Imperial College
London

Project report for Programming 3 Group Project

Data Labelling Tool for Detection and Tracking of Ants

Author names:

Chanasak Mahankarat, 01420806

Cal-Vin Ng, 01374327

Joshua Sia, 01412775

Yi Teng Voon, 01411911

Xian Zhang, 01367745

Supervisor(s): Mr. Martin Holloway, Mr. Fabian Plum, Dr. David Labonte

Feedback box for project markers:

WHAT I liked about the report:
WHAT could/should be improved:

Table of contents

1. Introduction

1.1 Background

The Evolutionary Biomechanics Group[1] led by Dr David Labonte is currently working to automate behavioural analyses on ants, specifically leaf cutter ants (*genus Atta*), to study emergence and swarm intelligence in eusocial insects. Individually, ants are unable to perform complex tasks and are likely to die. However, in a colony, ants are able to delegate tasks, communicate with each other, and forage for food. The research aims to answer the question of how ants are able to display such behaviours in a colony, and to use the results to build a model which can be studied and then applied to systems such as traffic handling and route planning. In order to achieve this, the research is broken down into two main parts. The first part involves tracking the position of individual ants, and the second part involves task recognition. This project focuses on the first part of the research - tracking individual ants' positions.

1.2 Problem

Machine learning and computer vision tools play a huge role in ant behaviour research, however, large training datasets are required. Collection of labelled data can be difficult because of the sheer number of ants, and the complexity of behaviours displayed.

1.3 Objective

Thus, there is a need to develop an application that will be used for labelled data collection, which should be easily used and understood by the general public. The UI will be used in a zoo in Zurich as part of a Citizen Science project, in order to collect more data efficiently and quickly to train the machine learning algorithms.

[1] Evolutionary Biomechanics Group, http://evo-biomech.ic.ac.uk/

2. Requirements Gathering

2.1 User requirements

An initial discussion was held with the project customer, Mr Fabian Plum, a PhD student from the Evolutionary Biomechanics Group. Through this meeting, the following user stories were recorded:

As a user:

1. I want to be able to view available videos for labelling.
2. I want to be able to select which video I will be labelling.
3. I want to view the most recently labelled frame, so that it is easier for me to update ant coordinates.
4. I want to view the next and previous frames, so that it is easier for me to see the path of the ants.
5. I want to label multiple consecutive frames for a video.
6. I want to add, update or remove ant labels in a frame.

As an administrator:

7. I want to upload videos for users to label.
8. I want to view all the submitted labelling data.
9. I want to export all the submitted labelling data as a CSV file format.

The user stories that were collected were visualised using a Universal Modelling Language (UML) Use Case Diagram, which is shown in Figure 2.1.

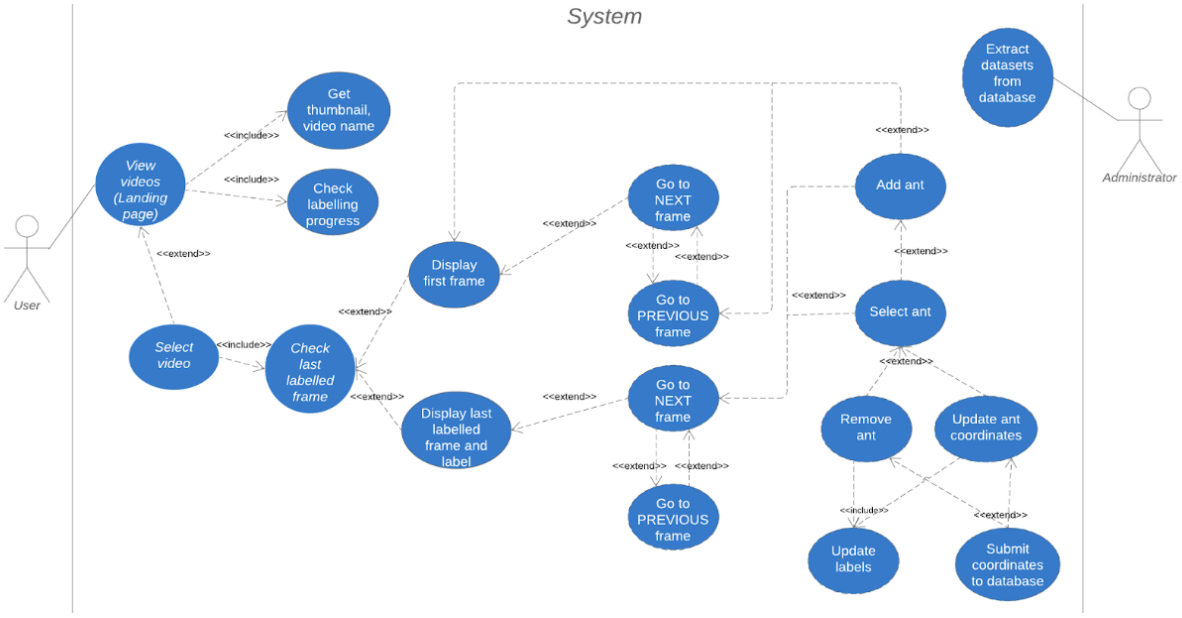


Figure 2.1: Use Case Diagram for the Ant Labelling Application

2.2 Functional requirements

Using the use case diagram, the functional requirements for the application were identified as shown in Table 2.1.

