List of features according to the project plan

1. Main Menu
    a. 1st Half Brenner: Created for the first two project milestones, it had a basic version of a main menu that was the first thing the player saw when the game started. Custom made its U.I. (designed images, animations, fonts, layout, and scripts). Users could play a quick match, create an account or log in, check their account information and stats, change settings, or view a placeholder tutorial menu. Logged in users can also log out of their account.
    b. 2nd Half Leo: In the second semester, it was updated to include better error messages in the login and create account systems, there was background music that the settings actually controlled, and a reset password feature was added as well. Also, users are emailed when they successfully create an account.
2. Deck selection
    a. Leo: players have to choose their deck color in the menu before the game starts and then save the selection for when the other scene starts. Original card database for each deck color.
    b. Brenner: Multiplayer version: Needed clients to tell the server what colour they chose in order for the server to mix both client's decks together and make the main game deck and deal the client their cards. Created PlayerManager.cs and ShareVarManager.cs with help from Carddatabase.cs and PlayerDeck.cs in order to populate the selected game decks to shuffle and start and game the scene properly.
3. Settings Menu
    a. Leo: Sound control for background music. Video setting UI.
4. Login/Create Account/Logout
    a. Brenner: Original sql table for user account information, original login system and create account system with extensive testing (Unity scripts connect to php files that query the sql table that's hosted locally using XAMPP),
    b. Leo: Added login and create validation, error message, icons. Setup email server for welcome emails.
5. Password Reset & Encryption

    a. Leo: Password encryption SH256. Users can receive reset token in their email and use it to reset their password with extensive error validation.
6. Account Statistic Tracking & Display (wins, losses, etc.)
    a. Brenner: Users can view their account stats including games won, games played etc.
7. Edit Account Information & Delete Account
    a. Leo: Added a button in accounts page to delete account with a confirmation popup
8. Decks of Cards (12 unique cards and 8 neutral cards that are in each deck)
    a. 4 Deck of Cards (originally in cardDatabase.cs, but now in sharedVarManager.cs)
    b. Projecting card information onto card prefabs from sharedVarManager.cs (dbDisplay.cs)
9. Quick Match (Can join and play a match that enforces the rules and knows who wins)
    a. Matchmaking (Brenner)
10. Game Mechanics (health system, resource system, etc.)
    a. Pre-multiplayer implementation (Adrian):
        i. Implementing game actions and translating rules from paper to code: The game would display each player's health and resources. Each turn, resources would increase and determine which cards were playable. Determine and implement abilities of certain cards when played. Determine which cards could attack each round. Implementing features to allow cards to attack and be played.
        ii. Creating assets: Creating the prefabs for game objects on the game board (not to be confused by assets for the main menu). Attaching scripts to prefabs and attached UI/GameObjects to the scripts.
    b. Multiplayer implementation (Brenner):
        i. Translating non-multiplayer code to suit multiplayer:
11. Creating card database
    a. Leo: Original card database in cardDatabase.cs that populates the player’s selected deck
    b. Brenner: Deck color selection integration and then multiplayer integration in sharVarManager.cs
12. Game over scene

    a. Scene triggers when a player is out of health, navigates back to main menu and updates player stats for each player (games played, games won, damage dealt).

13. Developing card interactions/abilities (Adrian)
    a. Translating abilities cards have from paper to code. When players play a card or take a certain action it should trigger certain abilities automatically. Each card has a unique ability that interacts with different elements of the game such as number of cards drawn in a turn and health gained/lost.
14. Web Admin (Rule's page):
    a. admin can add, delete, edit the rules in rules page.
    b. the buttons are located on the bottom of each card, the modal will be triggered if button is pressed
    c. there's user input validation (both in front and backend)
    d. existing rule's order will be shifted up or down depending on the user's new input
15. Web Admin (Card's page):
    a. can perform operations like rule's page
    b. there's user input validation (both in front and backend)
    c. each image in the card is dynamically shown by storing the image's path in mongoDB database
16. Web APIs:
    a. the client side / web front end is connected to the database and logics using API call.
    b. API is created using Node.js and Express.js.
    c. there are also validation checks on each API logic
    d. database is using mongoDB
17. Web Deployment:
    a. web (front and backend) are deployed live using Vercel
    b. auto deployment on each changes in 'website' branch
18. Game Deployment:
    a. Unity game will be built and uploaded to simmer.io for live deployment
    b. the game is embedded in our website for user to interact with.
19. API testing:
    a. API testing is made using Jest (javascript testing framework) and Postman
20. Web UI testing:
    a. made using Selenium testing

