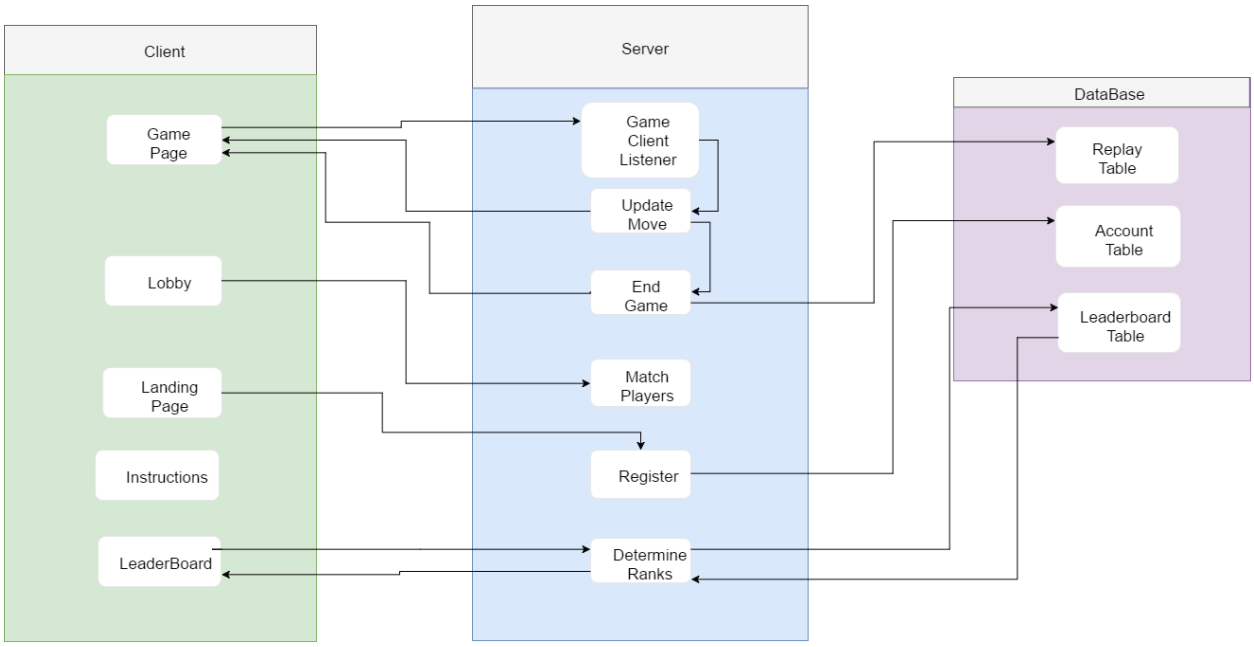
Incremental Testing and Regression Testing

1 Classification of Components

1.1 Define all components



Component A - Game Page

Inputs:

- Other User moves
- Game state

Outputs:

- User's move

Dependencies:

- Game Client Listener: sends move
- Update Move: receives other user's move
- End Game: receives game state

Component B - Game Client Listener

Inputs:

- A move command from the game page

Outputs:

- Passes the command along to UpdateMove

Dependencies:

- Game Page: Accepts a message from the Game Page
- UpdateMove: Passes the accepted message to UpdateMove

Component C - UpdateMove

Inputs:
- A move command from the game client listener
  - Ex: “MOVE POP-DECK” and “MOVE SPIT”

Outputs:
- Updates the data structures containing the current state of the game
- Calls endGame when appropriate

Dependencies:
- Game Client Listener: The game client listener passes commands from the game client into UpdateMove
- Game Page: signals the client game pages to show move.

Component D - endGame

Inputs:
- The number of valid moves remaining

Outputs:
- If no moves are left it ends the game, storing the game info and updating player ranks.
- If there are still valid moves, does nothing.

Dependencies:
- UpdateMove: After every move, checks whether the game should end or not.
- Game Page: If game ends, signals the client game pages to show game results.
- Store Game: Stores the game info for replays.
- Leaderboard Table: Updates the rank of involved players

Component E - Replay Table

Inputs:
- The game’s ID
- The game’s winner
- The game’s players
- The game’s total Moves
- The game’s moves

Outputs:
- There is no output returns to the backend server
- A json file is saved in the MongoDB based of off the game’s information

