MAGIC MINUTES

AND YET... IT COMPILES

Abby Wysopal,
Olivia Clough,
Denver Simmons,
Cole Bergmann,
Ryan Mitchell
PRODUCT NAME: **Magic Minutes**

TEAM NAME: And Yet... It Compiles

WEBSITE: [HTTPS://MAGIC-MINUTE.ME](https://MAGIC-MINUTE.ME)

SLACK: [HTTPS://ANDYETITCOMPILESGROUP.SLACK.COM/MESSAGES/CHNK9FN07/](https://ANDYETITCOMPILESGROUP.SLACK.COM/MESSAGES/CHNK9FN07/)

GitHub: [HTTPS://GITHUB.COM/COLEBERGMANN/AND-YET-IT-COMPILES](https://GITHUB.COM/COLEBERGMANN/AND-YET-IT-COMPILES)

Trello: [https://trello.com/b/iG3UuHoD/cs48](https://trello.com/b/iG3UuHoD/cs48)

TRAVIS: [HTTPS://TRAVIS-CI.COM/COLEBERGMANN/AND-YET-IT-COMPILES](https://TRAVIS-CI.COM/COLEBERGMANN/AND-YET-IT-COMPILES)

### Team

<table>
<thead>
<tr>
<th>Position</th>
<th>Name</th>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Team Lead</td>
<td>Abby Wysopal</td>
<td><a href="mailto:abbywysopal@gmail.com">abbywysopal@gmail.com</a></td>
</tr>
<tr>
<td>Developer</td>
<td>Olivia Clough</td>
<td><a href="mailto:olivia_clough@ucsb.edu">olivia_clough@ucsb.edu</a></td>
</tr>
<tr>
<td>Developer</td>
<td>Denver Simmons</td>
<td><a href="mailto:denverjamessimmons@gmail.com">denverjamessimmons@gmail.com</a></td>
</tr>
<tr>
<td>Developer</td>
<td>Cole Bergmann</td>
<td><a href="mailto:colebergmann@ucsb.edu">colebergmann@ucsb.edu</a></td>
</tr>
<tr>
<td>Developer</td>
<td>Ryan Mitchell</td>
<td><a href="mailto:ryanm777@gmail.com">ryanm777@gmail.com</a></td>
</tr>
</tbody>
</table>

### Background

Disneyland Parks in Anaheim, California attracts over 44,000 visitors per day. Each visitor arrives at Disneyland with the risk of waiting in line for multiple hours per attraction. Unnecessary stress is imposed upon visitors when waiting in excruciatingly long lines and being surrounded by thousands of people. We believe there should be a way to predict wait times and crowd sizes within each park. Through the prediction of wait times per ride at Disneyland, visitors will enjoy a more efficient day at Disneyland parks. Thus, making the happiest place on earth even happier.

Today, Disneyland currently has a mobile application that contains a variety of features. Disney, however, is a company focused mainly on increasing its profit margins, and therefore the user interface of their application focuses on advertising ticket sales, dining features, and fast passes in the ride time section. Disney has all of the necessary data to predict wait times, but they do not give consumers access to this information because it could negatively impact ticket sales. We hope to avoid this and share accurate predictions.

With their application, they only provide live updates. Therefore, the general public cannot plan their trips or visits accordingly. In our service, we will use machine learning to train an algorithm using data points found from their live updates. We will push ourselves to learn more technologies in order to solve a problem that we find interesting.
PROJECT

Outcome
The outcome of this project will be an application that allows the user to harness our trained models and predict the best day to attend Disneyland based on average wait times and park population density. The user will also be able to use daily predictions to plan when to go on individual rides in the park.

Milestones
The most essential component of this project is creating an accurate machine learning model to predict future wait times for Disneyland rides. Firstly, we will gather relevant data such as weather, wait times and population at each park. Creating a set of data which will be formatted to train our model. We will then research different machine learning frameworks in order to further train our model. As we train the model, we will create the web interface use ReactJS. After selecting a time and date, a graph of predicted wait times will be generated and shown to the user. While our backend data will be handled by either AWS/Google Firebase.