21. Card Artwork:
    a. made using BING AI,
22. Game Mechanic Tutorial

23. Unity Testing Framework:

Technology stack as mentioned in the project plan

1. Web Browser hosting and accessibility:
2. Unity and unity WebGl
3. Multiplayer networking
4. Unity testing framework
5. SQL databases to hold information
6. Web browser use React, Node, Express, MongoDB
7. Web browser use Vercel for live deployment
8. Unity, Netcode, and Photon use C#, and MySQL Workbench uses SQL.

Bonus stretch goals mentioned in the project plan

1. Friends List
2. Join/host friend games
3. Spectator Mode
4. 4 decks
5. Custom deck picker
6. Placeholder Art (cards and battlefield, etc.)
7. Custom Art (all cards and battlefield, game music, etc.) A.I. can be used to lighten the workload if needed.
8. Quick Match Emotes

READ ME Draft:

Color Break

Description

……

Technologies Used

- Web App: MongoDB, Express.js, React.js, Node.js
- Game: Unity and Mirror Networking for Unity, PHP

Setup Instructions

Frontend (Web App)

1. Ensure you have Node.js and npm installed on your machine. If not, you can download and install them from Node.js official website (https://nodejs.org/en).
2. Navigate to the client directory: cd /Group Exploration/Henry's Exploration/Web/client
3. Install dependencies: npm i
4. Start the development server: npm start

Backend (API and Server)

1. Ensure you have Node.js and npm installed on your machine. If not, you can download and install them from Node.js official website.
2. Navigate to the server directory: cd /Group Exploration/Henry's Exploration/Web/server
3. Install dependencies: npm i
4. Create a .env file in the server directory with the following variables:
5. PORT=<port_number>
6. MONGO_URI=<your_mongodb_uri>
7. Start the server: node index.js

Note: Make sure you have Node.js and npm installed on your machine.

Database (XAMPP)

1. Download and install Xampp (link here:)
2. Download the “sqlconnect” folder from the main repository, which should contain the backend php files and the useracc.sql database file. (add link to folder)
3. Place the sqlconnect folder in XAMPP’s “htdocs” folder to host its php files on your localhost.
4. Go to phpmyadmin, set up a new schema and import the useracc.sql file to create the user account table. Xampp will host this schema and table, and the php files in the sqlconnect folder will now be able to query it.
5. Launch Xampp and run the Apache and MySQL servers.

Unity (Game Dev)

1. Setup a Unity account and download Unity Hub.
2. Download Editor version 2021.30.3f (might have to check archive versions)
3. Clone the 499UnityGameT19 repository and import it as a new project in Unity Hub (add link)
4. Repo should automatically come with Mirror, parallelSync, and other packages required for multiplayer networking already downloaded in the project.
5. Open project in Unity Hub once imported, go to clone manager, and click on “open clone in new editor” to run another editor simultaneously.
6. Open the main-menu scene on both instances of the game

   To start a multiplayer game:
7. Start the main menu scene in each editor (press play at top of editor window) and select “quick match” (then pick any of the four deck colours). After selecting a colour, the actual game scene will start.
8. In the top left, there will be a network manager interface. First, on one editor, click “HOST” to run it as a host (server + client). Click “CLIENT” on the other one to have it connect to the host editor (clients will fail to connect if there is no host running yet).
9. Click “Deal Cards” in both editors once they are either a host or a client, and the server will mix the players' colours together into the game deck, and deal each player 3 cards from it. The editor who clicks “deal cards” first will become player 1 and the other editor will be player 2 (server assigns who is who, and tracks turns accordingly).
10. Each player will take turns playing cards from their hands (if they can afford their cost), and attacking each other using cards they have played (drag card into opponent’s health icon), until one player runs out of health.

    To start a tutorial:
11. Start the main menu scene in an editor.
12. Select “Tutorial” from the main menu, and it will start the tutorial scene.

