Intuitive Intel

Instantly relevant tips

Johnny Byrnes
Garrett Keefe
Nanda Kumar J
Alec Parent

October 21, 2022

Table of Contents

Executive Summary

We want to streamline learning during gaming, whether it is the mechanics of a new game or lineups on a familiar one. Typically, learning new skills in games is bogged down by the need to seek information outside the game. The information to improve is typically found in videos and perhaps even coaching on the specific game. These two sources can either be inadequate, elusive, or even expensive. Our project is designed so that users can receive free, in-game information about how to level up their gameplay!

Our primary users will be gamers who wish to improve. These gamers will launch Intuitive Intel alongside whatever game they feel like. They will be able to set a skill level and style of tip per game in the app's settings as well. When they are ready to begin, they just tell Intuitive Intel that they are searching for a match. Once they find a match, Intuitive Intel will automatically determine specific information about the match, such as character selection or which map it is on. Then it will find some tips that the player will be able to use in the upcoming match, such as map-specific lineups or character interactions. There will be an option to go to the community post about that specific tip in the app's complimenting online forum, which will be helpful for more complicated tips, or ones that require images. At the end of the match, our app will ask users to rate the tip to help improve the system.

You may be wondering where these tips come from. This is where our other type of user comes in: the content creator. Content creators will be able to create their tips for any other users or their followers to see. These users will have to manually tag which game, map, and character the tip should be associated with. Once a tip is uploaded, it can be accessed by anyone with Intuitive Intel. Our community forum will allow users to rate and request tips directly. To help facilitate this community interaction, we will have a website where creators can see their tips and users can discuss with each other about different games!

Map: Oasis: Gardens
Hero: Lucio
Tip: High ground on the center is NOT part of the control point, but wall-riding on its pillar will contest

Background and Technical Requirements

Idea Space

One critical problem with the current gaming industry is that improving at a game can be exceedingly frustrating. People can spend hours on a game without seeing improvement, and that can become very frustrating for a user. Our project aims to take out one of the problems that plague avid gamers, scarcity of relevant information to improve. Our project solves this issue by providing up-to-date and relevant tips to users so that they can instantly start to incorporate the advice into their gameplay.

Our project is somewhat similar to Overwolf and Youtube, but these platforms present key issues:
- Youtube allowed ratings as a way to filter good and bad content, and many content creators would post videos or compilations for a game that provides learning content for the watcher. It has since disabled the ability to rate videos which leaves it without a way for users to filter content effectively
- Overwolf tracks in-game stats and gives basic tips, but it doesn't adapt to the current match or allow content creators to upload/post.

Intuitive Intel is better for end-users because it guarantees a relevant tip during the time that the game is being played. It is also better for content creators because it allows them to update tips when they become outdated.

Technologies

We are currently using python as the application language, and these respective python libraries: easyocr, tkinter, pyautogui, psycopg2, PIL, Flask, AWS, and PostgreSQL

> PYTHON: We chose python because our app is relatively high level, so we won't require much low-level architecture. Python has an extensive library of free-to-use and well-documented add-ons that can provide much of the functionality we will need for this project.

> EASYOCR: We chose easyocr because it is well documented, has frequent updates, and is free. This is used to do our screen reading, and detect which map it is.

> TKINTER: We chose tkinter because it made it very simple to make a window and give buttons functionality.

PYAUTOGUI: We chose this because it made achieving screen grabs very simple.

PSYCOPG2: We chose this for database querying and other database-related needs. The library has many online examples and is easy to use with a background in manipulating databases.

PIL: We chose Python Image Library because it has a very simple way to translate the image from pyautogui to something easyocr could use.

FLASK: We are using the Flask framework with python to build our web application. Flask is easy & flexible for building web pages and also connecting to the remote database. It avoids the complexity of project structure that we see in many frameworks like .NET, React, etc.

AWS: We are using AWS to host both our database and our website.

POSTGRESQL: We choose postgres as our database language of choice as we have worked with it before and were able to find a readily available docker container that implemented it.

