

# Comprehensive Technical Report on the Project: Casper's Veil (A Tarot Reading Application)

By Afshin Ahmadin

---

## Abstract

This report delivers a comprehensive translation and expansion of the previously written Persian documentation for the project 'Casper's Veil.' The application is a command-line Tarot reading system developed in Python, backed by an SQLite database. This expanded English report preserves all content from the original Persian version while adding further elaboration, detailed explanations, and illustrative examples. The report covers the project's purpose, structure, detailed breakdown of its modules, functions, and classes, database schema, development methodology, design patterns, and iterative evolution. Each part is supplemented with code snippets, testing scenarios, and practical examples to ensure completeness. The document concludes with an extended reflection on the project's achievements, limitations, and future prospects.

## Introduction

Casper's Veil was envisioned as a digital alternative to traditional Tarot card readings. Historically, Tarot has been a physical practice requiring cards, spreads, and a reader to interpret. With the growth of technology and the increased demand for digital experiences, the idea of implementing a Tarot reading system in Python became both innovative and educational. The project provides a command-line interface that allows users to select spreads, draw cards, interpret meanings, and maintain a persistent history of readings. Development followed a Kanban methodology, with tasks split into feature-based units, progressing through stages of execution, testing, feedback, and deployment. This report translates all previously documented elements into English and expands them with additional details for clarity and technical depth.

## Project Files Overview

The application is composed of several Python modules and a central SQLite database. Each file contributes a unique role, and collectively they establish a modular, extensible architecture. Below is a high-level overview of the files:

- GameManager.py: Acts as the entry point of the application, handling menus, user inputs, and output rendering.
- TarotReader.py: Implements the system's core logic for executing spreads, saving readings, and retrieving histories.
- Cards.py: Defines the Card class and loads card data from the SQLite database.
- Spreads.py: Manages Tarot spreads, their symbolic positions, and interpretation logic.
- veilArchive.db: SQLite database storing Tarot cards, spreads, spread positions, and user reading histories.

## GameManager.py

GameManager is the controller of the application, facilitating the interaction between the user and the Tarot reading logic. It clears the console, renders menus, captures user input, and routes commands accordingly. The functions within GameManager can be viewed as command-handlers within a Command Pattern, mapping user intent to system functionality.

### show\_options()

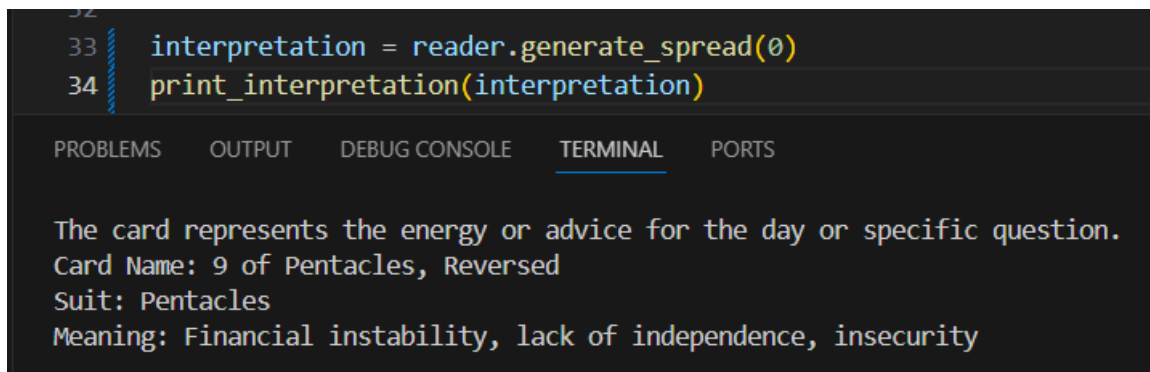
The 'show\_options' function is responsible for displaying the main menu to the user. It lists choices such as quitting, viewing reading history, or selecting a spread. After rendering the options with descriptions, the function waits for user input and returns the chosen value as an integer. This allows subsequent logic to branch accordingly.

### print\_interpretation()

This function receives an interpretation dictionary (mapping spread positions to card information) and prints it in a human-readable format. It details each position, the card name, whether the card is reversed, its suit (if applicable), and the meaning. By structuring the output, users gain a clear narrative similar to in-person Tarot sessions.

