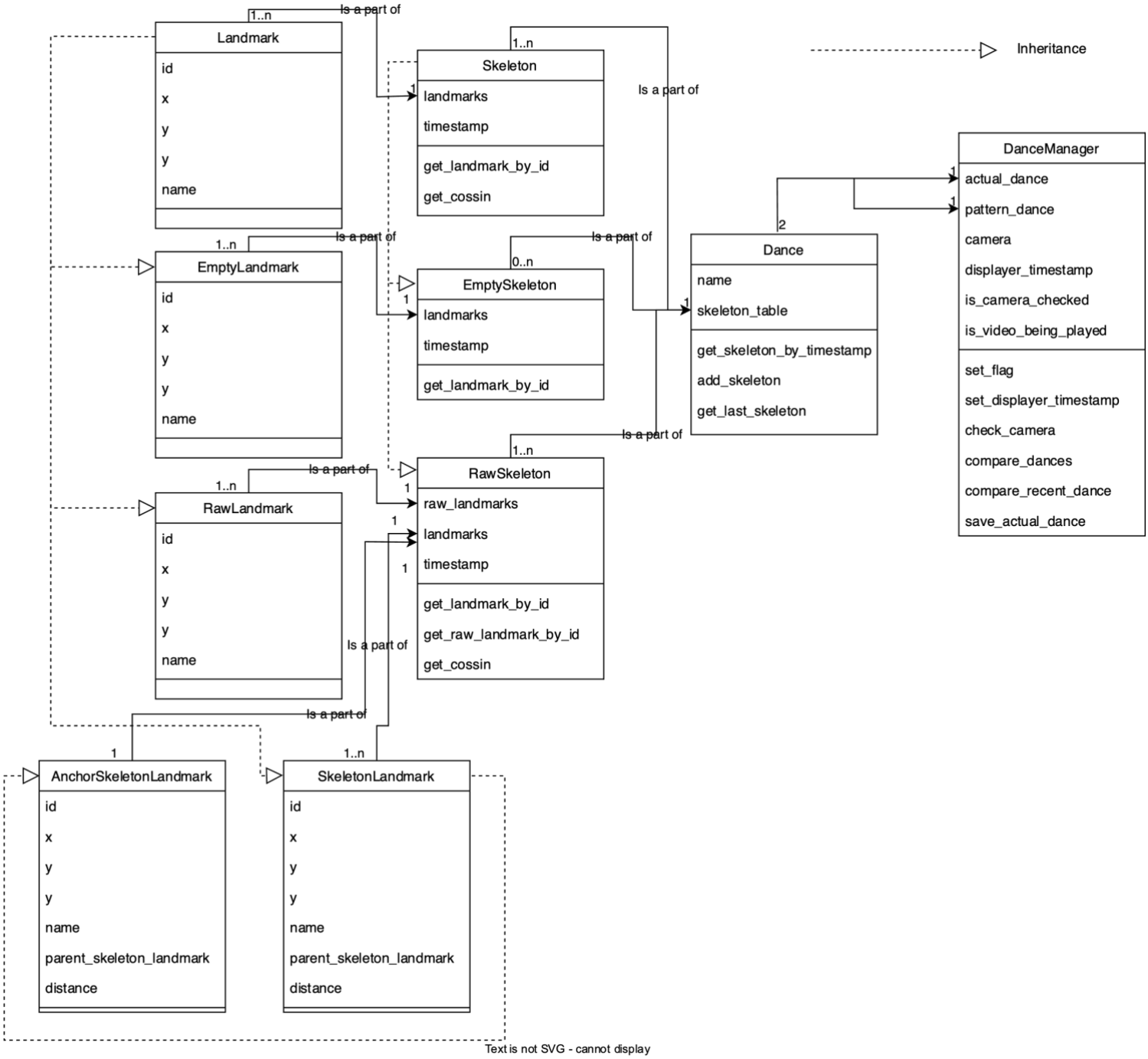


Figure 9: The architecture of the back-end code

Technology

The Python code, developed using version 3.10, combines OpenCV, MediaPipe, and CSV libraries. Matplotlib is utilized to visualize movements (solely for testing purposes). Bartosz explored alternative data models apart from Google MediaPipe but found limited accessibility and Internet access requirements. Drawing from MediaPipe documentation, Bartosz crafted specific sections of the program. The primary goal of the application was to implement an Artificial Intelligence (AI) model. This model is designed to detect body nodes from a single frame, making it particularly suitable for testing as it perceives and charts the body's nodes.

Prototype

As a prototype, Bartosz created a code that calculated the skeleton vertices (Figure *10*). He made a sample code showing the read skeleton movements and presented them as a video. In this version, each point had three coordinates – X and Y for two dimensions and Z for the third dimension. 3D could be added in future program updates since the code allows this improvement. Bartosz rewrote the code for 2D because it is enough to comprehend the application usage completely. He had to resign from multithreading because it did not comply with Grzegorz's idea on the front-end. One thread was supposed to retrieve data from the frame and create skeletons; the other was to retrieve skeletons and assess dance.