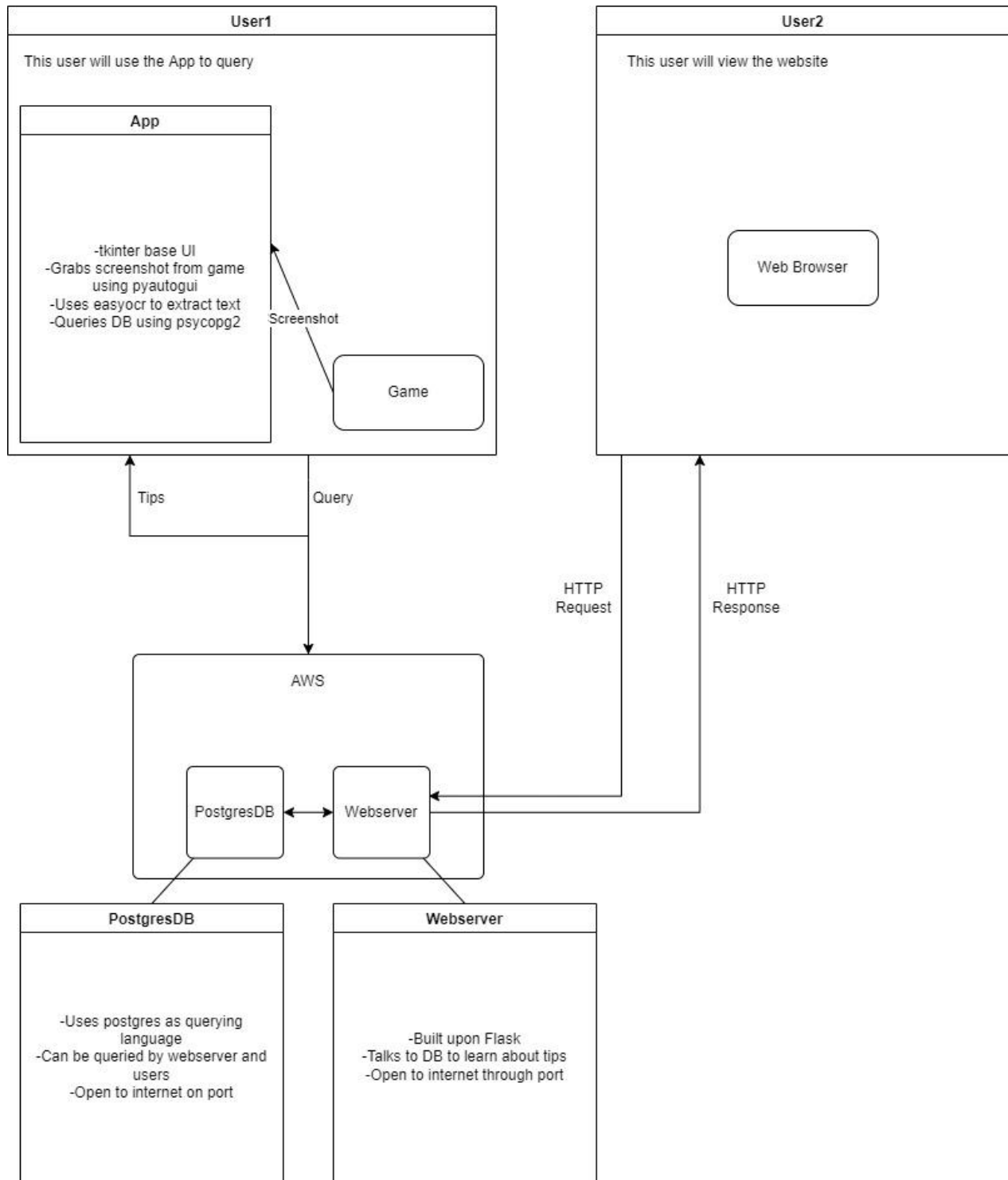**Software/Hardware User Requirements**

Users will need -
  a) Windows operating system
  b) A computer that provides adequate specifications to run whatever game they want.
  c) Enough free space to download our app and provide room for the app to take and store temporary screenshots

The users will not need much beyond the 3 simple requirements here, which is intended, as we don't want our app to be very technically complex or restraining to the user.

