Problem Statement

A robot is spawned in a maze with walls having distinctive patterns on them. Your task is to make use of these visual cues to reach a predetermined target location as quickly as you can.

The environment uses a pybullet engine to render the game and the robot can be controlled using the arrows keys on the keyboard. The game has two sections, **Exploration** and **Navigation**, as follows:-

1. **Exploration Stage** - In this stage, you can explore the entire maze however you want. Explore basically means move the robot around and “see” various sections of the maze and store information for future use. You may choose to store any information (images, feature descriptors, etc) you feel might be useful for reaching the target location in the shortest time possible.
2. **Navigation Stage** - In this stage, your task simply is to make use of any data you collected during the exploration stage to reach the target location which will be displayed as soon as you exit the exploration stage in the shortest time possible (as compared to your peers).

Setting up the Environment

Please refer to this README file for instruction on how to set up the project environment on your system.

How to run the Game

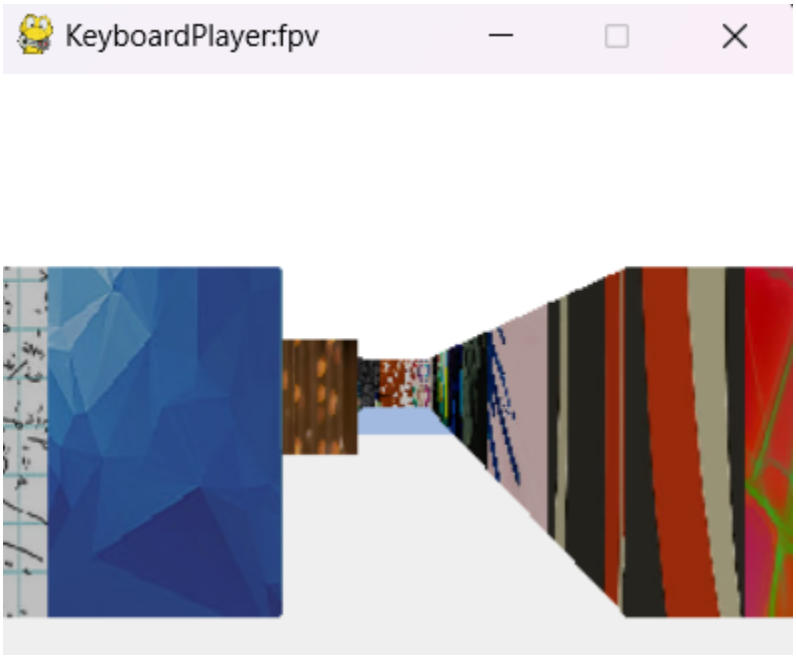
1. Activate the game environment using conda

```
conda activate game
```

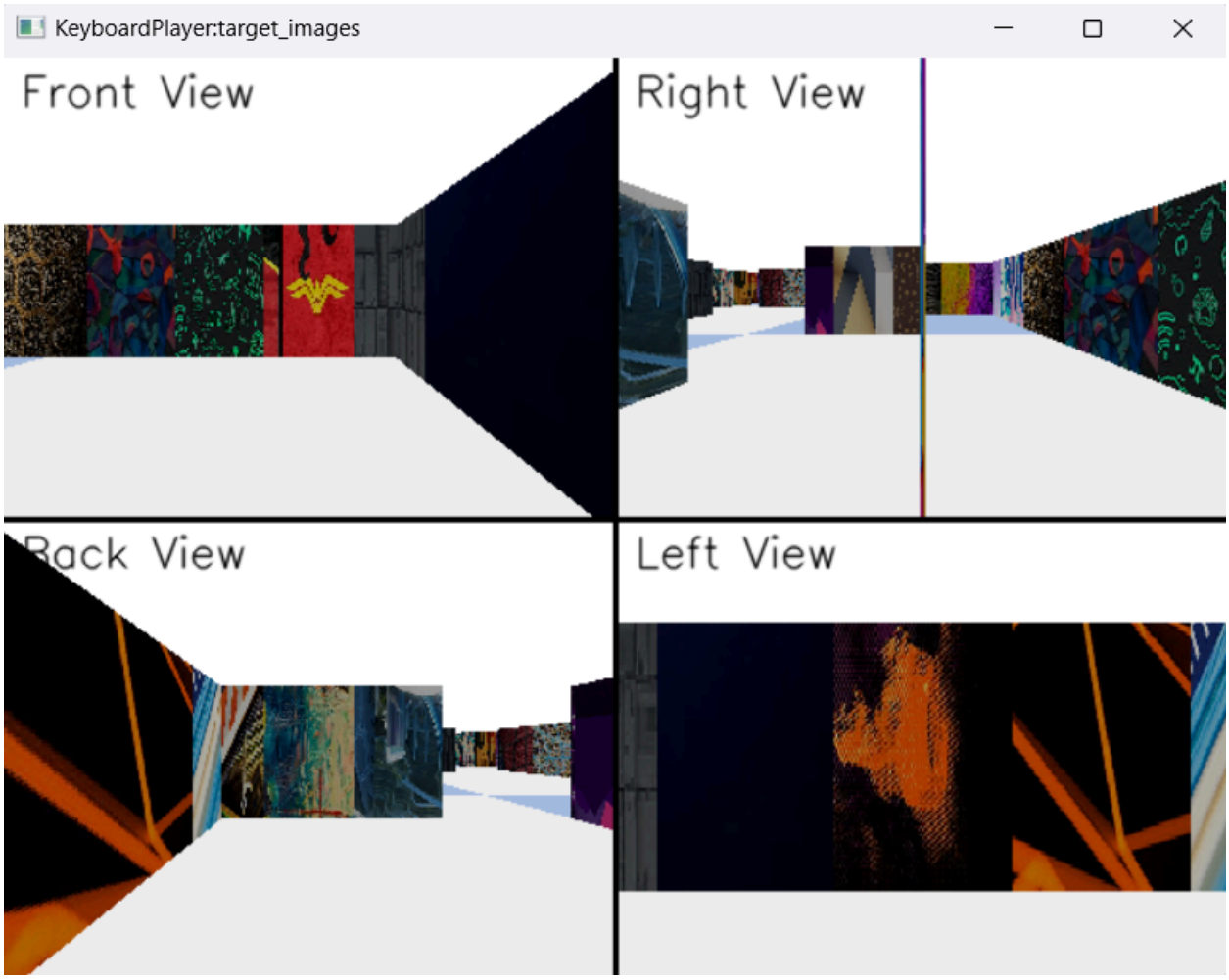
2. Run the following command to start the game

```
python player.py
```

