Card Games

By Group 6

Alex Ackerlund
Ransom Bowman
Tim Parrish
Anna Watson

Our deployed project (through GitHub pages) can be found here:
https://timparrish.github.io/card-games/

Or code repository can be found on GitHub:
https://github.com/TimParrish/card-games

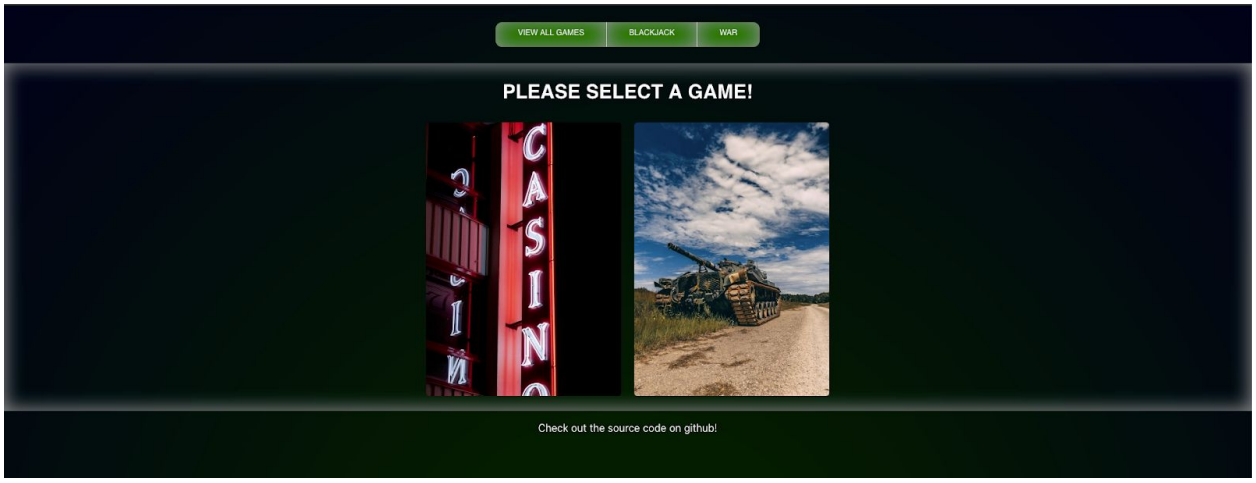
Objectives:

Our objective was to use the Deck of Cards API (https://deckofcardsapi.com) in a single page web application. This required us to learn how to properly use a web API and how to incorporate it into a web stack.

Summary:

The site is designed as a place to practice games you would commonly find at a casino. It creates a judgement free zone where you can learn to play these games without real money on the line. Ideally there will be more games in the future. Due to the nature of most casino card games, AI would be required for implementation. One really nice feature of the JavaScript framework React is its ability to control render cycles and state through “Lifecycle Functions” (see link in resources). We had a lot of fun learning about how to leverage Lifecycle Functions to do things like, auto-draw the initial cards or start a new game (making the API request) as soon as the page loads. With the internal state tracking, a user can switch from blackjack to War and back again without losing their game progress. The state of the game is tracked up until the web browser is reloaded or the player selects “new game”.

Home Screen



BlackJack

BlackJack is a common game in gambling and can be found at most casinos and gambling establishments. The game can be played by one to three players and the dealer. The game is played with one to eight decks of cards with six decks being the most common number of decks used.

The goal of this game is to get the value of the cards in your hand to equal twenty one. If the total value of your hand exceeds twenty one this is called a bust and you automatically lose the round. The Jack, Queen, and King cards all have a value of ten while the Ace can have a value of one or eleven. All other cards have the value that is printed on them. Each round starts with all players being dealt two cards. At this point the dealer also deals themselves two cards, the first one facedown, the second face up for everyone to see. Play then proceeds clockwise starting with the player that is to the right of the dealer. On a player's turn they can hit or stay

Hit: When a player chooses to hit this means they would like another card which the dealer gives to them.

Stay: When a player chooses to stay it means that they are satisfied with the cards in their hand and would not like another one.

When one elects to stay we move to the next stage of the game. At this point all players reveal their hands and the dealer turns over their facedown card. If the dealer's hand is under a value of 16 then they will draw cards for their hand until they reach sixteen. At this point it is up to the dealer if they want to continue drawing cards or not. While the hit until sixteen rules is fairly standard, not every dealer and/or casino may follow those rules. For this web application, the dealer will hit until their hand is sixteen or greater. When the dealer has stopped drawing cards for their hand all players and the dealer compare their hands to see who is closest to or equal to twenty one without exceeding twenty one. Whoever wins claims anything that was bet. At this point all cards are placed into the discard pile and the remaining unused deck is used for the next round.