Aspect	Specification
User Interface (UI)	UI has to be intuitive and user-friendly such that members of the general public are able to use it correctly with minimal instructions
Platform	Application has to be developed in Java and deployed on Heroku
Data storage	Frame images uploaded by administrator must be stored in a remote location accessible by all instances of the application
	Data submitted from the application must be stored in a remote location that is accessible only by the administrator
Data extraction	Viewing and extracting datasets from the database has to be user-friendly
Functionality	Application must allow users and administrators to perform the use cases defined in Figure 2.1

Table 2.1: Technical specifications

3. System and software design

3.1 Structure of our software

An overview of the system and software design is shown in Figure 3.1.

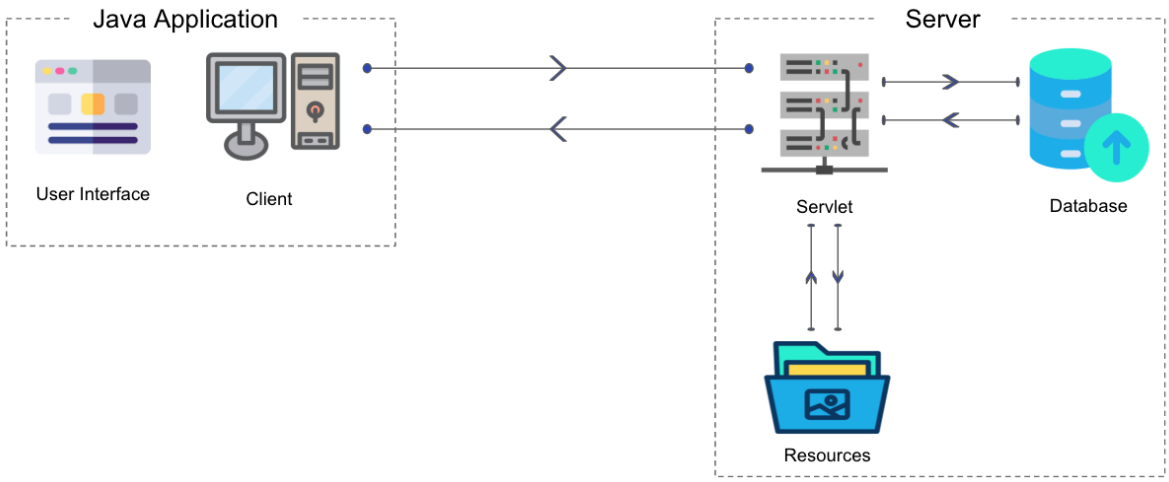


Figure 3.1: Overview of system design

The user interacts with the software through a Java application. On the backend, the server is comprised of a servlet, a database and a resource directory, which is used to process data sent by the client from the Java application, and to provide necessary information for the client. This is done by establishing a connection from the client to the server over a protocol, which in this project, is Hypertext Transfer Protocol (HTTP)[2].

This client-server model allows for requests to be sent to the server, and responses to be received by the client. Thus, a wider range of functionalities can be implemented such as the storage of labelled data in a cloud database, and the retrieval of information from the server's resources.

[2] HTTP Protocol, Mozilla, https://developer.mozilla.org/en-US/docs/Web/HTTP

3.2 Classes

Figure 3.2 shows screenshots of the different pages in the application, with the classes and their corresponding elements labelled. A full class diagram with their packages are shown in Appendix D.