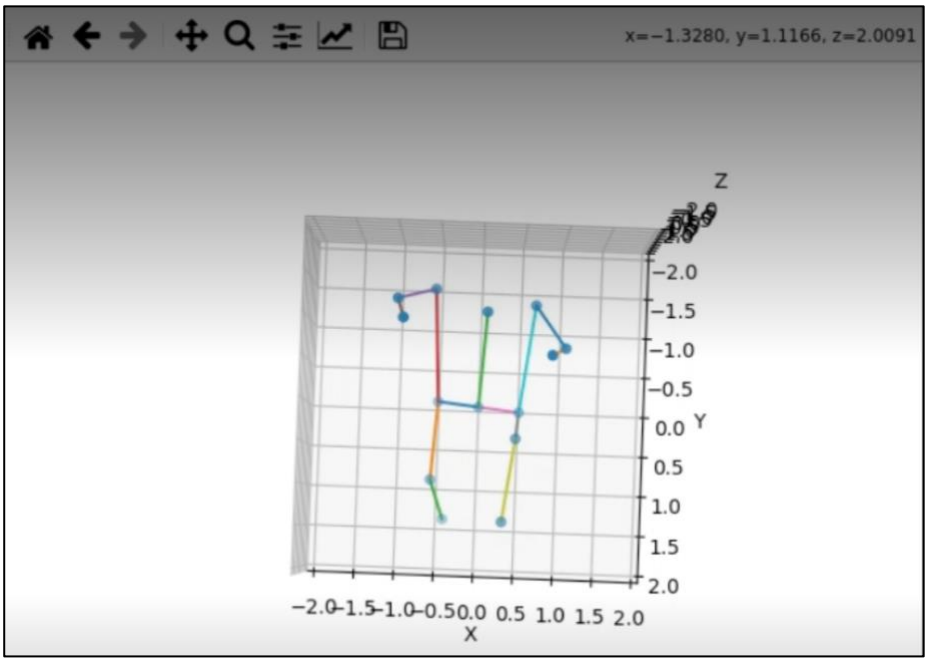


Figure 10: First version of the skeleton

Improvements

After discussing which points to choose, we focused on right and left hip, right and left knee, right and left ankle, right and left foot, right and left shoulder, right and left elbow, right and left wrist, right and left pinky finger, right and left index finger, and a nose (Figure *11*). Every body part is essential in dancing, but depending on the genre, different points significantly impact overall performance. We did not differentiate based on dance styles, but the change in the nodes is possible.

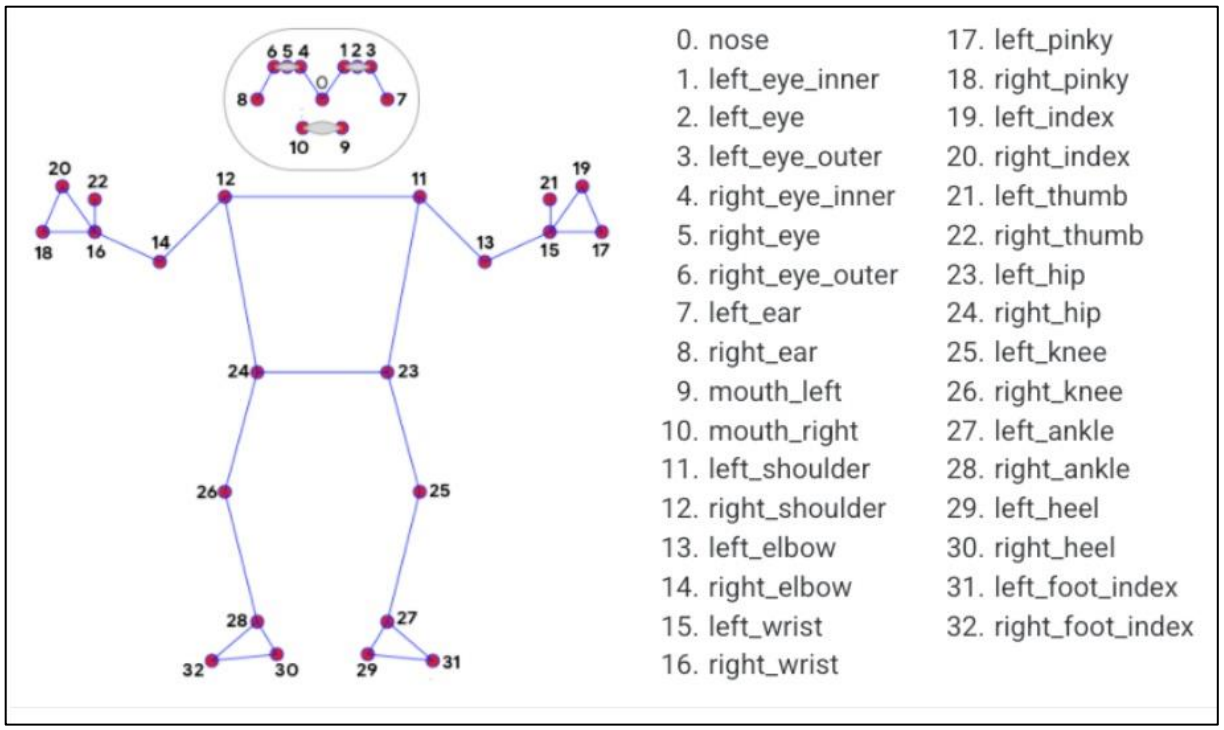


Figure 11: Available nodes on a skeleton

Insights on the code

One of the essential low-level functions reads landmarks (points, nodes) from a single picture (and writes as a CSV file) based on the pixel list. It returns a data dictionary, which is later processed for a skeleton class as a skeleton. The data class of the landmark has X, Y, and Z coordinates (floats), an ID (for distinguishing nodes based on (Figure *11*), and a name. It can be interpreted as a dictionary. Bartosz wrote a function that projects to the bool – when a node is not visible on the screen or in any other extreme scenario. To be consistent with other points, this created an empty skeleton with nullable landmarks, which can be compared by Boolean means.

Another class creates a skeleton based on an image. Another class connects not yet connected nodes, creates a graph relation (forest with one parent each), and normalizes points. Normalization is crucial since we will be measuring different people, so the program has to be accurate on lengths between the parent and the child points). It calculates the vector betwixt the parent and child, normalizes the vector on a picture, and then assigns it to the normalized parent's node. That is how the normalized node is created. Since each skeleton landmark requires a parent, there is a root object. Simply put, it is a skeleton without a parent and deprived of normalization. A skeleton is a container for landmarks (it enables accessing a landmark by its ID). It is an aggregation because the landmarks cannot exist solely (independently). Each skeleton has a timestamp assigned during reading from the video. It is needed for correct scoring.