3. A pybullet window will open up, give it a few seconds to load the maze.
4. Once the maze loads, you will see the First-Person View (FPV) of the robot in the pybullet window. It is basically the images acquired from the camera. Following is an example of what the window will look like. The view, however, might change as we change the maze layout for the exams.



5. By default, you start with the exploration stage. You can use the arrow keys on the keyboard to move the robot around and “see” the maze and patterns for a certain period of time (exact time will be announced soon). You cannot move to the navigation stage until this set amount of time has passed.
6. As soon as the set time has passed, you enter the pre-navigation stage and you will be able to see the four views (front, back, left, and right) of the target location. This is the stage where you could do any processing you want before actually starting navigating to the target.
   Following is an example of how the four views of the target location would look like. Again, it will change as we change maze and target location.



Note: Your time count starts exactly at this point. As soon as you see the target images, your time count starts.

7. Once you complete all your processing, you then enter the navigation stage. In this stage, you can move the robot around, again using the arrow keys, to reach the target location using the help of visual cues and whatever data you collected during the exploration stage.
8. Once you reach the target location (the game won't show you whether you've reached the target or not), you can press the 'ESC' on your keyboard to indicate that you've reached the target and you want to exit the game.
9. You'll find a *game-yyyymmdd-hhmmss.npy* file saved in the *data/save/* directory in the *vis_nav_player* folder. Do not bother extracting the contents of the file because it is encrypted. This is the file you will be required to upload once you complete the challenge.

How to run the Baseline Solution

We provide a standard baseline solution based on Visual Place Recognition technique for you to get started with. You are required to improvise certain sections of the solution in order to reach the target in the shortest time possible. Please make sure to go through the comments thoroughly before running the solution. This'll help you understand the solution and the motivation behind why we're doing what we're doing. Comments starting with "TODO: " are places where you can potentially improvise. You are not required to work on all the TODOs, you may choose to work on only a few or all of them. But remember, your end goal is to reach the target location using visual cues in the shortest time possible.
Following are the steps for effectively running the solution:-

1. **Exploration Stage**
    a. We move the robot around and save every FPV image to the *data/images/* directory. The image names are *0.jpg*, *1.jpg*, *2.jpg*, and so on.
    b. The VLAD feature of every FPV is calculated and is appended to the database list. You won't be able to calculate the VLAD feature without having generated the codebook.

2. **Pre-Navigation Stage**
    a. The first step is to generate the codebook (*codebook.pkl* file) i.e. our visual vocabulary. In order to do so, we first calculate the SIFT descriptors for every image we saved during exploration and train a KMeans Clustering model and save the model. This step needs to be performed when you run the game for the very first time. For every subsequent run, you can simply import the saved *codebook.pkl* (the clustering model) file. This is our visual vocabulary and will help us calculate the VLAD embeddings in future runs.
    b. Next we create a tree (BallTree) using our database of VLAD features for fast nearest neighbor search. We will use this tree to find a nearest neighbor that'll potentially help us reach the target location.
    c. The last step before actually starting navigation is find, in the database, where our target location is. So we query the tree with the

front view of the target image to give us the ID of the database image that matches closely with the target location. In simple terms, we ask the tree to give us the ID of the image that looks very similar to the given (provided) image. Now our task is to reach that image in the maze from the starting point.

3. **Navigation Stage**
    a. The robot will return back to the starting position.
    b. Now in order to reach the target, we will use a simple mechanism. We initially don't know which direction to go. So we will start with querying the current FPV by pressing the 'q' key on the keyboard and get the image closest to it in a different OpenCV window. This is basically the next-best view to go to. So you look/move around for the view that matches the next-best view and then query again for moving forward.
    c. This will print the IDs of that image and the target on the terminal. The ID will help you understand whether you're moving towards the goal ID or away from the goal ID.
    d. You can save time by not querying along straight paths, because basically there is no other way to go.
    e. If you are not able to find the best view nearby, it is possible that the view is a false positive. You can guess this either by looking at the ID of that view, if it is too far off from the previous best-view ID then there are high chances that it is a false positive, but not necessarily. So you might need to monitor the next best view and the ID of the view in case you're not able to find the best view nearby. In such cases, just move the robot a bit in another direction and query again.
    f. You keep querying when you don't know where to go next by keeping track of the next-best view ID.
    g. As soon as the next-best view ID is very close to the goal ID, you visually compare the FPV with the target location view to confirm whether you've reached the target location or not.
    h. You can quit the game in any direction you want (the orientation of the robot does not matter), however the location should be the same.