    To play the game using an account:
13. Ensure the XAMPP is set up and RUNNING (see above Database setup).
14. Start the main menu scene in an editor, and choose the “Login” option. Create a new account using a valid email (will be sent a “welcome” email, and can be sent a “reset password” email), will be automatically logged in upon creating a valid account, and will be able to log in using the same credentials unless the account is manually deleted from phpmyadmin table.
15. Once logged in, click “quick match” and follow multiplayer steps above to play the game regularly. At the end, the game will update your account’s stats accordingly.

System Features: (BOLD == Brenner System Features)

1. Main Menu System:
    a. **Basic Main Menu System:** The first screen the user sees when the game starts, it's the main menu that players will use to do anything in the game. It consists of 8 main GameObject menus and a script, and it allowed users to click different buttons in order to play a “Quick Match”, view the “Login” and “Create Account” menus, view the “Tutorial” placeholder menu and the “Settings” placeholder menu. There was also an alternate menu for logged in users that had buttons for the “Account Information” and “Account Stats” menus as well, plus the option to logout. It was a custom designed U.I. that used custom images and animated buttons. BRENNER

    b. Current Main Menu System: Updated version of game’s main menu that includes background music that the Settings menu actually controls, and a “Select Deck” menu that players use to choose what colour deck they want to play with before actually playing a match of the game.

2. Authentication System:
    a. **First Login + Create Account System:** An SQL table and some PHP files hosted locally by XAMPP, that were used by a couple scripts in Unity in order for the “Login” and “Create Account" menus to query the database and actually create accounts for the player as well as log them in. It took user input from the menus in the game and stored them in a script, in order to connect to and send variables to the PHP files. Said PHP files had to query the database using those variables in order to create a new account in the table or check the variables against an already existing account in the table. BRENNER

    b. **Login + Create Account Tests and Account Stats:** Extensive tests covering the user input possibilities of the “Login” and “Create Account" menus, made using Unity’s editMode test scripts–which test the results of the PHP query to ensure that invalid user inputs return the correct error messages. Covered several cases of invalid input combinations that include SQL injection, invalid email addresses (missing “.com”, too long or too short), invalid passwords (too long or too short), already existing email addresses or passwords (for “Create Account System”), and mismatched passwords (“Create Account System” requires typing out the password twice) etc.. Also tweaked “Login” and “Create Account” scripts return the account’s stats from the table after successfully logging in, and the “Account Information” and “Stats” menus actually use the table’s values to display the user’s stats. BRENNER

    c. Password Reset and Encryption: Passwords now stored as salt and hash in the SQL table using encryption SH256, and a “reset token” was added to the table as well. Users are sent the temporary “reset token” in their email (using an email server), and they can use the token to reset their password.

    d. Current Login and Create Account System: updated system that includes better confirmation and error messages in the login and create account menus, and additionally, users are emailed a “welcome” email when they successfully create an account. Players can also delete their account using the Account Info menu once logged in.

3. Offline Base Build of Game (what's on the website):
    a. Custom Card Game Rules:
    b. 4 Custom Coloured Playing Decks:
    c. Deck Selection and Creation:
    d. Player Health and Mana:
    e. Playing Cards:
    f. Attacking with Cards:
    g.

4. Multiplayer Build of Game:
    a. **Multiplayer Development Environment with server creating game deck:** Unity Packages, GameObjects, and scripts used to start developing and implementing server and client builds of the game. Colour Break has the custom rule where the deck that both players draw from is made up of both player's decks that they chose to play with, so it needs to get the server to see what colour both players picked, and have it properly mix the game deck accordingly–avoiding clients creating their gamedeck locally and possibly cheating, as well as telling the server inaccurate information in attempts to cheat as well. Took weeks and a couple iterations to finally create the correct deck using the ParallelSync API (to run and test "host" and "client" builds using multiple editors at the same time) and the Mirror Networking API for Unity (uses specific network objects and new scripts with commands & rpcs to create and manipulate different things on "client" & "server" builds). BRENNER

    b. **Server deals cards, clients display cards properly:** The server deals each player 3 cards from the deck, and the client's player can only see the backs of their opponent's cards. The server has to spawn the card objects on the clients and assign the proper authority over the card to the right client, in order for the player to properly see and actually manipulate their cards and not their opponent's. Involved tweaking original deal and display functions to work over the server using multiple commands and rpc calls. BRENNER