Sprint 1 (4/15 - 4/26)
- a. Gather relevant data to feed the neural network
- b. Format the data for the network to read
- c. Determine a machine learning framework/service to use
- d. Determine the most effective network structure / hyperparameters for our problem
- e. Begin training the machine learning algorithm with our dataset
- f. Create a basic web interface with React JS

Sprint 2 (4/29 - 5/10)
- g. Explore different databases (cloud services, local sql, etc) and pick a solution
- h. Update web interface to link between pages
- i. Use Node JS for backend development and API call
- j. Type in a ride, return a google maps type graph of what the best time to go on it is
- k. Display a google maps type graph for park crowds for the next 30 days

Sprint 3 (5/13 - 5/24)
- l. Finalize the web interface
- m. Update models for increased accuracy
n. Input real time data into ML model

o. Clean up the web app backend in Node JS

**Optional** (if time permits)

p. Add the capability for the user to set a date and select a list of rides they want to go on. The app will create the best schedule.

q. Further improve the model with additional data points if we have time (% of schools in session, etc)

r. Further improve the front end of our website application, and possibly adapt our website application to iOS and Android.
High Level Overview
Detailed Design
UML Diagrams

1) Machine Learning

MyModels
Operations:
- LoadCSV(csv_file)
- Predict(model)

CSV File

Model
Attributes:
- Machine Learning Model
Operations:
- LoadModel(model)
- Predict()

2) Live Data

Live Data

LocalWeather
- dayHigh
- dayLow
- dayPrecipProb
- tempNow
- precipProbNow
+ Constructor
  + getDayHigh
  + getDayLow
  + getDayPrecipProb
  + getTempNow
  + getPrecipProbNow
  + getWeatherNow
  - streamToJson

ParkHours
- dOpenHour
- dCloseHour
- dcaOpenHour
- dcaCloseHour
+ Constructor
  + getDIOpenHour
  + getDICloseHour
  + getDcaOpenHour
  + getDcaCloseHour
  + fetchCurrentHours
  - formatData

CurrentWaitTimes
- rideWaits[]
+ Constructor
  + getWaitTime
  + isDown
  + isClosed
  - fetchLatestWait
  - parseRide
3) UI Today Page

```
+ Props
+ State

+ Constructor
  + Render
    - handleChange(event)
    - handleSubmit(event)
```

4) UI Plan Page

```
<<interface>>
Component

Plan

PopulationGraph

+ Props
  + next30daysPop []
  + State

+ Constructor
  + Render
```
Sequence Diagrams

1) Interaction between Classes

Class: Plan

Class: PlanGraph

Class: Database

- 1.1 Import PlanGraph
- 1.2 getCharData()
- 1.3 Return data array
- 1.4 Render() graph

2) Interaction with User and System

User

Web App

API

ML

1. Open Web App

1.1 Display Home Page

2. Select Today Link

2.1 Display Today Page

3. Select Ride

3.1 WaitTimeData

3.2 Real Time & Predictive Data

3.3 Return Data

3.4 Array of Wait time Data

3.5 Display Ride Wait time graph
UI Mockups

Welcome to Magic Minutes

Disneyland Park

Population

Months

Park crowds for the upcoming month:

Population

Day

Please choose ride:

Ride Name

Wait Time

Hour
Feature Cases: User stories

Web App:

1. Annual Population
As a web page visitor, I want to see which month out of the year is the most crowded.
Scenario 1: Annual Population
Given I am a user
And I want to see the different month’s population Disney
When I navigate to the home page
Then I will see a graph comparing the crowds for each month of the year

2. Homepage Options
As a web page visitor, I want to select a time range in order to see graphed data of wait times or population at Disneyland
Scenario 1: Web Page formatting
Given I am on the website front page
And there is a selection of today or planner
When I select today
Then I will be directed to a new web page to choose my ride
And when I select planner
Then I will be directed to a new web page to help plan my Disneyland trip

3. Ride and Times
As a Disneyland Park visitor, I can see the ride wait times for today so that I can plan the best time to go on the ride.
Scenario 1: I am in Disney today only.
Given the date is today,
And the ride is Space Mountain.
When the user chooses the ride
Then the graph will show me the wait times,
And I will be able to pick a time to go.

4. Date Range and Crowds
As a User, I can choose to plan my trip so that I can see what days in the next month are the busiest.
Scenario 1: I want to go to Disney in the next month.
Given the user chooses planning page
and the range is at most 30 days,
When the user clicks Search
Then a graph of the park population for the next month will appear.

5. Web App Backend to access ML model
As a website, I can connect to a server and access the ML model so that I can display the results of the model.
Scenario 1: Website calls model
Given the website calls the model,
and the model is hosted correctly,
The model can take inputs from the website
and provide the website with the model’s output.
ML Network:

6. **Transform data into segments**
   As a machine learning bot, I have the relevant historical ride data in the format I need.
   Scenario 1:
   Given a database of historical ride data,
   When the Java program is run,
   Then a csv file with all the historical data points for Disney rides are outputted
   And they are in the correct format to be fed into the model training.