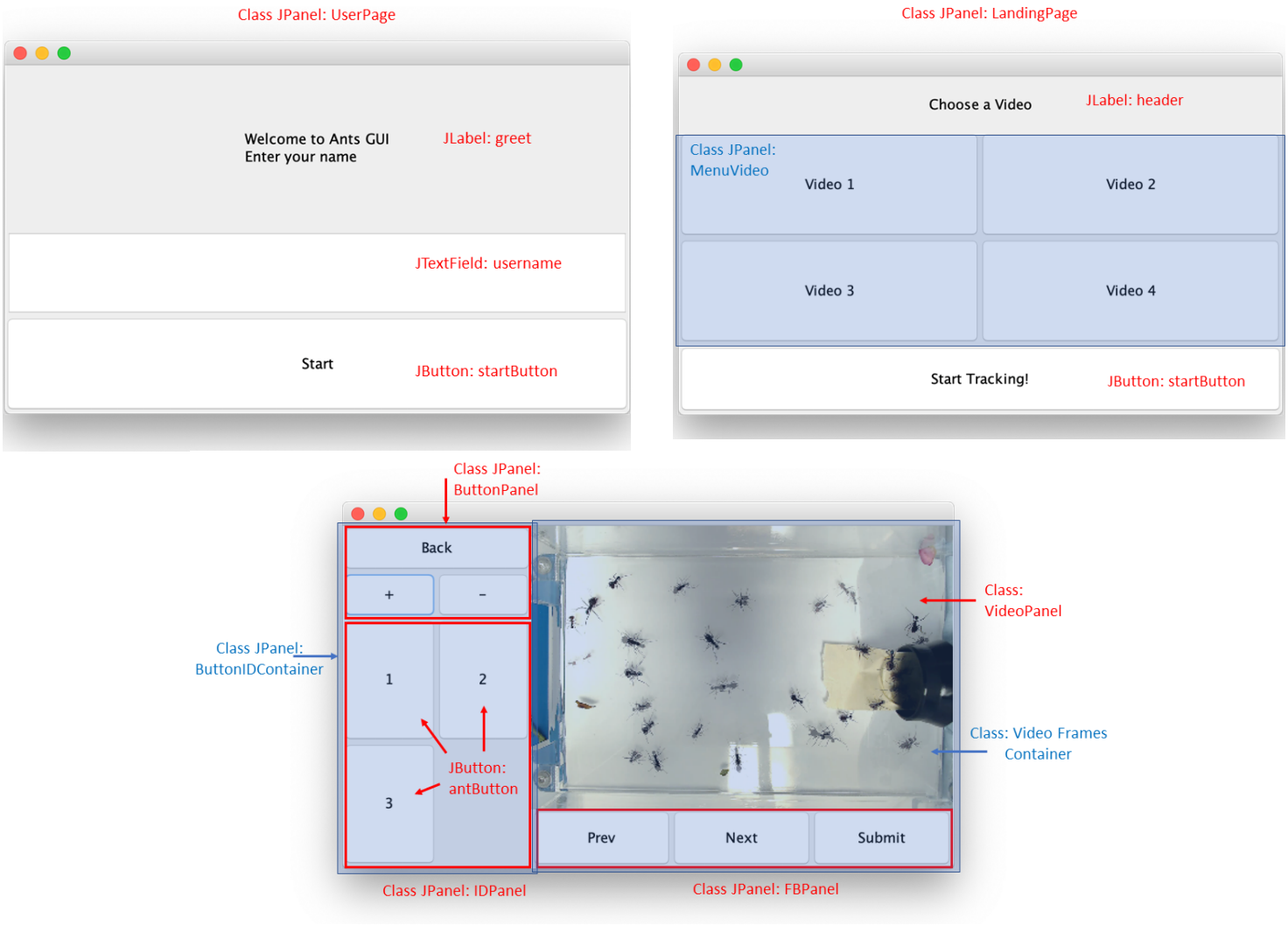


Figure 3.2: Labelled screenshots of the pages in the Java application with their corresponding classes

3.3 Frontend

3.3.1 User Interface

There are three main pages associated with the UI: User page, Landing page, and Tracking page. The three pages are shown in Figures 3.3a, b, c respectively.

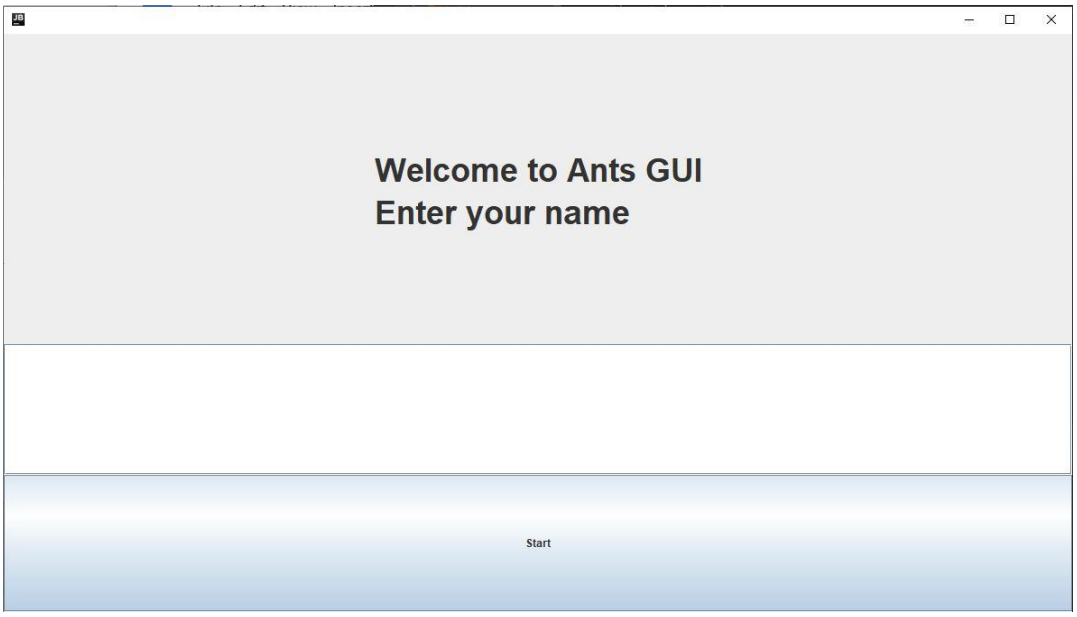


Figure 3.3a: User page

User page: When the application starts, the first page shown to the user is the User page (Figure 3.3a) where the user is able to enter their name. Currently, the username is not stored and does not have any significance. However, in future developments for use in Zoo Zurich, the username can be used to filter input data in case a user repeatedly inputs inaccurate data by the administrator. Usernames can also be used for gamification, allowing different users to compete with one another and encouraging this competition through a leaderboard. After clicking on the "Start" button, the user is taken to the Landing page (Figure 3.3b).

