**Overview:**

Highcarder is a casual/semi-competitive card game where the main premise is to survive as many “antes” as possible while gathering the highest score possible by playing poker hands from an ever-dwindling deck of cards. The aim of this web application is to develop a simple, yet addicting gameplay loop that would keep the player engaged and aim to reach for the highest possible score and level. The game is inspired by other card games and deck builder roguelikes like Texas Hold ‘em Poker, *Uno (YEAR), Persona 5 Royal’s (20XX)* Tycoon Minigame, Slay the Spire (YEAR), and mainly the recent *Balatro (2024).*

**Usability and User Experience.**

I wanted to create a game that was simple but also had depth to it’s gameplay. To do this, I looked to games with simple gameplay loops. I settled on a deck-builder card game as one game, Balatro, inspired me as it was coded entirely in the Lua language using the Love2D framework, without using a game engine. To create a similar game, I decided to use JavaScript and the p5.js library as both were like Lua and the Love2D framework, but arguably more powerful as JavaScript featured Classes where Lua does not.

A major part of the user experience is through the gameplay, which is as follow:

* You start with a basic 52 card deck and 7 cards in your hand.
* There are 5 rounds in every ante.
* You progress each round by playing a poker hand.
* You cannot progress onto the next ante until you meet the score requirement.
* The score requirement will increase every ante.
* After every ante, you can upgrade your deck through Booster Packs (add cards), Refinements (edit your deck/cards), Perks (increase resources/meta upgrades), or Abilities (passive effects that occur every round played, either with score influencing, card adding, or meta effects).
* Exceeding the requirement by multiples will earn you multiple upgrade choices.
* You can burn an upgrade to permanently remove it from appearing again.
* You can freeze an upgrade to keep it within your choices while choosing another upgrade.
* You can skip an upgrade choice to store it for the next upgrade phase.
* Failing to meet the score requirement whilst you have stored upgrade choices decrease them by 1 for every round that you are still unable to meet the requirement.
* Every five antes, the difficulty will further increase, and a debuff will be added.
* There is no win condition, only a lose condition: Run out of cards to play.

The similarity to Poker, along with options during the upgrade phase (to either “burn” or “freeze” an upgrade) provides a satisfying user experience rather than frustrating, giving the user choice of freedom and a level of strategy. Additionally, the wide variety of upgrades allows the user to experiment and create strategies with upgrades best suited to their playstyle, making the gameplay loop addicting and engaging. There being no win condition also incentives players to keep going to see how high of a score they can achieve, creating a slight competitive aspect to the game. In addition, players will be able to view their final total scores in the Leaderboard. Hypothetically, scores will be shared across other players, and they will be able to see each other’s scores. For assessment purposes, the leaderboard scores are stored via localStorage. This gives a sense of progress run after run, to see whether they have improved, or stagnated, giving a sense of completion or self-improvement, whilst having a digital copy of their best score or runs.

Highcarder only requires one mode of input, which is mouse or touch input to interact with any of the elements on the screen: to click on cards, upgrades, or to navigate the page. This makes it widely accessible to mobile devices.

Players will also be able to learn know to play the game within the “HOW TO PLAY” tab. It explains every feature and nuance of the game, from how the upgrade system works, list of playable poker hands (excluding secret poker hands), how scoring is calculated, rarity rates, etc. Allowing this amount of information in an easily accessible location (on top of the page) means they can quickly understand how to play the game, and experiment within game synergies, or explore what they can do and how much they can “break” the game.

Due to the nature of the game being coded entirely on a canvas and in p5.js, the game itself is not accessible via keyboard (arrow keys, enter, etc). Keyboard is only used to input the name for submitting scores to the leaderboard. This is something I wished to solve, this would’ve required me to widen my scope in an already ambitious project and spend more time on which I did not have.

**Technical Implementation**

The code of Highcarder is not really dedicated to any one part of the code but rather interacts with each other and the main global variables defined within the globals.js script. Each function pulls information from the global variables and sends information back to track the current game state, what is needed, etc.

A screenshot of a computer program

AI-generated content may be incorrect.  
A portion of Highcarder’s global variables.

The Card class manages and stores relevant information about the cards (suit and ranks), including rendering the card into the UI. The cards render themselves from a sprite sheet, which is a 923px by 380px png image. To determine which portion of the sprite sheet the card should render,

Upgrades are stored in their own separate object array, often sharing the same properties, or differing slightly based on their effects. Every type of upgrade shares a “name”, “description”, and “rarity”. Boosters (known internally as PACKS) and Refinements (known internally as EDIT PERKS)

The “generateUpgradeChoices()” method is the one of the more complicated and high-level implementations in Highcarder. When called, it first pulls 4 upgrade object arrays and filters them through a set list of rules:

* It is not already owned (if an ability, or passive perk as referred to internally)
* It has not been “burned”.
* It has not been “frozen”.

A computer screen shot of a program code

AI-generated content may be incorrect.

After sorting and filtering through all upgrades that are not frozen, burned, or already possessed if it is a passive perk, generateUpgradeChoices() then picks and chooses an upgrade from the available list of upgrades it had previously defined. How it chooses an upgrade is based upon the random() method from the p5.js library, and the defined rarity weights.

Round Logic.

Card/hand evaluation is done in both evaluateHand() whereas score calculation is done in playHand(). For evaluating the cards, and checking whether they are a poker hand, evaluateHand() uses many Booleans, if statements, and checks. Selected cards are passed via the (cards) argument whilst used

* `counts` checks for how many duplicate ranks are present. It is mostly used for identifying Pairs, Two Pairs, etc.
* `isFlush` checks if every card present is

A computer screen shot of a program code

AI-generated content may be incorrect.

Upgrades -> How they are stored, applied, conditions, effects, etc

Burning and Freezing -> How that works

Card Evaluation -> How cards and hands are evaluated

Leaderboard ->

HTML display ->

**Critique**

Sound  
More focus on HTML and CSS – achievements page maybe/unlockable  
More features that had to be cut because going over scope/too ambitious.  
Maybe animations  
inclusion of more upgrades -> legendary and cursed upgrades (not included because ran out of ideas/scope got too ambitious which lead to not enough time.  
upgrades aren’t as well balanced as I’d like it to be (more focused on high ranking cards rather than other strategies) – this is a game design flaw rather than a user experience flaw, and a result of poor planning (throwing random ideas at the wall/taking too many ideas from already existing games, most notably balatro)