Requirements Analysis

System architecture

User1
This user will use the App to query

App
-tkinter base UI -Grabs screenshot from game using pyautogui -Uses easyocr to extract text -Queries DB using psycopg2

Screenshot

Game

User2
This user will view the website

Web Browser

Tips

Query

HTTP Request

HTTP Response

AWS

PostgresDB

Webserver

PostgresDB
-Uses postgres as querying language -Can be queried by webserver and users -Open to internet on port

Webserver
-Built upon Flask -Talks to DB to learn about tips -Open to internet through port

App

We will use tkinter for our UI and buttons. The UI will be the user's ability to interact with our app, and the buttons will link it to each other options they have. Each button has its own purpose where a user can "Find a match", "Click to navigate to website", and finally "Cancel search".

We will use pyautogui to take screenshots. This is activated within a loop whenever the user presses the "finding a match" button. This loop is cancellable by the user pressing the "cancel match" button.

After each screenshot is taken, we use PIL to translate it into a format that easyocr can use. Then our app will use easyocr to determine the map/character. When we find a map/character, we return it to the UI. We display it on the screen for the user to know (and double-check if the app made a mistake).

After finding the current map/character, we then use psycopg2 to query the postgreSQL database located on AWS for relevant tips for our game, map, and character tuple.

Finally, after receiving the tip, we can now again use tkinter to display the tip to the user. A separate window will pop up and display the tip and then disappear after a reasonable amount of time.

Community Forum

Our Community Forum will run on AWS where the web server can receive and respond to HTTP requests. We will use Flask to implement our website where we can handle all sorts of different web requests that will allow for our necessary functionality. This community forum is responsible for creating users and giving them access to all the resources we offer. In this forum, we are going to implement numerous game discussion rooms where they can discuss a specific game. Users are also given access to create a new room if they don't find the game room they wanted. Inside each game room, we are adding a thread modeled discussion where users can reply to each other, like their tip information, and contribute game tips to the community by typing **"#TIP: this is a tip".** Our flask app will detect the tip added by the user, pull it from the forum and add it to the database.

Database

We will have a database hosted on our AWS RDS instance where we can properly access our tables from both our web server and end-users host systems. This database will run off of PostgreSQL and will be maintained by an admin.

Personnel

Johnny: UI. I have taken graphics (CS 4600) and interactive graphics (CS 6610). I'm comfortable with multi-threading and understanding user flow which will be important in our app. I will make the UI for users to interact with and program the buttons.

Alec: Database (backend) hosted and managed on an AWS RDS server as well as database interaction with the forum website. I am comfortable with architecting databases and keeping them performant as this will be my main focus for the project. I have taken Databases (CS 5530) which will be my main resource, Data Mining (CS 5140) which taught me how to deal with data appropriately, and Networking (CS 4480) which taught me how to keep networks performant.

Nanda: Designing a community forum website. I have started making a community forum app using Flask & Python. I am going to build a simple forum where users can interact with each other and contribute to the database. I have taken Databases (CS 5530) & CS 4530 (Web software architecture) which can help me in this area of the application.

Garrett: Database interaction with the app and assisting Nanda in developing the community forum. I've taken both Networks (CS 4480) and Database Systems (CS 5530) which prepares me for both of my jobs.