Dependencies:
- A connection to the MongoDB

Component F - Lobby

Inputs:
- The user

Outputs:
- The game for the user to join

Dependencies:
- There being more than 1 person in the lobby

Component G - Match Players

Inputs:
- The match for users to join

Outputs:
- Starting game state

Dependencies:
- Lobby: receives the party

Component H - Landing Page

Inputs:
- A user's username

Outputs:
- A user's username
- A user's locally cached key

Dependencies:
- Register: sends the username and pass; expects verification back

Component I - Register

Inputs:
- A user's username
- A user's locally cached key

Outputs:
- User's username
- User's locally cached key
- Verification

Dependencies:
- Landing Page: sends success verification
- Account Table: sends username and password

Component J - Account Table

Inputs:
- A user's username
- A user's locally cached key

Outputs:
- There is no output returns to the backend server
- A json file is saved in the MongoDB based of off the user's information

Dependencies:
- A connection to the MongoDB

Component K - Leaderboard

Inputs:

- Leaderboard table

Outputs:
- none

Dependencies:
- Determine Rank: receives the leaderboard

Component L - Determine Rank

Inputs:
- Leaderboard table

Outputs:
- Leaderboard table

Dependencies:
- Leaderboard table: receives the table

Component M - Leaderboard Table

Inputs:
- None

Outputs:
- A json file with all of users' usernames, wins, and total score sorted by the user's total score

Dependencies:
- A connection to the MongoDB

Component N - Returning Game Replay

Inputs:
- The game's ID

Outputs:
- A json document with the game's players, winners, and the moves

Dependencies:
- A connection to the MongoDB

Component O - Instructions

Inputs:
- none

Outputs:
- none

Dependencies:
- none

1.2 Which form of incremental testing did you follow

We did a bottom-up approach for our incremental testing. This made more sense since our project contains many components and modules -- front-end/client, backend/server, and database -- that all need to operate independently in order for the game to be playable/testable.

By testing small portions of each module as we built them we significantly decreased the amount of debugging time that would've needed to be done had we taken a top-down approach.

2 Incremental and Regression Testing

2.1 Automation

Our test suite for the frontend and backend is fully automated, using *mocha* as our test runner, *chai* as our assertion library, and *istanbul* as our code coverage runner. Whenever one of us pushes to github, *TravisCI* automatically runs the tests for that commit or Pull Request, and raises flags if anything fails, ensuring that we **always** have functional code on our master branch.

2.2 Defect log

Module	**Component A - Game Page**

Incremental Testing

Defect #	Description	Severity	How to Correct
1	Spectators can make moves for player 1.	1	Ignore commands issued by spectators.
2	Animating card from hand to pile does not hold correct value.	2	Created logic to time assignment of card value to animating card, hand, pile.
3	Piles from players not in game have undefined values for their images.	3	Added empty pile image to be “undefined.png”
4	Opponent cards not showing correct values.	2	Changed logic to correct the values.
5	Opponent pile and hand count is always four.	3	Hiding piles and hands not in use.
6	Game time did not persist when page refresh	2	Stored game timestamp in server

Regression Testing

Defect #	Description	Severity	How to Correct

1	No defects	N/A	N/A

Module	**Component B - Game Client Listener**

Incremental Testing

Defect #	Description	Severity	How to Correct
1	No defects	N/A	N/A

Regression Testing

Defect #	Description	Severity	How to Correct
1	No defects	N/A	N/A

Module	**Component C - UpdateMove**

Incremental Testing

Defect #	Description	Severity	How to Correct
1	Doesn't send game updates to spectator clients	1	Added spectators to list of client who receive game updates.

Regression Testing

Defect #	Description	Severity	How to Correct
1	No defects	N/A	N/A

Module	**Component D - endGame**

Incremental Testing

Defect #	Description	Severity	How to Correct
1	Could be called more than once so the game wouldn't	2	Disabled the game functions when end game is called.

	actually end and it would continue running.		
2	Only ranked players who played all their cards. (If nobody played all their cards, nobody would receive points)	1	Ranked remaining players by number of unplayed cards.

Regression Testing

Defect #	Description	Severity	How to Correct
1	No defects	N/A	N/A

Module	**Component E - Replay Table**

Incremental Testing

Defect #	Description	Severity	How to Correct
1	The game information being saved was incorrect. The default value was overwriting the value that was being set before the save	2	The mongoose schema was corrected to the right format. Before the default value was not correctly set, this was fixed.
2	End game was calling save game multiple times so we were trying to save the game multiple times causing the Database to throw an error.	2	Disable function calls when a game ends.
3	When calling the replay data function the parameter parser was incorrect.	2	Use the format router.get('/:id', req.params.id

			To read the parameters

Regression Testing

Defect #	Description	Severity	How to Correct
1	Stored Data in the wrong Database	1	Change the mongoose connection setting
2	The code threw an exception when trying to create a mongoose item based off of our game schema.	2	Mongoose has different type names based on the version of mongoose and we were using depreciated types.