    c. **Server controls turns, clients play cards, attack with cards:** The server assigns the first "client" that pressed the "deal cards" button to go first, and that player will be able to "play" cards from their hand if they can afford its "mana" cost, "attack" with a card they have "played", or end their turn. Had to ensure that a player cannot zoom in on, drag, or attack with their opponent's cards at all, that they cannot do anything while it isn't their turn, and that the server keeps track of the turns properly. The server also has to manage each player's "health" and "mana" in order for the players to actually lose mana when they play a card, as well as to communicate to both players when a player loses health from being attacked. BRENNER

    d. Game Over Scene:This consisted of determining the end of the game by constantly checking if a player is out of health after each attack. Creating the layout for the "GameOver" scene which triggers when the game ends. The server needs to communicate to both clients that the game is over, stop the server, and display the game over scene for both clients. Additionally, the "ShareVarManager" script needs to track the game stats for each client if the player is logged, such as damage dealt and win or loss. This involved

restructuring the health deduction system in “ShareVarManager” and updating a remote a script so that those variables could be sent to the Game Over scene after the server has stopped. The Game Over scene also displays whether the game was won or lost. each client must track its own player number and pass it to the Game Over scene as well as which player won the game so that each client knows their result after the server has stopped. It includes another script which communicates with a php file to update each players game stats in the SQL table if they are logged in.

a. Zoom on hover: hover over cards to see a larger version: when the user hovers their mouse cursor over a card it creates a new window which shows a zoomed version of the card so that the reader is able to read the card details clearly and make strategic choices in the game accordingly. It uses a custom zoomscript to create a window and uses variables to display each card info clearly. It took two weeks to fix the scaling and text blurring issue in the feature

b. Cards have abilities: Based on the text found on a card, certain things happen. For example, playing a certain card may draw you another card

1. Tutorial Build of Game: a new scene triggered from the main menu that is made from the base build of the game in order to skip the multiplayer requirements. It uses a custom made playing deck to walk the player through the basic mechanics of the game like playing a card, what “mana” is, how to attack and explaind the general flow of the game to the user etc. It uses a custom dialogue script for the flow of the dialogue, tutorial script to establish the flow of each tutorial sequence. The tutorial is made up of three sequences and the tutorial script establishes the flow of these sequences
   Uses custom animations with the Dotween animation package to explain each step to the player and animate all the arrows and pointers.
5.

6. Website System:
   a. Web APIs: the client side / web front end is connected to the mongoDB database and logics using API calls. API is created using Node.js and Express.js, and there are also validation checks on each API function.

   b. Web API testing: API functions are tested using Jest (javascript testing framework) and Postman to make sure that each function works correctly and return a proper HTTP return code.

   c. Web Deployment: web (front and backend) are deployed using Vercel with CI/CD pipeline. This pipeline allows for auto-deployment on each change in the repository’s ‘website’ branch.

d. Web Pages: “Home” page runs Unity game, “Rules” page to display the game’s rules in a written form instead of the playbale tutorial in the game, and a “Cards” page that displays all the cards and their information.

e. Game Deployment: Unity game built using WebGL and uploaded to simmer.io for live deployment, and that build is embedded into the website for users to interact with.

f. Web UI testing: made using Selenium IDE testing to check each input field and buttons works properly.

7. Website Admin functionality:
    a. Admins Login Page: admin account is predefined in the database (uname + pwd: admin), admin can login by changing the url path to “.../adminLogin”

    b. Rules Page: logged in admins can add, delete, or edit any rules on the rules page using the add rule button at the top of the page, or the buttons located on the bottom of each rule. The modal will be triggered if a button is pressed, and there is user input validation for the rules being added or changed (both in front and backend). The order of the existing rules on the page will be shifted up or down, depending on the user’s new input.

    c. Cards Page: logged in admins can add, delete, or edit cards in any of the decks using the add card button at the top of the page or buttons at bottom of each card. Still has user input validation in both the front and backend, and each card’s card art is dynamically shown by storing the image's path in the mongoDB database.

8. Unity testing framework (UTF): Using the unity testing framework, created test to test in game functionalities. Using the playmode testing feature created test to test the shuffle feature in the game, card in hand testing to see if cards have spawned in the player hand, test to check if the deck is being populated, test to check if the turn system is functioning like it should.