System Features

Bare essentials:

      B1) Overlay UI to display a basic text version of the tip

      B2) Database to store tips on AWS focused on highly performant reads

      B3) Setup React App and express

      B4) Users can create tips

      B5) User tips go to database

      B6) Users can do all database tasks through the UI

Planned Features:

      P1) Add support for multiple games

      P2) Users can set their skill level per game

      P3) Users can add images to tip (decided not likely, swapped for w10)

      P4) Rating system for received tip

      P5) Forum allows for chat rooms

      P6) Forum displays users past tips (More polishing needed for UI pages)

      P7) Improve database to be robust and useful for more functionality

      P8) Users can rate tips within UI

      P9) Discussion section for each thread (we decided this was a sub-task of P5, removed from timeline)

      P10) Users can login to UI

      P11) Users can see the current overall rating of the tip they recieved

Bells and Whistles:

      W1) App gives multiple different tips per match

      W2) Rate tips in forum

      W3) Follow specific users (browsing decided not as important)

      W4) Searchable user-defined tags

      W5) App launches automatically when a game is launched

      W6) Add generalized support for any game (end user uploads)

      W7) Different causes for rating: relevance, skill rating accuracy, etc.

      W8) UI displays images directly (decided not important, polishing instead)

      W9) Add generalized support for any game (end user uploads) directly from UI (pushed back for W1)

      W10) Different reasons for rating: relevance, skill rating accuracy, etc. in UI

Software Engineering Tools and Techniques

- **Agile:** We followed agile methodology (not as strict as it is done in organizations) by meeting every Monday, sharing the progress on a task, and when a teammate explored a new/easy way of achieving some task we all would jump in a meeting to narrow it down and make an agreement if we should choose that path or not.

- **Versioning:** Versioning is also a huge part of any project management, all the team members used Git to host versions of their respective parts in the application and they were versioning for every checkpoint with meaningful commit messages and a small description of what they have done. We all will continue to follow the same structure we have been doing so far with versioning.

- **Sprint planning:** We have not necessarily followed a sprint basis to accomplish small pieces of the project but we kept it open-ended as things we are working on kept changing often depending on the needs of the project but we all agreed that we need to be able to make some prototype to see if that fits in our project within every two weeks. In the project class, we are looking forward to having a strict timeline with a 2-week sprint where each developer will be assigned two tasks with at least 3 to 5 story points.

- **Video conference tool:** Coming to communication, we all chose to perform our video conferences on discord as it is handy to share videos and communicate with teammates. Discord is flexible with starting a meeting compared to zoom and we really felt that it is easier to start a meeting with a single tap.

- **Kanban board:** We decided to use the Git kanban board as it is free to use. We assign tasks at the main repository and track the bugs, developments, and any other discussions related to the task tickets using this tool. This tool is effective and we will continue using this.

- **Meetings:** Moving forward, we all would like to meet at least 3-4 times a week since the actual development of the application will happen, and having meetings on a daily basis will really help our team make progress within a short time. We would like to have our stand-up meetings flexible according to our school schedule but a short 30-min meeting will be good enough to go over each developer's progress/questions.
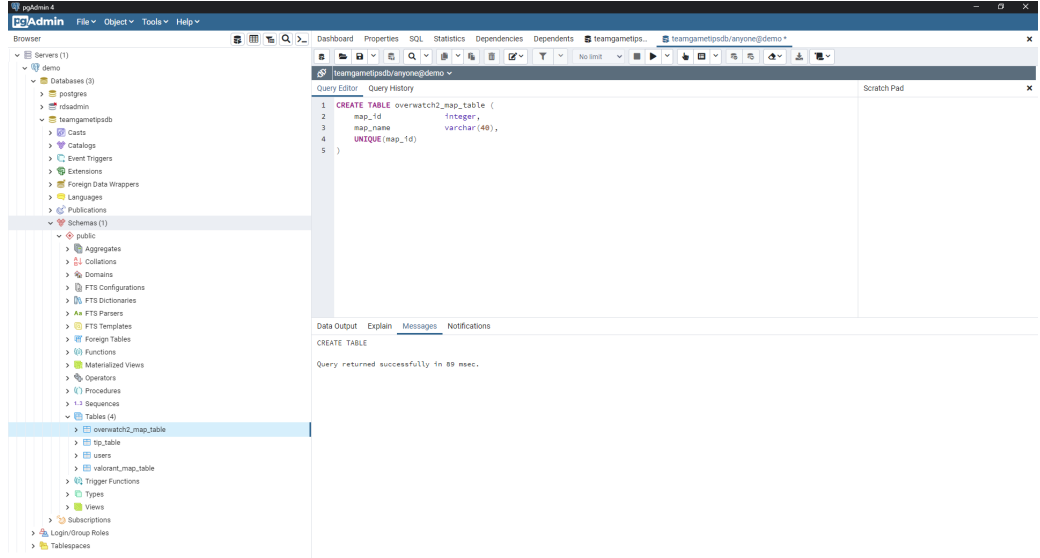
Timeline