Module	**Component F - Lobby**

Incremental Testing

Defect #	Description	Severity	How to Correct
1	Ready up button starts the game and redirects site to game page.	3	Design decision to change this button to start the game.

Regression Testing

Defect #	Description	Severity	How to Correct
1	No defects	N/A	N/A

Module	**Component G - Match Player**

Incremental Testing

Defect #	Description	Severity	How to Correct
1	No defects	N/A	N/A

Regression Testing

Defect #	Description	Severity	How to Correct
1	No defects	N/A	N/A

Module	**Component H - Landing Page**

Incremental Testing

Defect #	Description	Severity	How to Correct
1	Play Now, Lobby, and Leaderboard buttons do not direct to the correct location	1	Have Onclick functions direct the page to the correct location
2	Username would not cache correctly and have a empty value causing the user to not be able to play.	1	Created a logout/re-register button to take use back to register screen.

Regression Testing

Defect #	Description	Severity	How to Correct
1	No defects	N/A	N/A

Module	**Component I - Register**

Incremental Testing

Defect #	Description	Severity	How to Correct
1	No defects	N/A	N/A

Regression Testing

Defect #	Description	Severity	How to Correct
1	No defects	N/A	N/A

Module	**Component J - Account Table**

Incremental Testing

Defect #	Description	Severity	How to Correct
1	Server crashes when the same username is used twice	1	Better exception handling around the server
2	Certain fields were not being added to the user database when a user was saved	2	Adding default values to the mongoose schema
3	The user information being saved was incorrect. The default value was overwriting the value that was being set before the save	3	Change the way the user data was being saved into the MongoDB.
4	Users were not being assigned a default value for wins and losses so when trying to get	3	Set default values in the model page in the backend

	player information so wins and games played		

Regression Testing

Defect #	Description	Severity	How to Correct
1	Stored user data in the wrong Database	2	Change the mongoose connection setting
2	The connection to MongoDB did not work.	1	Change the mongoose connection method.
3	Updating users at the end of a game was assigning 1 to the value vs adding 1	2	Use the increment flag in mongoose. The format is: $inc: { gamesWon: 1} }

Module	**Component K - Leaderboard**

Incremental Testing

Defect #	Description	Severity	How to Correct
1	The leaderboard is not showing wins and losses	3	Fix the mapping function in the front end to match the Database fields
2	The leaderboard is not sorting the users	3	Add the sort flag to the mongoose query so the query result is sorted
3	The leaderboard is being sorted in ascending order versus descending order	3	Fix the sort flag to the mongoose query so it is: sort({gamesWon :

			-1})

Regression Testing

Defect #	Description	Severity	How to Correct
1	No defects	N/A	N/A

Module	**Component L - Determine Rank**

Incremental Testing

Defect #	Description	Severity	How to Correct
1	The game was determining the wrong winner	2	Fixing the endgame method fixed this issue. Endgame was overwriting the winner.

Regression Testing

Defect #	Description	Severity	How to Correct
1	No defects	N/A	N/A

Module	**Component M - Leaderboard Table**

Incremental Testing

Defect #	Description	Severity	How to Correct
1	The query wasn't sorting the users by totalScore	3	Rewriting the query to sort by totalScore.
2	Was returning all of the user data, which was wasteful and it didn't map well to the	3	Rewriting the query to only include the: 'username' , 'totalScore', 'gamesWon'

	leaderboard table		

Regression Testing

Defect #	Description	Severity	How to Correct
1	No defects	N/A	N/A

Module	**Component N - Returning Game Replay**

Incremental Testing

Defect #	Description	Severity	How to Correct
1	Was returning all of the game data, which was wasteful. It also make it harder to build the replay data	3	Rewriting the query to only include the: ‘player’ , ‘winner’, ‘state’
2	The gameReplayID in the url was not returning the correct game	2	Rewriting the way the URL was being parsed.

Regression Testing

Defect #	Description	Severity	How to Correct
1	No defects	N/A	N/A

Module	**Component G - Instructions**

Incremental Testing

Defect #	Description	Severity	How to Correct
1	No defects	N/A	N/A

Regression Testing

Defect #	Description	Severity	How to Correct
1	No defects	N/A	N/A