Creating skeletons is done "live", from camera output (with normalization) or done based in the CSV file (where the normalization was done beforehand). The variable deciding the parent-child weight can be changed to make different relations for other purposes. Another container

is for fresh skeletons (with time stamps). Timestamps allow the withdrawal of a skeleton based on time (precision is based on rounding, so it is more of a closest-to-time skeleton). Dance manager class is mainly used to compare dances. It is a direct connection to the front-end which evokes this function. As input, it gets a given dance file (CSV) and creates a dance object. The web application decides from what time the video is played. It makes a ready final dance and an empty dance (for user input). In a loop, it reads a dance and goes through the process of landmark creation for the scoring system to be applied.

Final version

After testing, the final version of the skeleton and back-end part of the project saw the daylight. Each dance was tested with the printed skeleton to ensure correct node recognition (Figure 12). Some points were unable to deselect, i.e., fingers or eyes, but despite this minor inconvenience, mapping works as predicted in most cases.

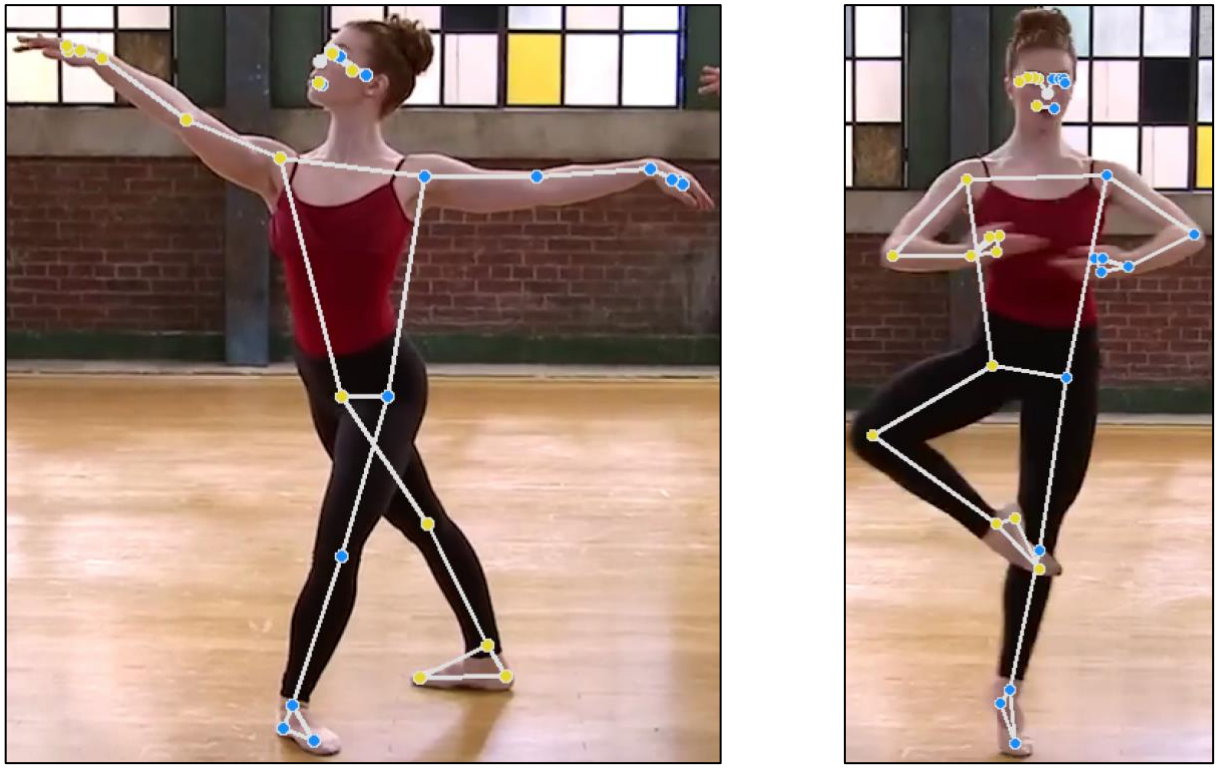


Figure 12: Mapped body

Scoring system

Marcel implements the scoring system for this project by creating several different groups of landmarks from the skeleton. Every group consists of three points, and the angle they create between them is calculated by creating two vectors from them, then using their dot product to calculate the angle's cosine, which is then converted to degrees via the arccosine function. All the groups and their weights can be seen in the code snippet presented in the figure (Figure *13*).

```
limbs = {#orientation from their perspective
        #key -> [points, weight(for cumulative error)]
        "right_arm":  [[14,12,24], 1],
        "right_hand": [[16,12,24], 0.3],
        "left_arm":   [[13,11,23], 1],
        "left_hand":  [[15,11,23], 0.3],

        "right_leg":  [[23,24,26], 1],
        "right_foot": [[23,24,28], 0.3],
        "left_leg":   [[24,23,25], 1],
        "left_foot":  [[24,23,27], 0.3]
        }
```

Figure 13: Weights of selected nodes

Then, all the two sets are compared by finding the angle difference between the appropriate groups. The weighted average of all angle differences is generated and returned. It can be done for any pair of skeletons.

Out of the ten pairs of skeletons measured, we plot the ones with the highest difference as a base, the one with the lowest difference as best, and the one without accounting for time delay as the base. As we can see on the graph (Figure *14*), the "best" plot has a much lower difference and a lower variance.