Item	Due Date	Person Assigned
B1	8/27	Johnny
Design Doc update 1	9/9	Alec
P1	9/11	Johnny
B2	9/11	Johnny
Alpha Demo React Website	9/30	Nanda
Alpha Demo App	9/30	Johnny
B4	10/9	Johnny
B5	10/9	Johnny
B6	10/15	Johnny
P2	10/16	Johnny
P6	10/16	Nanda
P4	10/16	Alec
User Guide 1st draft (App)	10/21	Johnny
User Guide 1st draft (React Website)	10/21	Garrett
Design Doc update 2	10/21	Alec
P10	10/23	Johnny (accomplished by Garrett)
Beta Demo (React Website)	10/26	Nanda
Beta Demo (App)	10/26	Johnny
B3	10/28	Garrett
P8	10/30	Johnny
W3	11/13	Johnny
W6	11/6	Alec
W7	11/6	Nanda
W5	11/10	Garrett
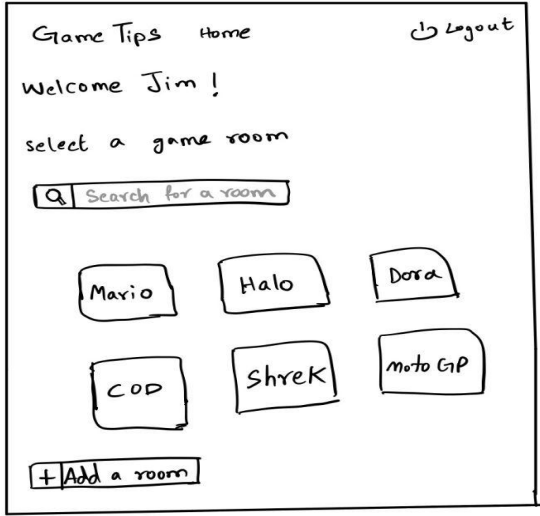
W1	11/13	Johnny
P7	11/17	Alec
W9	11/20	Johnny
P11	11/20	Johnny
P5	11/26	Nanda
W10	11/27	Johnny
Final Design Doc	12/2	Alec
Final User Guide (App)	12/2	Johnny
Final User Guide (React Website)	12/2	Garrett
Final Demo	12/9	Everyone

Appendix A & B: Use Cases and UI Sketches

Admin/System-side Use Cases:

Number	Use Case S1
Title	**Supporting New Games**
Description	The admin directly connects to the database via a GUI tool to add support for new games. In this case, a table for maps in Overwatch 2 is being added.
Steps	1. Load up pgAdmin and connect to the online database 2. Open the query tool 3. Use PostgreSQL to create a new table query in the database 4. Run the query with the arrow button at the top
UI	

User-side Use Cases:

Number	Use Case U1
Title	**Choose a Chat Room/Add a Room**
Description	The user can choose a game room when they are successfully logged in or add a new game room.
Steps	1. Login using credentials and go to the home screen. 2. On the screen you can see the numerous chat rooms where you can select one. 3. If not, you can create your own using the "Add new room" option. 4. Also, you can use the search option to find a chat room easily.
UI	

Number	Use Case U2
Title	**Find Tips For Map**
Description	The user can use the App to find tips for specific maps.
Steps	1. The user starts searching for a map in-game and selects the finding a match option on our window. 2. The user waits till a match is found, the app will be taking screenshots and reading them until a map name is found. 3. When a map name is found, our app will query the database for a tip pertaining to the map. 4. The user can then read the tip displayed by our app and utilize it in the match!
UI	