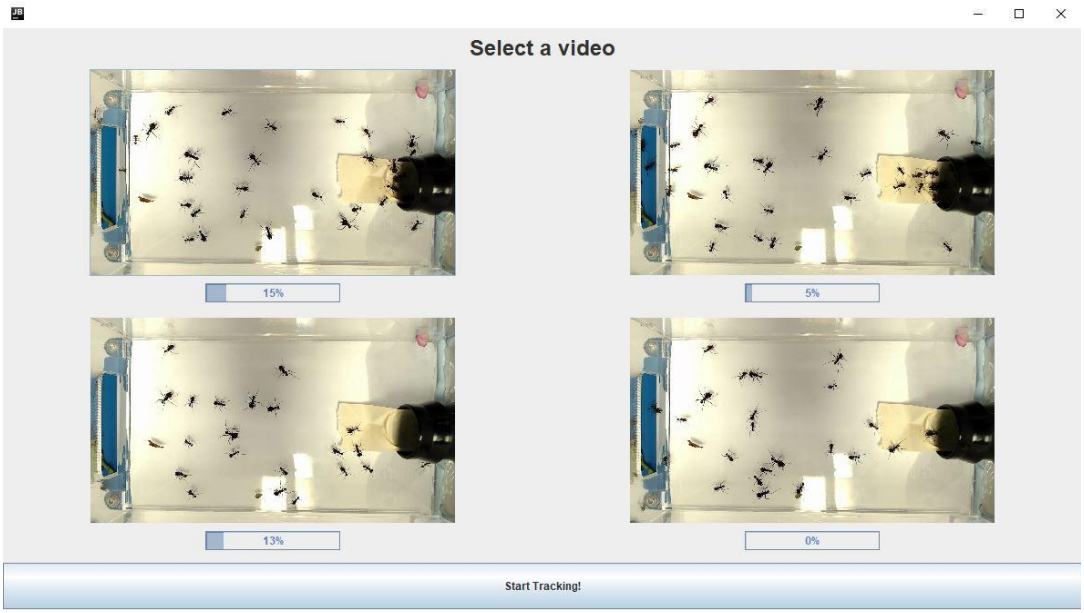


Figure 3.3b: Landing page

Landing page: The user is directed to the Landing page (Figure 3.3b) where thumbnails of 4 videos are shown in the video menu. Their corresponding progress bars are also shown below each video thumbnail. The user selects one of them and proceeds to the Tracking page (Figures 3.3c and 3.3d) by clicking the "Start Tracking!" button. By default, Video 1 is selected in case the user forgets to click on a video.

Figure 3.3c: Initial Tracking page

Figure 3.3d: Tracking page

Tracking page (buttons): After clicking the "Start Tracking" button, the user is taken to the Tracking page (Figure 3.3c) where an initial frame of the video is shown. The initial frame displayed is the last labelled frame, and it has the labelled ant coordinates displayed in green dots which each correspond to a

unique ant ID. Each ant button shown on the left corresponds to one ant. When the user clicks the “Next” button, the next frame of the video (Figure 3.3d) is shown with its overlay frame, together with the coordinates corresponding to the overlay frame shown in red dots. When the user clicks on an ant to label, the dots will be shown in green.

Users are able to add new ants if any enters the frame, or remove ants if any moves out of the frame. After clicking an ant ID button, the user clicks on the corresponding ant in the image, the coordinates of which is then stored into an ArrayList. Each successive click for the same ant replaces the previous coordinates and thus, only the last click is saved. This action is repeated for all ants in the frame. After the frame is finished labelling, the user clicks on the Submit button and the data is sent to the server to be stored in the database.

Tracking image (display): There are always two images displayed. The first image corresponds to the current frame, and the second corresponds to one frame before the current frame, and is called an overlay frame. The current frame is displayed with 99% opacity and the overlay with 50% opacity. This is done so that users can get a better idea of the trajectory of an individual ant, which ensures more accurate labelling. In addition, the “Next” and “Previous” buttons allow the user to move through frames in case the overlayed image does not provide a good enough estimate of the ant’s trajectory. If applicable, the overlay frame is also displayed along with the labelled ant coordinates in red dots.

3.4 Backend

3.4.1 Servlet

The servlet is deployed on to heroku server such that the application can access data and video frames from anywhere. The protocol used in the client-server model of this project is the Hypertext Transfer Protocol (HTTP). Thus, all data transferred between the client and the server is sent as text/html. This results in large buffer times when images are converted into bytes and then sent from the servlet to the client. The data to be transferred is also stored in an object which is converted from a GSon object to a JSon string.

POST methods are used in this project since they allow both a header and body to be sent to the server in contrast to a GET method which only allows a header to be sent. 4 different POST methods are used by the client in the project, and in order for the server to respond in a desired manner, there are 4 different URL patterns in the servlet, which are matched to the corresponding POST methods in the client, as presented in the Table 3.4a below.