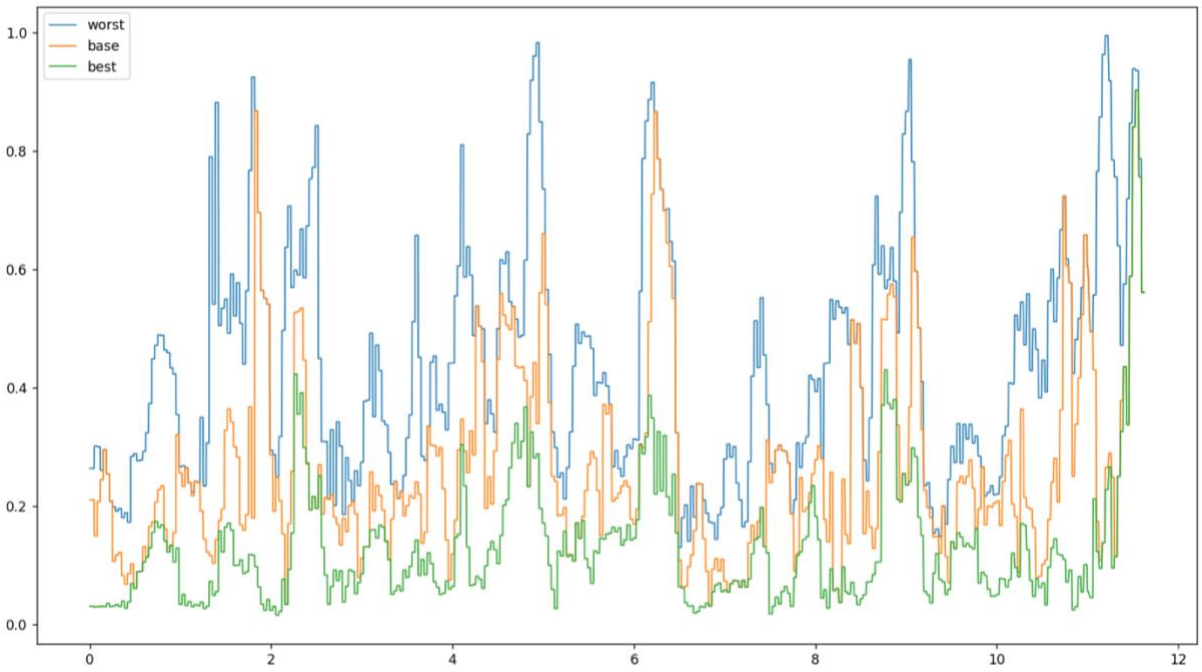


Figure 14: Comparing performances

To address the possible delay from the human reaction time, we take 10 pairs of the one-player skeleton and compare them to the dance video skeleton within 0.5 seconds backward from the given time at equal time intervals (Figure *15*). This lets us account for the human reaction time by taking only the result with the lowest difference. Finally, we convert the average degree distance to a score of "good," "very good," and others. This is assigned for

arbitrary ranges of degree differences and can be easily modified in code to give more precise results or have a different difficulty.

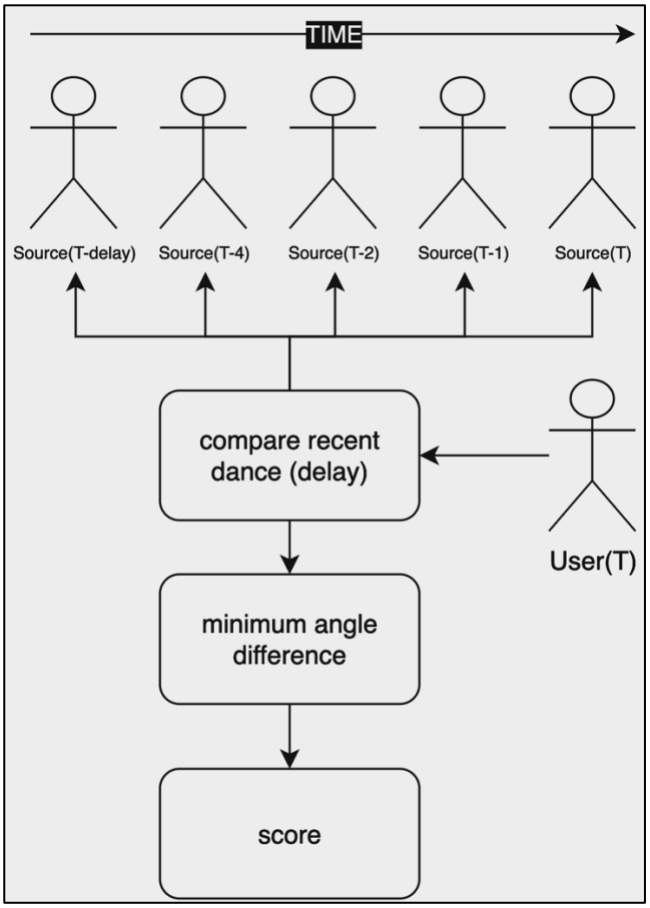


Figure 15: Time intervals

Front-end

Technology

The Python code, developed in the 3.10 version, combined with Flask 3.0 library, JavaScript ECMAScript 2023, CSS3, and HTML5, was responsible for managing the front-end part. Grzegorz used Server-Sent Events (SSE) too.

Prototype

At first, Grzegorz created the index.html (the main page). It introduced a screen that has a button redirecting to the menu. Menu.html had a hardcoded video and the function to redirect to dance.html, where the dance was shown with their miniaturized versions. Those miniatures, when hovered, showed a dance preview – they expanded a little and played. Dance.html was the page that showed the score, the choreography, and the camera stream. In the prototype, the page also showed the first version of the visual scoring representation – the change of background colors throughout the dance (black, green, red).

Later, the calibration.html page was added and responsible for calibration – showing the camera in the middle of the page only. It was needed to ensure the user was visible in the camera. After the calibration, the user was encouraged to start dancing. After assessing the

performance, index.html was complemented by a result score, showing the pie chart and the number of all the user's dance moves.