7. **Trained Neural Network**
   As a programmer, I can implement the trained neural network into my software so that it actively predicts the wait times of a ride over the current day when fed park data in real time.
   Scenario 1: I am programming software that uses predictive modeling
   Given that I have real time disneyland historical data
   And I input the real time data into the network
   The network outputs ride wait predictions for the day

8. **Collect Data Points**
   As a programmer, I can gather valid points of data so that the TensorFlow model can run accurately.
   Scenario 1: API setup is correct
   Given the data collection is exhausted
   And the programmer has formatted it correctly
   When the ML TensorFlow is ready
   Denver and Ryan will be able to input an API

9. **Connection between back and front end**
   As a web page interface, I will be able to call python scripts to display the ML model data.
   Scenario 1: HandleSubmit
   Given I am a web page
   And I want to display the model information
   When a button is pressed, I will call the python script
   Then I be given an array style API of predictive data

10. **iOS App**
    As a user, I will be able to open a native iOS app and view the previous/real time predictive data for each ride on a graph
    Scenario 1:
    Given I am on the iOS app
    And the ride is Space Mountain
    When the user chooses the ride
    Then the graph will show me the wait times,
    And I will be able to pick a time to go.
Appendix: Technologies Used

ReactJS
Tensorflow
Java, Python
MySQL
Travis-CI
Selenium
Swift
Ubuntu
Firebase

Sprint Progress

Sprint 1

At the end of sprint 1, we have a working ML model for one ride. This is set up in a way to be able to change one number in order to change the ride (CSV is formatted correctly for this as well). We also have a running UI with all components that is hosted locally.
Sprint 2

In sprint 2, we focused on getting the model up and running for all 10 rides. We also succeeded in connecting the front and back end to perform fetches to the ML output in order to render graphing results.

Sprint 3

In sprint 3, we made the graphs more user friendly by showing the wait times for the entire time the park is open by time of day. In addition, we introduced a real time data component so the graph up until the current time of day is accurate data and the future is the prediction. Finally, we created an iOS app that shows the ride time section of our web app to be used on the go as you are in the park.
Retrospective Documentation

Sprint 1

<table>
<thead>
<tr>
<th>Pro</th>
<th>Con</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Really good start</td>
<td>- More accurate acceptance tests</td>
</tr>
<tr>
<td>- We are meeting in person 5 days a week</td>
<td>- Should use github more</td>
</tr>
<tr>
<td>- ML model is accurate!!</td>
<td>- Spreading out work each day instead of putting it off to one</td>
</tr>
<tr>
<td>- Working well together to discuss problems that come up</td>
<td>long day of work during busy times</td>
</tr>
</tbody>
</table>
| - Willing to help with any piece of the project                     |   - **Be more involved with all of the pieces of the project instead of just**
| - Learning a ton of material while still getting things done        |   **the one you’re focusing on**                                  |
| - Things are running, although separately                           | - Tasks were not in the best order                                 |
|                                                                      | - Everyone must be present for sprint planning                    |

After voting for what con was the most important to work on for the next sprint, we decided on: Be more involved with all of the pieces of the project instead of just the one you’re focusing on

Sprint 2

<table>
<thead>
<tr>
<th>Pro</th>
<th>Con</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Working together as a group</td>
<td>- Need to make sure the model is correct when fetching the data</td>
</tr>
<tr>
<td>- On track for finishing without stress of packing too much into the</td>
<td>from JS</td>
</tr>
<tr>
<td>last sprint</td>
<td>- Time management between this and other classes</td>
</tr>
<tr>
<td>- Scott was impressed with our work</td>
<td>- Bug detection wasn’t our strong suit this sprint</td>
</tr>
<tr>
<td>- We got a functional navbar</td>
<td>- <strong>All tasks are reliant on each other, need to stay on top of</strong></td>
</tr>
<tr>
<td>- Minimum viable product done after sprint 2</td>
<td><strong>them to continue working</strong></td>
</tr>
<tr>
<td>- Magic-minutes file is clean!!!</td>
<td></td>
</tr>
</tbody>
</table>

After voting for what con was the most important to work on for the next sprint, we decided on: All tasks are reliant on each other, need to stay on top of them to be able to continue working
Sprint 3

<table>
<thead>
<tr>
<th>Pro</th>
<th>Con</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Able to tie together all of the loose ends</td>
<td>- Last week didn’t have much work due to completing our tasks during prior sprints</td>
</tr>
<tr>
<td>- We completed an optional user story → iOS app</td>
<td>- Didn’t get optional Android app done</td>
</tr>
<tr>
<td>- Low stress</td>
<td>- Able to focus on the demo and final write up</td>
</tr>
</tbody>
</table>

After voting for what con was the most important to work on for the next sprint, we decided on: Developing an android app

Challenges

Throughout the six weeks of developing our project we encountered many obstacles. During our first sprint, we quickly learned how critical sprint planning is. The division of tasks needed to be done very thoughtfully in order to maximize progress and efficiency. Due to inefficient division of tasks, some people had an excessively large workload while others were looking for something to do. Another obstacle we overcame was a large learning curve. Our team had never used many of the technologies needed for this project before (such as tensorflow, reactJS, etc.) As a result, we spent the majority of sprint one learning these technologies and implementing them into our project.