URL Pattern	Situation	Client	Servlet
/init	Initialisation of landing page requires the app to display the thumbnails of 4 videos and their corresponding	postInit method is called after the user enters their username and presses the “Start” button to request for videoIDs, thumbnails and progresses for 4	Receives the request, queries the information from the database and retrieves a thumbnail from the resources directory. This is stored in an object of class InitData, in ArrayList type, which is then sent back to the client as a JSon string

	progress bars	videos	
/landingpage	Once the user selects one of the videos from the Landing page, the app has to display the last labelled frame, and relevant ant data	postLanding method is called, sending the videoID of the selected video to the servlet and requesting necessary information	Receives the request and queries the last labelled frame and the ant data corresponding to that frame from the database. The servlet also retrieves the last labelled image frame from the resources. In addition, it also queries overlay ant data and overlay frame image. This is stored in an object of class LandingData, which is sent back to the client as a JSon string
/fbpage (forwards/ backwards page)	When the user clicks 'Next' or 'Previous' buttons, the app has to display the next or previous frame, their corresponding overlay frame and the ants' positions	postFB method is called, sending an object of class FBData containing the current videoID and frameID and a boolean variable indicating whether the 'Next' or 'Previous' button was pressed	Receives the request, queries the ant data of the correct frame and retrieves the correct frame image. The same is done for the overlay frame as well. This is stored in the same object received from the client and sent back as a JSon string
/submitpage	Once the user has labelled all the ants and clicks on the 'Submit' button to submit the labels.	postSubmit method is called, storing videoID, frameID and ants data (a two-dimensional ArrayList, each row containing antID and its x,y-coordinates) in an object of class SubmitData and sending the object to the servlet	Receives the submit data and inserts them into the database

Table 3.4a: List of URL patterns and their functions

3.4.2 Database

The data collected is stored in a PostgreSQL[3] database provisioned by Heroku[4]. The data is stored in a table named 'coordinates' and consists of the following five columns: video_id, frame_id, ant_id, x_coord and y_coord. The constraints for each attribute are shown in the entity relationship diagram, shown in Figure 3.4a. A composite primary key was chosen to uniquely identify each record in the table. This composite primary key is a combination of the attributes video_id, frame_id and ant_id, which is always unique for every record in the table.

[3] PostgreSQL, https://www.postgresql.org/
[4] Heroku, https://www.heroku.com/

Coordinates
ant_id, int, NOT NULL, PK
frame_id, int, NOT NULL, PK
video_id, varchar(128), NOT NULL, PK
x_coord, int, NOT NULL
y_coord, int, NOT NULL

Figure 3.4a: Entity relationship diagram for the table 'Coordinates'

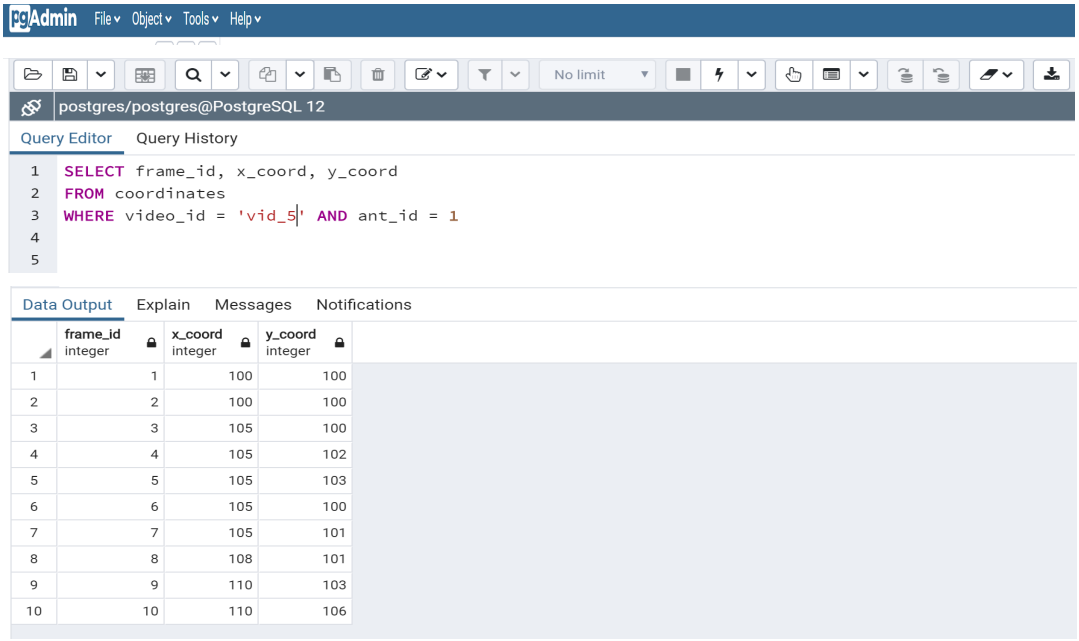


Figure 3.4b: Screen capture of pgAdmin4

The administrator accesses the database through the administration and development platform for PostgreSQL, pgAdmin4[5], shown in Figure 3.4b. Interactions with the database can be done either through SQL queries, or the graphical interface in pgAdmin 4, providing the administrator with an easy way of quickly retrieving very specific pieces of data from the database. The administrator is able to extract the desired datasets by downloading them as a CSV file format.

[5] pgAdmin, https://www.pgadmin.org/

4. Implementation and unit testing

4.1 Collaboration Tool

In terms of version control, GitHub[6] was used. Two remote repositories were each created for the app and the servlet. In both these repositories, branches were created so that individual members of the project are able to make their own changes to the code, which can then be merged together. Tags were also created to keep track of the version of the app and servlet so that if there are major changes in the code, previous versions can still be restored. The app repository is called Ants, which has 5 branches. The servlet repository is called AntsServlet which has 3 branches.

4.2 Unit testing

Unit tests are essential in the project because if in future developments, the code is changed significantly, unit tests enable users to identify the parts of the code that are still functioning as intended.