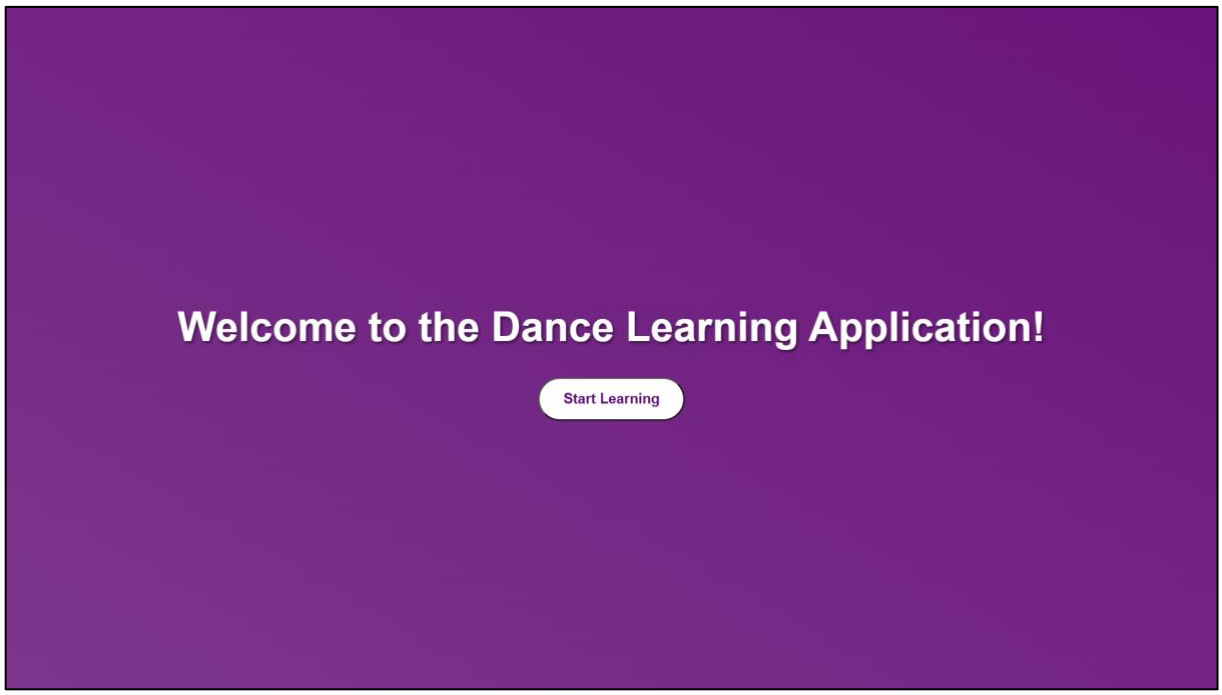
Development and the result

Grzegorz made a significant improvement on a contact point between the front-end and the back-end. That required sending the information about the chosen dance by the HTML POST to the back-end code (REST API). Calibration also needs a sign from the back-end. Dance.html was getting imaginary scores from the server initially (by SSE). They could not be calculated properly at this stage due to the poorly implemented scoring code. Almost the last version of the front-end improved the page by changing its layout (using background CSS with the same color for all pages (with the total reduction of green and red, following the same style)). The scoring page was also changed – the background animation is faster or slower (if the moves are being done incorrectly (leading to the full animation stop)). The choreography window was moved to the center-left side, and text messages relaying the ongoing movement score were added. The pie chart was taken down and replaced with the percentage score showing. The score is calculated with the Grzegorz's authorial equation:

$$\frac{\text{excellent\_moves} \cdot 4 + \text{good\_moves} \cdot 3 + \text{okay\_moves} \cdot 2 + \text{bad\_moves} \cdot 0}{\text{total\_moves} \cdot 4}$$

The "encouraging message" is shown at the end (based on the user performance) as well as the number of movements (that fall into the four categories stated in the equation).

The app.py file is a back-end file that communicates with the web page. It generates the system; by it, the messages are sent, and it receives them. The back-end starts the front-end, generates all the HTML files, and establishes the communication. The page is accessible at http://127.0.0.1:5000, where 127.0.0.1 means localhost, and 5000 is a port.



Choose the dance!