Sample Usage:

```
32
33 interpretation = reader.generate_spread(0)
34 print_interpretation(interpretation)
```



The screenshot shows a code editor with a dark theme. The code is as follows:

```
32
33 interpretation = reader.generate_spread(0)
34 print_interpretation(interpretation)
```

Below the code, the 'TERMINAL' tab is active, displaying the following output:

```
The card represents the energy or advice for the day or specific question.
Card Name: 9 of Pentacles, Reversed
Suit: Pentacles
Meaning: Financial instability, lack of independence, insecurity
```

### back\_to\_menu()

The 'back\_to\_menu' function allows the user to either return to the main menu or quit the program after a reading. It prompts the user for input and returns control flow to the application loop. This function adds a layer of interactivity, ensuring the application does not terminate abruptly.

## Cards.py

The Cards module models Tarot cards and provides methods for loading them from the database. It distinguishes between the Major Arcana and Minor Arcana, reflecting the traditional structure of Tarot. Each card includes attributes for its upright and reversed meanings, which are central to the interpretive framework of readings.

### Card Class

The Card class represents an individual Tarot card with attributes including ID, name, suit (for Minor Arcana), upright meaning, and reversed meaning. Encapsulating this data into objects enables modular handling of cards throughout the system.

## load\_cards\_from\_db()

This static method loads all cards from the SQLite database. It separates them into Major and Minor Arcana and returns two lists of Card objects. This follows a Factory Pattern, converting raw database rows into fully-fledged objects.

Sample Usage:

```
36 major, minor = Card.load_cards_from_db()
37 print(f"Loaded {len(major)} Major and {len(minor)} Minor Arcana cards")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Jonas\Desktop\Tarot> & C:/Users/Jonas/AppData/Local/Microsoft/WindowsApps/python3.13.exe
Loaded 22 Major and 56 Minor Arcana cards
```

## get\_meaning()

The 'get\_meaning' instance method retrieves the upright or reversed meaning of a card, depending on the orientation drawn during the spread. This ensures that the interpretive process accurately reflects the symbolic nuance of reversed cards.

Sample Usage:

```
36 major, minor = Card.load_cards_from_db()
37 card = major[0]
38 meaning = card.get_meaning(False)
39 print(meaning)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Jonas\Desktop\Tarot> & C:/Users/Jonas/AppData/Local/Microsoft/WindowsApps/python3.13.exe
Recklessness, foolishness, naivety, taken advantage of
```

## Spreads.py

Spreads.py manages Tarot spreads, which define how multiple cards are drawn and interpreted in relation to one another. Each spread consists of positions with symbolic labels, such as Past, Present, or Future. This file provides the Spread class and utility methods to shuffle decks, draw cards, and construct interpretations.

### Spread Class

The Spread class models a Tarot spread session, including its ID, name, positions, and drawn cards. It is instantiated whenever a user performs a reading and is discarded afterward. The class organizes the spread as a container for both structural definitions and dynamic card draws.

## load\_spreads\_from\_db()

This method loads spread definitions and their positions from the database. It returns dictionaries for each spread, including metadata, card count, and symbolic positions. By externalizing spread definitions into the database, the system is extensible and adaptable.

shuffle()

This static method randomizes the order of the deck, emulating the physical act of shuffling. It ensures fairness and unpredictability for each reading.

draw\_card()

This method selects a random card from the deck, determines whether it is reversed, and records it in the spread. The card is then removed from the deck to avoid duplicates. This process simulates the experience of drawing cards in physical Tarot.

interpret()

This method constructs the final interpretation of the spread. For each drawn card, it associates the position with the card details, orientation, and meaning. It returns a dictionary that forms the narrative of the reading.

TarotReader.py

TarotReader is the application’s logic engine. It orchestrates the flow from spreads to cards to database persistence. It can generate new readings, save them in the database, enforce history limits, and retrieve past readings for review.

generate\_spread()

This method executes a reading by building a deck, shuffling it, drawing cards, and interpreting the results. It also calls save\_reading to persist the outcome.

save\_reading()

This method inserts the reading results into the database, including the spread and each card drawn. It enforces a maximum of 20 saved readings, pruning older records to maintain the limit.