During sprint two, we encountered challenges while trying to connect different pieces of the project together. Upon attempting to run our software on the server, we found that the server was too slow and lacked the necessary memory space. To solve the memory issue, we broke down the computations into small enough pieces for the server to handle. However, this caused the runtime to increase drastically. The server needed to perform one prediction cycle every 10 minutes, but instead was taking around 11 minutes. Denver was able to fix this problem by optimising his code and was able to bring the prediction cycle time down to just 3 minutes. Ultimately, we were able to work together to solve the issues that arose while developing our project.

Future Additions

We have additional improvements to add to our project in the future. We want to make a mobile application for an android phone similar to our current IOS application. In addition, we will add more rides to the wait time section and include the park California Adventure. A major addition to our project is the capability for the user to set a date and select a list of rides they want to go on. The user will be able to select the best schedule for their day at Disneyland. They will be provided the best time to go on each attraction and directions between the attractions ultimately creating the best schedule for the user. We also
want to give the program the ability to re-train itself each night. This would allow the models to become even more accurate over time. Finally, we hope to continuously improve our machine learning model through adding more data points (schools in session, traffic levels, etc).

Git Commits

Github commits pdf

Testing

We used Selenium to test our front-end web application. This was beneficial as we were able to test individual structures as well as the app as a whole. Selenium gives output to show that buttons, links, and other pieces work as expected. In addition, we were able to ensure our react-router-dom links were functioning properly by taking the user to the correct new page URLs as well as pushing onto the browser history.

Also, We created unit tests for the Java program that runs every 10 minutes to provide our program with the latest park data. We test the class that gathers weather data to make sure it is returning a valid response. We test several of the data classes by sending valid or invalid parameters and ensuring the .toString matches what we expect for the given input. These test were added to Travis CI to run every time we push to GitHub to make sure we never break anything mission critical. Our tests for this program are as follows:

- CSVManagerTest: Tests creation, writing, reading, and format of a CSV file as created by our CSVManager class
- CurrentWaitTimesTest: Tests instantiating the CurrentWaitTimes class that reaches out to Disney and populates with the wait times of all the current rides. We test the response for an invalid ride ID, but we do not test a valid ride ID as these values are subject to change from Disney’s end
- DisneyRequestsTests: Tests getting an API key for Disney and requesting data from one of their known valid endpoints. We also try fetching data from a known invalid endpoint from Disney and make sure we handle that error correctly. We also test to make sure we throw the proper error if Disney sends us invalid data.
- LocalWeatherTests: Tests our LocalWeather class that fetches the current weather from the DarkSky API. Tests pass as long as we get a positive value for the daily high, low, and current temperature.
- ParkHoursTests: Tests an invalid case to make sure we throw the proper error. We can not test the success case because we can not control what Disney returns to us.
- TableEntryTest: Tests the data structure for our TableEntry class. Ensures the .toString for TableEntry is correct with several inputs. Also ensures we throw an error when necessary.

The tests for this program are not entirely comprehensive because several classes depend on a response from Disney and change their outputs accordingly. Because of this, it is unreasonable to test the success cases for many of these classes.

In addition, We tested the model accuracy using RMSE (Root Mean Squared Error). If the
RMSE of the model prediction is lower than the RMSE of always predicting current values, then the model is successful. Finally, to test if the correct data was being served to the API, we compared the data in the API with the data in the database and the real time data from the live csv. We also used standard output messages to communicate successful connections to the database as well as communicating whether the API is in standby mode or if it is updating.

UI testing:  
https://github.com/colebergmann/and-yet-it-compiles/blob/master/selenium%20testing

Live Data testing:  
https://github.com/colebergmann/and-yet-it-compiles/tree/master/data%20programs/LiveData/src/test/java

API testing:  
https://github.com/colebergmann/and-yet-it-compiles/blob/master/backend/testPredDB.py

ML testing:  

Presentation PDF:  
Final Presentation Slides on Github  
Final Presentation Slides on Google Drive

Demo:  
Final Demo