Oops!...I Did It Again - Britney Spears

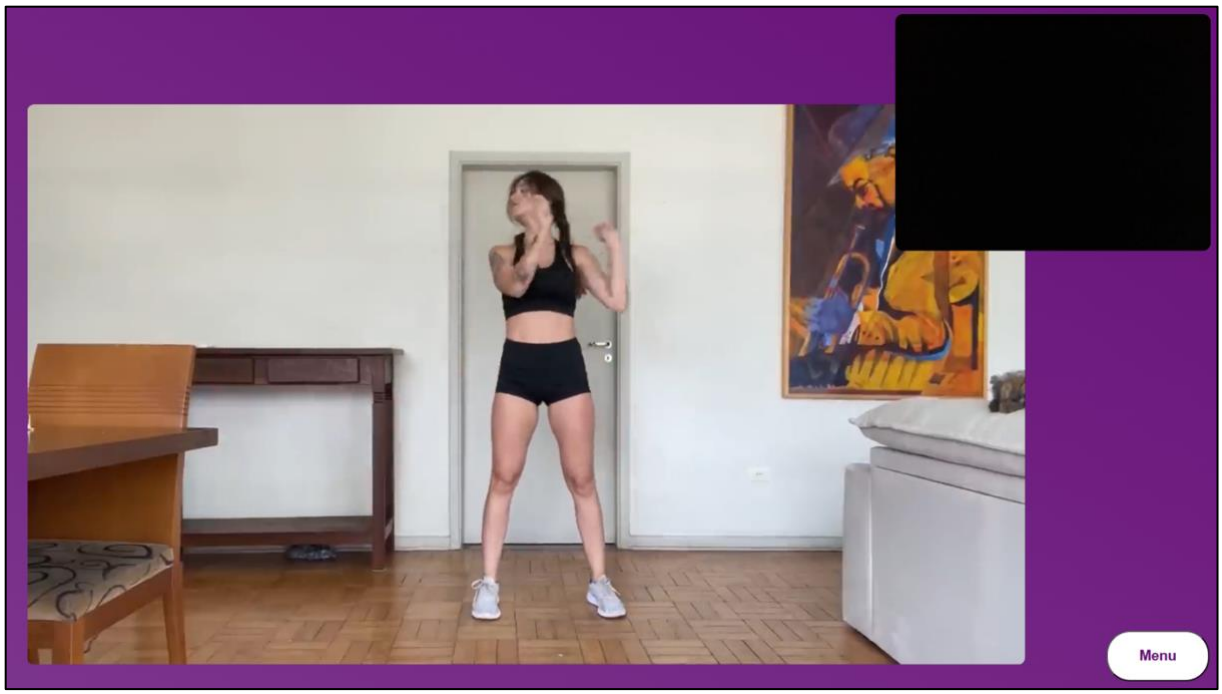
Take You Dancing - Jason Derulo

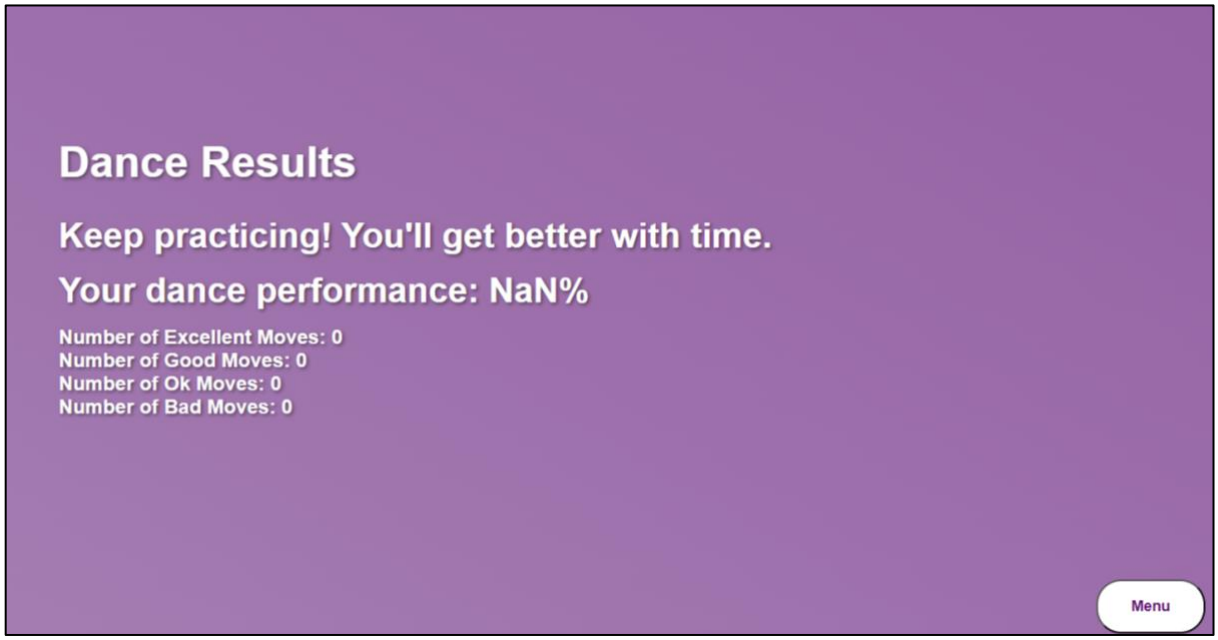
Scream - Usher

Goo Goo Muck - The Cramps

Please ensure your entire body is visible in the camera frame

Menu





Possible further changes

Grzegorz mentioned migrating the web application from the Flask to the React.js. This change would prevent the page from constantly refreshing (redirecting pages). React.js would ensure that the whole process happens on one main page. Another idea is to add announcements by the web sockets because, currently, the system sends them by SSE. It is not efficient enough and makes the whole process slightly more complex. SSE works now by setting the point stream so that the pages with access to particular data do not poll all the information. SSE operates in a client-server architecture.

Testing

An indispensable feature of incremental development is frequent testing. We started testing after the first implementation of the back-end code. It showed essential flaws in program operation at the first stage – the laptop camera was not reading the user movement correctly. Early testing allowed us to revise and repair the code before merging it with the front-end part, which would cost much time in waterfall development. From that moment, tests showed compatibility between the requirements and the code. We could focus on accurate scoring, different testing environments, and program efficiency.

Testing was the most significant part of delivering the application to the client. Not only had Wiktoria recorded the choreographies (in three versions: bad, medium, and excellent performance), which required days or weeks of preparation and simple movements for scoring testing (on Marcel's command), but also chose dances for the main web page. As a dancer, she advised the back-end team on which body parts were the most important in scoring or for skeleton-making where the application was lacking and what they had to repair. Wiktoria tested the app in diverse environments – light and dark rooms, wearing various clothes, and standing too close or far from the camera. She also tested the application regarding incorrect body movements to see how well-made the scoring is and even tried performing delays to know how the game responds. Many bugs were found during the testing, including simple or challenging code mistakes. They should be resolved in the future because we ran out of time. Apart from dancing in front of the camera, Wiktoria unsuccessfully tried running the app in a macOS laptop environment. All tests in a Windows environment went well and corresponded to the role of the user character.