Rows: 20				
	id	spread...	created_at	
	Filter	Filter	Filter...	
1	6	2	2025-09-14 10:39:56	
2	7	1	2025-09-14 10:44:38	
3	8	5	2025-09-14 10:45:11	
4	9	1	2025-09-14 10:45:41	
5	10	1	2025-09-14 10:45:45	
6	11	1	2025-09-14 10:47:42	
7	12	1	2025-09-14 10:55:11	
8	13	1	2025-09-14 10:55:30	
9	14	1	2025-09-14 10:56:03	
10	15	1	2025-09-14 10:56:46	

get\_spreads()

This method returns the list of spreads available in the database, allowing dynamic flexibility as new spreads are added.

Rows: 54					
	id	readin...	card_id	position_i...	is_reversed
	Filter	Filter	Filter	Filter	Filter
1	15	6	19	1	0
2	16	6	33	2	0
3	17	6	41	3	1
4	18	7	57	1	1
5	19	8	4	1	0
	20	8	20	2	0
7	21	8	11	3	1
8	22	8	19	4	0
9	23	8	6	5	0
10	24	8	21	6	1
11	25	9	28	1	0
12	26	10	59	1	1
13	27	11	56	1	1
14	28	12	77	1	0
15	29	13	73	1	0
16	30	14	56	1	0
17	31	15	34	1	0
18	32	16	57	1	0
19	33	17	21	1	1
20	34	18	46	1	0

## get\_history()

This method retrieves reading history, lists past sessions with timestamps, and allows users to view details of individual readings.

## Database Schema

The veilArchive.db SQLite database contains five main tables: cards, spreads, spread\_positions, readings, and reading\_cards. The schema is normalized and supports both static Tarot definitions and dynamic user data.

id	name	suit	upright_meaning
1	The Fool	NULL	Beginnings, innocence, spontaneity, free spirit
2	The Magician	NULL	Willpower, creation, manifestation, mastery
3	The High Priestess	NULL	Intuition, unconscious knowledge, mystery, wisdom
4	The Empress	NULL	Nurturing, abundance, fertility, creativity
5	The Emperor	NULL	Authority, structure, control, stability
6	The Hierophant	NULL	Tradition, spiritual guidance, conformity, wisdom
7	The Lovers	NULL	Love, partnership, union, choices
8	The Chariot	NULL	Determination, willpower, control, victory
9	Strength	NULL	Courage, inner strength, patience, compassion
10	The Hermit	NULL	Solitude, introspection, wisdom, guidance
11	Wheel of Fortune	NULL	Luck, fate, cycles, destiny
12	Justice	NULL	Fairness, truth, balance, accountability
13	The Hanged Man	NULL	Pause, suspension, letting go, new perspective
14	Death	NULL	Endings, transformation, change, rebirth
15	Temperance	NULL	Balance, moderation, purpose, patience
16	The Devil	NULL	Temptation, addiction, materialism, bondage
17	The Tower	NULL	Destruction, revelation, chaos, sudden change
18	The Star	NULL	Hope, inspiration, renewal, guidance
19	The Moon	NULL	Illusion, intuition, subconscious, dreams
20	The Sun	NULL	Success, vitality, joy, clarity
21	Judgement	NULL	Judgement, rebirth, inner calling, transformation
22	The World	NULL	Completion, fulfillment, wholeness, achievement
23	1 of Cups	Cups	New emotional beginnings, joy, creativity

## cards

Defines all Tarot cards. Columns: id (PK), name, suit (nullable for Major Arcana), upright\_meaning, reversed\_meaning, arcana\_type.

## spreads

Defines Tarot spreads. Columns: id (PK), name, description, number\_of\_cards, deck\_used.

## spread\_positions

Defines symbolic positions in a spread. Columns: id (PK), spread\_id (FK), position\_index, position\_name.

## readings

Represents user reading sessions. Columns: id (PK), spread\_id (FK), created\_at.

## reading\_cards

Maps cards to readings. Columns: id (PK), reading\_id (FK), card\_id (FK), position\_index, is\_reversed.

Relationships:

- One spread has many spread\_positions.

- One reading belongs to one spread.
- One reading has many reading\_cards.
- One card can appear in many reading\_cards.

## Conclusion

Casper's Veil is a successful proof-of-concept that digitizes the art of Tarot reading. The project's modular structure, backed by an SQLite database and guided by the Kanban methodology, allowed incremental progress and consistent quality. Factory and Command patterns strengthened code maintainability and flexibility. The database schema balanced normalization with functionality, enabling scalable extensions. Looking forward, future iterations could include graphical interfaces, multilingual support, analytical features, and user personalization. Thus, Casper's Veil is both a functional digital Tarot system and a foundation for more ambitious software evolutions.