In terms of unit testing, the Hamcrest[7], Mockito[8], and Abbot[9] dependencies were used. Abbot is used for testing GUI components. Mockito allows mock objects to be created, which means that tests can be carried out independently of the app. This is useful because in a Gradle project, tests are carried out before the actual app. The order of tasks to be executed is determined by the Directed Acyclic Graph (DAG) shown in Figure 4.1, and it goes from the bottom to the top.

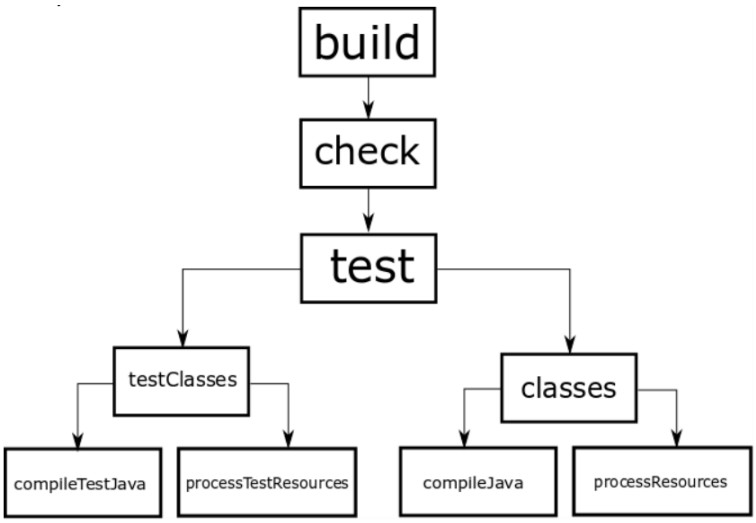


Figure 4.1: Gradle Java DAG[10]

[6] GitHub, https://github.com/
[7] Java Hamcrest, http://hamcrest.org/JavaHamcrest/
[8] Mockito, https://site.mockito.org/
[9] Abbot framework for automated testing of Java GUI components and programs, Abbot, http://abbot.sourceforge.net/doc/overview.shtml
[10] Programming 3 Notes 2, Mr Martin Holloway, 2019

5. Integration and system testing

Heroku is a cloud platform that enables developers to deploy applications to the platform and is built in a remote environment. Since the war (web jar) file is deployed on a remote Heroku server, the app and server can be accessed from any computer. This has obvious advantages over applications hosted on a local server.

Travis CI is a continuous integration tool that automates testing, integration and deployment of changes in code. This tool was used in the project to integrate changes and testing the system after making changes. This was done by linking Travis CI to the GitHub repository, such that every time changes are made in the code and pushed to the remote repository, it is automatically built and deployed to Heroku by Travis CI.

6. Conclusion

6.1 Operation and maintenance

An important aspect of ensuring a smooth operation is error handling. At this stage of the development, although larger errors have been handled, there are still more to account for. For instance, if an ant is added, its coordinates selected, then the same ant is removed, the ant and its coordinates are still sent to the database.

6.2 Future development

Future development goals and features that could not be implemented in this project timeline are listed below:

a. Compatible for different devices and screens
   To make the application compatible and user friendly with touch screens, and different operating systems. Currently, one known bug is that the frame to be labelled does not fill up the video panel when viewed on a Mac device. Additionally, the size of the JFrame and JPanels are fixed. If the window is resized or viewed in full screen, the position and size of the labels will not change accordingly.

b. Game UI
   To gamify the process of labelling ants. This would potentially include a more attractive UI, a leaderboard to show the top contributor/labeller, sound effects, and other features that need to be identified.

c. Speed and performance
   To speed up the request and response between the server and the client. Currently, the connection is done through HTTP protocol and this is very slow as all the data is transferred as text/html. A faster protocol such as Network socket could be implemented to speed up the data transfer.

d. Tutorial
   An interactive tutorial or video should be created to make the labelling process/game easier to understand.

e. More videos and scroll bar
   To display more videos on the landing page, and include a scroll bar to scroll through the videos.

f. Warning for out-of-boundary label
   To warn a user when a labelled coordinate is outside the acceptable range, which may be done accidentally. This would help reduce the amount of inaccurate data.

g. Warning for no submission
   To warn a user when the “Submit” button is not clicked before changing the frame. The user needs to submit their labelled data once a frame is done labelling, before starting on the next frame. Otherwise, all labelled data on the current frame will be overwritten.

h. User experience
When the user goes to the Tracking page with the last labelled frame, buttons should already be initialised depending on the labelled ants

i. Storage space
Currently, the video frames are stored on the heroku server and only 500 MB of data is allowed (after compressing). This does not allow that many video frames to be stored. In future iterations, the storage for video frames could be moved to other cloud storage platform such as Amazon S3[11], which provides larger space and scalable storage infrastructure.

j. General bug fixes and stricter error handling
More tests should be conducted to ensure that the app is bug-free.
E.g. If an ant is added, its coordinates selected, then the same ant is removed, the ant and its coordinates are still sent to the database, even though it should be removed.

6.3 Feedback

Due to the iterative development process, the project updates were frequently communicated to our project customer, Mr Plum. An in-person demonstration was also carried out with Mr Plum to gain feedback on the progress of the project. Overall, the feedback from Mr Plum was positive, and he verified that the application met the major requirements that have been identified at the start of the project. He provided suggestions on how the application can be improved, as well as ideas for future development.