Problems regarding the macOS operating system were due to MediaPipe library failure (installing the components did not help – probably, there was not a valid version of MediaPipe for the current system distribution) (Figure *16*). Although this error prevented the program from running in the console, it did not interfere with its web version.

```
Wiktoria@BBDRVs-MacBook-Pro dance_learning-master % python3 ./src/temp_dance_debug.py ./pattern.csv ./actual.csv
Traceback (most recent call last):
  File "/Users/Wiktoria/Downloads/dance_learning-master/./src/temp_dance_debug.py", line 2, in <module>
    from dance import create_dance_from_data_file, MockDanceManager
  File "/Users/Wiktoria/Downloads/dance_learning-master/src/dance.py", line 3, in <module>
    from pose_estimation import estaminate_from_frame, create_skeleton_from_raw_pose_landmarks, reverse_dictionary
  File "/Users/Wiktoria/Downloads/dance_learning-master/src/pose_estimation.py", line 2, in <module>
    import mediapipe as mp
  File "/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/mediapipe/__init__.py", line 15, in <m
odule>
    from mediapipe.python import *
  File "/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/mediapipe/python/__init__.py", line 17
, in <module>
    from mediapipe.python._framework_bindings import resource_util
ImportError: dlopen(/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/mediapipe/python/_framewor
k_bindings.cpython-311-darwin.so, 2): Symbol not found: __ZNKSt3__115basic_stringbufIcNS_11char_traitsIcEENS_9allocatorIcEEE3s
trEv
  Referenced from: /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/mediapipe/python/_framework
_bindings.cpython-311-darwin.so (which was built for Mac OS X 13.0)
  Expected in: /usr/lib/libc++.1.dylib

Wiktoria@BBDRVs-MacBook-Pro dance_learning-master %
```

Figure 16: MediaPipe Failure

Carrying out software testing showed a few vulnerabilities:

- when the dancer's pants and shoes are of a similar, dark color, the system will not correctly distinguish them (Figure 18);
- when an arm is slightly behind the dancer's back – the system has a problem identifying it (Figure 17);

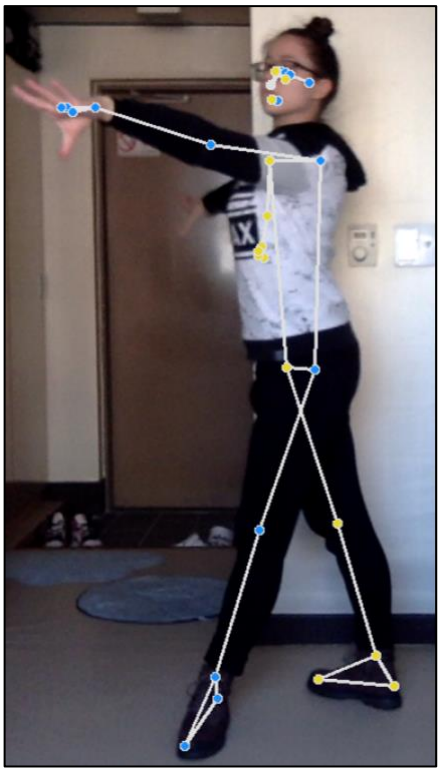


Figure 17: Incorrect mapping of an arm

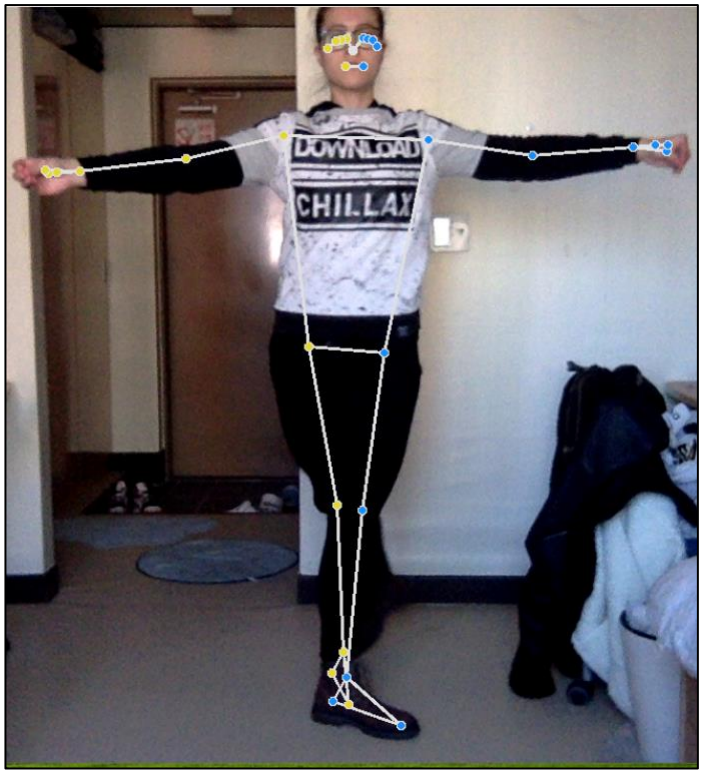


Figure 18: Incorrenct mapping of a foot

- when there are many dancers in the background – it is not determined which one will be chosen by the system (it is not always the main, the closest person) (Figure *19*);

Figure 19: Incorrect mapping decision

- when there are at least two dancers, the system might jump from marking one to another and periodically mapping limbs of both as one person (Figure *20*);

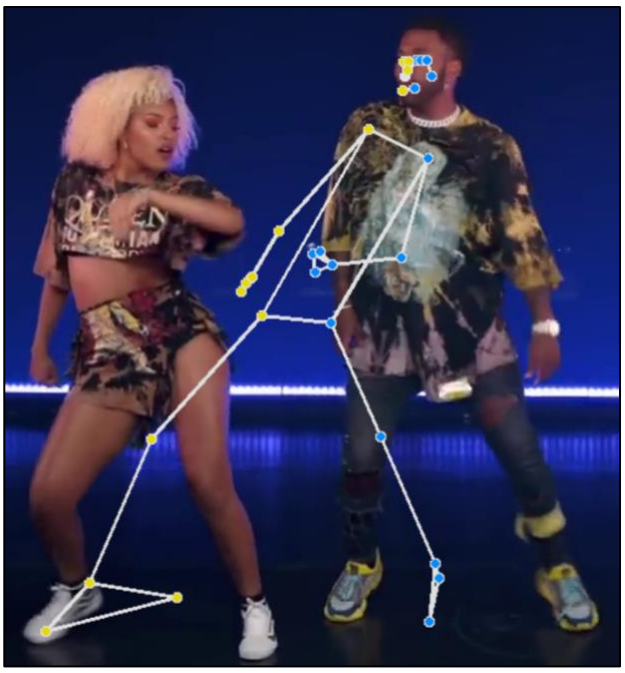


Figure 20: Incorrect mapping of bodies

- there were situations where the skeleton was staying still while the dancer was moving, or even more peculiar – the skeleton was moving while the dancer was staying still;
- hips are placed too low on the human body (Figure 21);
- some points cannot be settled correctly in closed poses (Figure 22).