War

War is a simple card game played by two people (in this case the user and “dealer”). The game is played with a single deck of cards. At the start of the game each player is given half of the deck (face down) with the goal being to get the entire deck leaving your opponent with no cards. The game is played in rounds. Each round starts with both players revealing the top card of their deck. The player who has the higher valued card (From lowest to highest cards are 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King, Ace) take both cards and places them face down on the bottom of their deck thus ending one round. If both players reveal cards with the same value then the players enter into a war. At this stage both players place three cards face down, this is known as the “pot”. Both players then reveal another card and again, the higher valued card wins, the winner takes all revealed cards and the pot and places them on the bottom of their pile. Should both players reveal cards of the same value again then the war is repeated (this process continues until one player wins the war and claim all revealed cards and the pot. Should a player run out of cards in the middle of a war then they turn their last card face up and use that as their next revealed card. That player will continue to use that card until the round ends. A player wins when the entire deck is in their pile.

Tools used:

1. NodeJS
2. ReactJS
3. React Router (for creating navigation between components)
4. Styled components (allows for the use of CSS with components)
5. Axios (for making HTTP requests to the RESTful API
6. Github pages: for a simple way to host the site.

You can find links to these tools and more in the references section of the report.

Alex:

Key contributions:

- Javascript/backend expert
- Creator of War
- Worked with Tim on API optimization, caching all the cards from the deck locally to improve speed and limit both API and internet dependency

I was responsible for creating the card game War. This required the implementation of several key pieces of functionality: obtaining a deck from the API and maintaining a list of cards each player has as well as the card(s) each player has flipped over. I was also one of the javascript experts. Tim and myself knew javascript the best among the people in the group so we handled most of the javascript logic. Due to this we were constantly in communication with each other about what we have done and any useful tools we discovered.

Anna:

Key contributions:

- Front-end
- Functionality testing

I worked on creating consistent and interesting front end styling which is intuitive and responsive. I worked on testing of functionality and trying to break things so we can fix them.

Ransom:

Key contributions:

- Front-end
- Mobile Responsiveness

I was mainly responsible for the website's mobile responsiveness capabilities. To properly scale the website to most screens, I hope to implement CSS's flex feature. Currently, the website utilizes mobile-responsive breakpoints that change based off of @media queries to check the actual screen resolution of the device. Breakpoints are being implemented as follows: 600 pixels (mobile), 800 pixels (tablet) , 1000 pixels (desktop). Those breakpoints are stored in variables so that we can modify the breakpoints in just one spot and have it reflect the whole library. The method I'm using to calculate the scaling values is: ((designated size for the

laptop)/(laptop size)) * (category size device falls into). An example would be: an object's width is 400 pixels under the desktop breakpoint. To scale it to tablet, I take 400, divide it by 1000, and then multiply the result by 800, resulting in a new size of 320 pixels. I applied this formula to all of the styling folders.

Tim:

Key contributions:

- Created the NodeJs project
- Deployed project to GitHub Pages
- Initial design of playing field
- Creator of Blackjack
- Created folder structure with import and exports to carry the file structure
- In charge of deployment of Styled-Components styling
- Added mobile responsive statements to utilities folder to make importing to components simple
- Implemented React Router for navigation
- Built header
- Made initial connection to API with the Axios library to generate deck
- Added copyright free images to the project

The first hurdle that I encountered was finding a way to dynamically render cards into a player or dealer's hand. For the 'pitch' I had hard coded some card images to render as a simple illustration but I realized that without going over 21, it is possible to have 11 cards in a hand without going over 21 if the dealer is using just one deck. Since blackjack is typically played with six decks of cards, that number may be even higher. The render part of a React component will allow you to write JSX which is very much like JavaScript with a few subtle differences. The solution to the issue of rendering an unknown number of cards was to use the JavaScript array method '.map'.

Conclusion:

While a fun idea, we were rather limited in the number of implementable card games as most games require some kind of AI. Attempts to implement these games were deemed utter failures as we watched the program flounder hilariously while playing a game of go fish. It seems like the goal of the original project, to create casino card games, had to be abandoned in favor of making it a cool card game website instead. We learned the importance of having a designer in web development. It is extremely difficult to curate consistent and engaging visuals for our site. We learned we don't have the vision for it. If we were to do this again we would design for mobile first. We would have more consistent meetings to help keep everyone up to date on the project.

References

Deck of Cards API - https://deckofcardsapi.com
Axios - https://www.npmjs.com/package/axios
React Router - https://www.npmjs.com/package/react-router
Styled Components - https://www.npmjs.com/package/styled-components
Unsplash.com - https://unsplash.com/
NodeJS - https://nodejs.org/en/
ReactJS - https://reactjs.org/
GitHub Pages - https://www.npmjs.com/package/gh-pages
React State and LifeCycles - https://reactjs.org/docs/state-and-lifecycle.html
CSS Flex - https://www.w3schools.com/css/css3_flexbox.asp