[11] Amazon S3, https://aws.amazon.com/s3/

8. Appendices

8.1 Appendix A: Work delegation

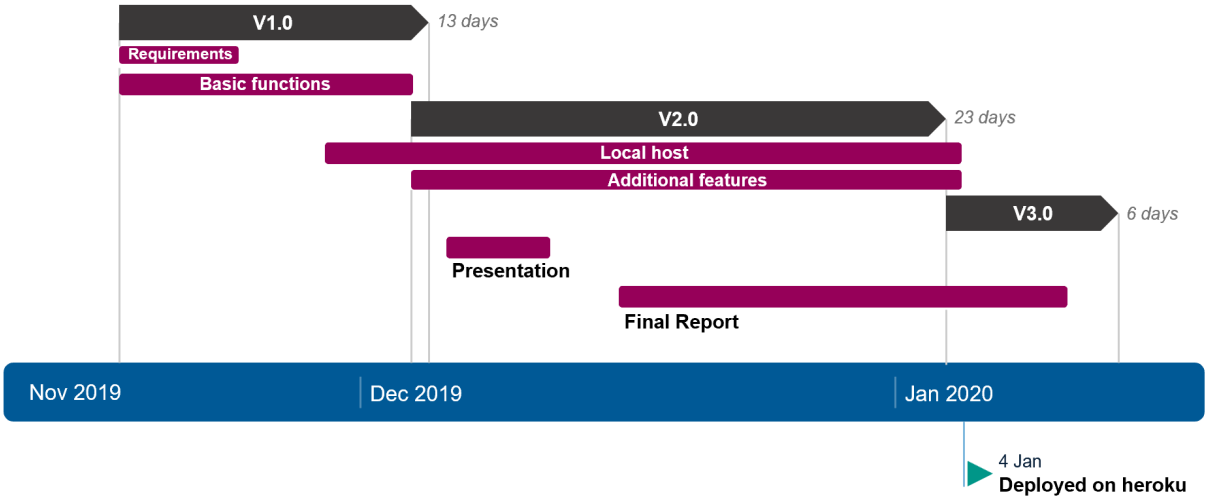
Member	**Scope**
Joshua Sia	UI Functionalities (Ant buttons and coordinates) Video Thumbnails Visual Feedback Page Management Client-server connection Travis CI Heroku Deployment Unit Testing Version Control
Yi Teng Voon	Tracking Page Overlaying of Video Frames Visual Feedback Video Processing
Chanasak Mahankarat	Servlet Database Queries Resource Storage Heroku Deployment Travis-CI Version Control Database Management
Xian Zhang	User Page Landing Page Progress Bar GUI Unit Testing
Cal-Vin Ng	Project Management User Testing and Feedback Database Queries Database Management

8.2 Appendix B: Link to GitHub repository

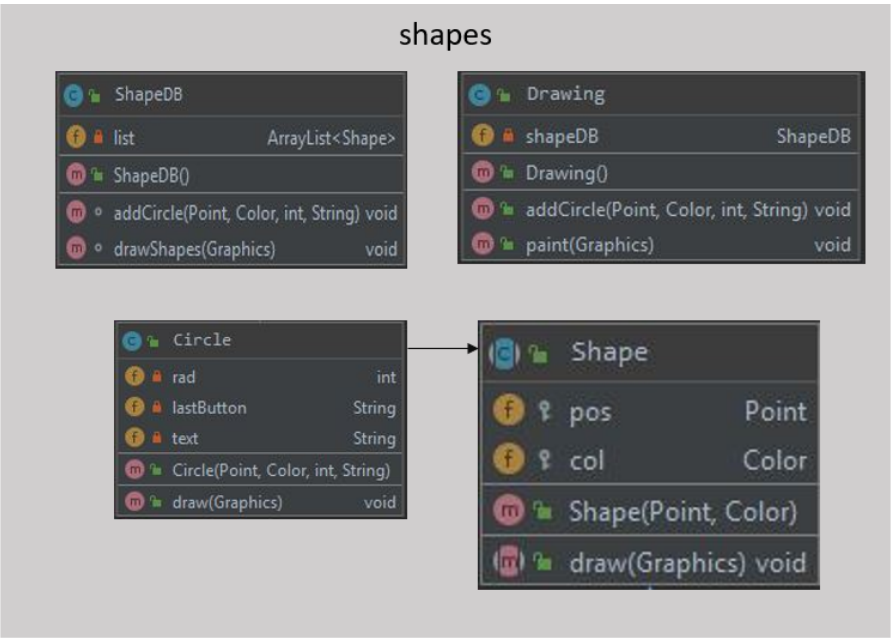
Ants: https://github.com/jgs3817/Ants
Download link: https://github.com/jgs3817/Ants.git