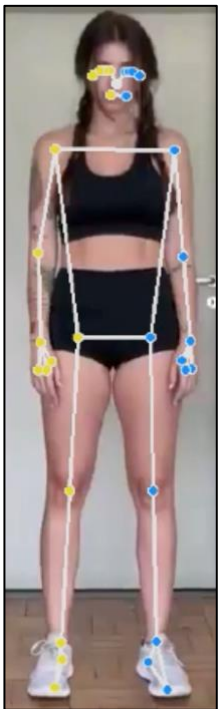


Figure 22: Incorrect mapping of hips

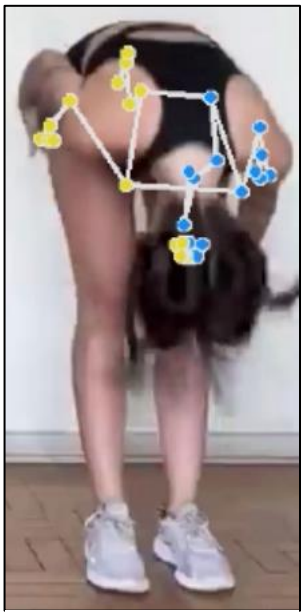


Figure 21: Incorrect mapping of closed position

How the programs assess the body parts outside the frame has yet to be determined. Some tests showed that, for instance, a foot outside of the video frame is still somehow correctly mapped. Because of the late score calculation implementation, tests regarding delays and weights are yet to be carried out.

Product delivery

Assessment of compliance with the proposal

"Develop a web-based application with intuitive UI to interact with the application." *fulfilled*

"Implement pose estimation using OpenCV to analyze user dance performances." *fulfilled*

"Generate a scoring system that provides feedback on accuracy in real-time." *fulfilled*

Back-end

Building the back-end infrastructure to process video frames, perform pose estimation using OpenCV, and stream the results to the front-end:

- utilizing Python with the Flask framework to create the back-end server; *fulfilled*
- setting up routes and endpoints for video streaming and SSE to transmit processed frames to the front-end; *fulfilled*
- implementing video capture from webcam; *fulfilled*
- ensuring seamless communication between the front-end and back-end components to maintain a real-time flow of data. *fulfilled*

Scoing system

- using OpenCV to create a frame from crucial points on the body; *fulfilled*
- creating an algorithm that compares two frames based on the relative position of key points as well as the angle between lines; *fulfilled*
- approximating an appropriate margin of error for the algorithm; *fulfilled*
- setting an appropriate time delay for the algorithm; *fulfilled*
- developing a function that assigns a score from the pose accuracy calculated by the algorithm; *fulfilled*
- introducing the scoring system to the application, assigning appropriate video feeds as "reference" and "user"; *fulfilled*
- displaying the actual time score for the user and the average throughout the reference video. *fulfilled*

Front-end

Developing an intuitive and user-friendly front-end for the web application:

- utilizing JavaScript to create UI and UX elements, including buttons, video display, and feedback mechanisms; *fulfilled*
- implementing the logic for displaying real-time video frames received from the backend; *fulfilled*
- styling the web application for a visually appealing and responsive design. *fulfilled*

Testing

- creating or using choreographies for testing; *fulfilled*
- dancing numerous sequences (from easy to hard ones) to test if the system captures well, scores correctly, and is fast enough; *fulfilled*
- using various environments and different devices to see if the system copes. *fulfilled*

Maintenance & future improvements

If we commercialize the product and put it into the market, we will prepare for conservation. This includes regular software updates and patches to address any potential vulnerabilities or bugs that may arise. We will ensure that the end product meets the standards stated in this report requirements – with great emphasis on delays. We would prepare an immense dance library and create a score database with state-of-the-art security (prioritizing safeguarding user information).

Application extensions could include sign-up and sign-in options. We would allow users to use YouTube to the fullest because, currently, the application enables dancing to videos chosen by programmers. We would work on reducing delays and improving thorough scoring based on the newest algorithms and deeper testing. Another update could include 3-dimensional reading and assesion. This would require not only Bartosz's code update but also Marcel's. Finally, the multiplayer mode could be implemented alongside the online scoring board.

Challenges

One of the first issues we encountered was a problem with running the application on macOS. It was not resolved before the back-end–front-end merge, but after that, it worked fine. We are not one hundred percent sure if the app has the same quality of performance on macOS as on Windows. The score is sometimes odd and too low – skeleton mapping imprecisions beget it. This issue is probably connected to back-end, scoring, and testing. Another challenge is that the videos might freeze or not show during dancing. Those challenges are critical – they prevent the user from becoming fully immersed – and must be resolved before a complete publication.

Ideas

We could:

- commercialize the app by adding ads to the web version;
- write the applications for a desktop for Windows (still in Python) and macOS (in Swift). This would require another approach to designing and building the app but would allow more accessibility and planning;
- allow users to choose any dance video from the YouTube platform;
- create accounts, publish scores, add multiplayer and a scoring board;
- allow users to select which dancer they would like to follow from a video;
- adapt to the split screen.

The plans for future development assume we commit ourselves to the project in the upcoming winter break, which is not a pipe dream. All goals could be milestones leading to the concluding fully independent end product.

List of figures