Documentation Guide.

Abstract: It is important to state that this game still lacks a lot of stuff. For example the weather and leader cards. It also lacks the “special powers” that certain cards have. The file paths are also hardcoded. One possible solution to this, is hosting the files in a blob storage, but that’s a future implementation. For now, I want to continue with other projects.

Cards

I’ll start off with the card recognition system. The first issue that I encountered was: How to recognise an image? How would my game recognise that this image has a certain character, with certain damage point, special ability, etc.

The solution that I gave to this problem was by comparing hashes. I did a quick bash command to retrieve all the hashes for all the images and further stored them in a json file. This JSON will contain the hash from the file, with its corresponding character name, character damage, etc.Such as shown in the image below:

```
  { "hash":
"fc89dc82f86eed3edd0ec43d4eb2230be683689741fdf48b8f1187b5f4a4b142",
"name": "Dol Blathanna Scout 2", "house": "Scoia'tael", "damage": 6,
"field": "sword and bow", "ability": "switch"},
```

With this, we can appreciate that in card.py, there is a function called getCard(). In this function we could retrieve the file from a s3 or blob storage, however I did this locally for the sake of efficiency. Here, the function calculates the hash of the retrieved file and calculates its hash, to then search and compare it with the JSON.

With all this information, we can create a card object, such as this:

```
card = HouseCard (
        cardInfo["house"],
        cardInfo["name"],
        url,
        cardInfo["damage"],
        cardInfo["field"],
        isLegend,
        cardInfo["ability"],
        isLeader)
    return card
```

Move card
This was genuinely the most difficult thing in this project. I attempted lots of things for days and I'm satisfied with my solution.

So first, I added the self.on_board property into the card class. This boolean variable will be false by default. This property will verify wether the card is on the bord or not.

```
self.on_board = False
```

It is important to note that you need to make your class to inherit from the pygame sprite class. Just like this:

```
class Card(pygame.sprite.Sprite):
    def __init__(self, name, image, damage, isLegend=False,
specialAbility=None, weatherType=None):
        pygame.sprite.Sprite.__init__(self)
```

Main

Going to my main.py is where all the spice comes up. At first I had the cards organised in a pygame group, however by simply using a list it was easier and it's way more easier if you have two lists: cards on deck, cards on board.

```
user1_cards = list(user1.deckCards)
user2_cards = list(user2.deckCards)
user1_battlefieldCards = []
user2_battlefieldCards = []
```

Display cards on deck

First, allocate coordinates where to initially spawn them:

```
y_start = 50        # Starting Y position
y_spacing = 150     # Vertical space between cards
```

So, at the beginning (just before the main loop starts), create a for loop to write the cards.

```
for index, card in enumerate(user1_cards):
    card.rect.topleft = (10, y_start + index * y_spacing)

for index, card in enumerate(user2_cards):
    card.rect.topleft = (1200, y_start + index * y_spacing)
```

This y_start + index * y_spacing is to give a certain separation between cards.

Motion logic
The logic for the motion of the cards is simple: You click a card, the card moves into an available space of the board, corresponding with its field (range, archer, sword)

For this if a card get's clicked, the on_board property of the card will become true, the card will be removed from the list of cards on deck and will be added to the list of cards in the battlefield. After this, the card will be moved to the corresponding position.

```
 elif event.type == pygame.MOUSEBUTTONDOWN:
            #Rounds system
            if game.isUser1Turn == True:
                for card in user1_cards:
                    print(f"User 1's turn: {game.isUser1Turn}, User 2's
turn: {game.isUser2Turn}")
                    if card in user1_cards and
card.rect.collidepoint(event.pos):
                        if not card.on_board:
                            card.on_board = True
                            # Delete the card from hand
                            user1_cards.remove(card)
                            # Add the card to the battlefield
                            user1_battlefieldCards.append(card)
                            # Move to battlefield position
                            if card.field == "bow":
                                card.rect.topleft = (bow_x_start +
bow_index * 110, bow_y)
                                bow_index += 1
```

It is worth noting that the indexes will always start in 0.

After this event capture loop (but still inside the main loop), we have to refresh the whole site, so that the select cards stop drawing on the deck.

```
screen.fill((0,0,0))
```

And finally, we can draw all cards in the main loop:

```
for card in user1_cards:
        screen.blit(card.image, card.rect)

    for card in user1_battlefieldCards:
        screen.blit(card.image, card.rect)

    for card in user2_cards:
```

```
        screen.blit(card.image, card.rect)

    for card in user2_battlefieldCards:
        screen.blit(card.image, card.rect)
```

User and game system
So, in easy words… The user class assigns the deck and saves all the cards and the game class, get the users.

User class methods:

```
def assignDeck(self, cards):
        # cards is a list of paths to the cards.
        for x, _ in enumerate(cards):
            card = getCard(cards[x])
            self.deckCards.append(card)

        return self.deckCards

    def assignLeaderCard(self, card):
        self.leaderCard = getCard(card)
        return self.leaderCards
```

Game class method:

```
def startGame(self, cardsUser1, cardsUser2):
        self.user1.assignDeck(cardsUser1)
        self.user2.assignDeck(cardsUser2)
```

Syanna's song and final reflections:

This is just a personal message for me, so please skip.
Syanna's theme is that kind of song that makes you travel into another era, into a time that won't come back. A song to stop, give yourself a break, reflect on what's going on in your life. I feel proud about this project, since I was able to get it work in such a difficult year for me. The witcher, for me, was a special support. A place where I could forge everything and Syanna's theme represents it.