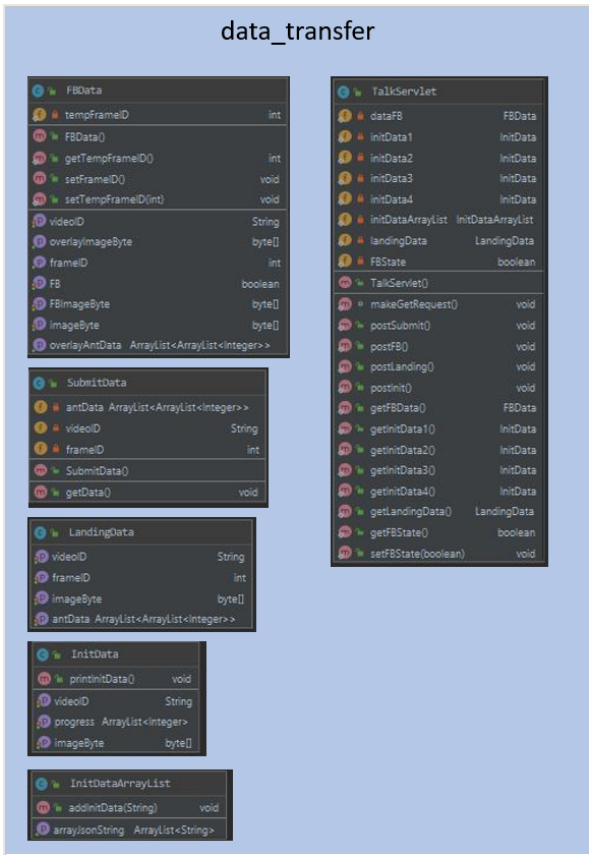
Ants Servlet: https://github.com/jgs3817/AntsServlet
Download link: https://github.com/jgs3817/AntsServlet.git

8.3 Appendix C: Development timeline (Gantt Chart)



8.4 Appendix D: Packages with their corresponding classes




data_transfer
FBData
tempFrameID int
FBData()
getTempFrameID() int
setFrameID() void
setTempFrameID(int) void
videoID String
overlayImageByte byte[]
frameID int
FB boolean
FBImageByte byte[]
imageByte byte[]
overlayAntData ArrayList<ArrayList<Integer>>
TalkServlet
dataFB FBData
initData1 InitData
initData2 InitData
initData3 InitData
initData4 InitData
initDataArrayList InitDataArrayList
landingData LandingData
FBState boolean
TalkServlet()
makeGetRequest() void
postSubmit() void
postFB() void
postLanding() void
postInit() void
getFBData() FBData
getInitData1() InitData
getInitData2() InitData
getInitData3() InitData
getInitData4() InitData
getLandingData() LandingData
getFBState() boolean
setFBState(boolean) void
SubmitData
antData ArrayList<ArrayList<Integer>>
videoID String
frameID int
SubmitData()
getData() void
LandingData
videoID String
frameID int
imageByte byte[]
antData ArrayList<ArrayList<Integer>>
InitData
printInitData() void
videoID String
progress ArrayList<Integer>
imageByte byte[]
InitDataArrayList
addInitData(String) void
arrayJsonString ArrayList<String>


panels
ButtonIDContainer
idPanel IDPanel
ButtonIDContainer()
setGridBagLayout(GridBagConstraints, int, int, int, int) void
LandingPage
header JLabel
startButton JButton
landingFlag boolean
LandingPage()
setGridBagLayout(GridBagConstraints, int, int, int, int) void
getLandingFlag() boolean
setLandingFlag(boolean) void
FBPanel
nextButton JButton
prevButton JButton
submitButton JButton
videoPanel VideoPanel
fb boolean
frameID int
landingData LandingData
FBPanel()
getFBState() boolean
returnVideoPanel() VideoPanel
getFrameID() int
VideoPanel
antData ArrayList<ArrayList<Integer>>
individualAntData ArrayList<Integer>
buttonID int
dataFB FBData
landingData LandingData
indicator Drawing
drawFlag boolean
lastButton String
initButton String
dataLanding LandingData
initIndicator Drawing
VideoPanel()
setDrawFlag(boolean) void
getDrawFlag() boolean
getAntData() ArrayList<ArrayList<Integer>>
fillIndividualAntData(ArrayList<Integer>, int, int, int) void
loadFrame() void
convertImageByte() BufferedImage
convertFBImageByte() BufferedImage
initialImage() BufferedImage
UserPage
startButton JButton
greet JLabel
username JTextField
user String
userpageFlag boolean
UserPage()
setGridBagLayout(GridBagConstraints, int, int, int, int) void
getUserFlag() boolean
setUserFlag(boolean) void
TrackingPage
TrackingPage()
setGridBagLayout(GridBagConstraints, int, int, int, int) void
ProgressBar
ProgressBar(int, int)
IDPanel
IDPanel()
PageHandler
gc GraphicsConfiguration
PageHandler()
VideoFramesContainer
VideoFramesContainer()
MenuVideo
vid1 JButton
vid2 JButton
vid3 JButton
vid4 JButton
vidID int
initData ArrayList<InitData>
bImage BufferedImage
MenuVideo()
setGridBagLayout(GridBagConstraints, int, int, int, int) void
getVidID() String
setThumbnail(ArrayList<InitData>, int) JButton
ButtonPanel
lastButton JButton
addButton JButton
minusButton JButton
backButton JButton
organiserPanel JPanel
backFlag boolean
idPanel IDPanel
antData ArrayList<ArrayList<Integer>>
overlayAntData ArrayList<ArrayList<Integer>>
dataLanding LandingData
count int
ButtonPanel()
getIDPanel() JPanel
getLastButton() JButton
getBackFlag() boolean
setBackFlag(boolean) void