**Blackjack/Card Counting**

**A black and yellow card with red letters

AI-generated content may be incorrect.**

2025

Adam Wilkie

Table of Contents:

1: Title page

2: Contents

3: Blackjack explained

5: Card Counting

6: The design

8: Data Types/Structures

13: End Data Types/Structures and Flow Chart

14: Requirements/User Specifications and Design Approach

15: Algorithms

20: Debugging

21: End Debugging and Errors

23: End Errors and Variables

27: System Models

29: Testing

#: Sources

**What is Blackjack?**

Blackjack is a card game where you’re given 2 cards. Blackjack can be played with 2+ players. In Blackjack you don’t play against other players. If you, your friend and another friend played blackjack together, you’re not playing against them. Blackjack is played against the dealer.

Blackjack can also be known as 21, pontoon, etc.

* Card Values:
  + 2-10 – follow their respective values, 2 = 2, 5 = 5, 10 = 10 etc.
  + J, Q, K, – Jack, Queens and kings also = 10
  + Aces are either 1 or 11.

**What is a blackjack?**

A Blackjack is when you are dealt 2 cards with a score of 21 in total, whether it be Ace + 10, Ace + Jack, Ace + Queen, Ace + King.

If you don’t get 21 its okay, you score is still added up. e.g., say you get an 8 and a 7, you have 15, but if the dealer gets 16, the dealer wins.

In Blackjack there’s also rules such as Hitting, Standing and Busting.

**Hitting:**

Hitting is the action of asking the dealer for another card, e.g., I was delt a 4 and a 3, for a total of 7. And I hit and get another 3, and I hit again and get an Ace, I know have 21, it’s not a blackjack but it’s still 21.

**Busting:**

Busting is when you go OVER 21, If you go over 21 you lose, doesn’t matter if you have 22 and the dealer has 10, the dealer wins, same goes if the dealer busts, then you win.

**Standing:**

If you like your hand or believe you’re likely to bust if you hit again you can stand. Standing is essential just saying “I’m fine with this hand” and your turn is over. You can stand on 21, 10, 6, 15, etc.

When you’re playing, the cards are handed out as such:

* Your first card (face up)
* (if anyone else is playing, their card)
* Dealer's first card (face down)
* Your second card (face up)
* (if anyone else is playing, their second card)
* Dealer's second card (face up)

And obviously, if there are more players, it goes in order of where they sat at the table.

**Why Blackjack?**

In the Syllabus for maths for stages 4 and 5 they must learn about probability and chance. In the game of Blackjack there is probability for every flip of the card and thus it can be predicted.

In the Syllabus stages 4 and 5 must be able to:

* Recognise language of chance in everyday contexts
* Identify events that are certain and uncertain.
* Use language of chance to indicate the likelihood of familiar events
* Apply chance in everyday contexts
* Determine the likelihood of an event
* Predict the likelihood of a non-routine event
* Order events according to their likelihood
* Use chance and probability to reason in everyday contexts
* Recognise numerical values assigned to the probability of events occurring in everyday contexts
* Apply the element of chance to make decisions about familiar events
* Conduct experiments to determine the probability of an outcome
* Use knowledge of probability to make a prediction
* Apply knowledge of chance and probability to justify a prediction

The ability to know the probability of what card will show next in blackjack is one only some have. Knowing how to do such a feat will make learning more things regarding probability easier. Teaching younger generations this skill through a fun, interactive game of Blackjack, teaching the basis of card counting, will greatly improve these students skills

**How to card count:**

Card counting is a very simple endeavour, anybody of any skill can do it, it’s merely keeping track of some numbers.

The values:

2-6 = -1

7-9 = 0

10, J, Q, K, Q = +1

What does this mean??

Let’s say 6 rounds are played, if the total value is a negative number, You are more likely to get a lower number between 2-6 then something above 10, if it’s in the positives you’re more likely to get above 10 then below.

e.g., you get 3, 5, dealer gets a 2, 6, that’s 1 + 1 + 1 + 1 = 4.

The next hand you are MORE LIKELY to get a card below 10, it isn’t guaranteed but it’s more likely

This Method of card counting is called the Hi-Lo system.

**Design:**

The original idea for the start of this project is to explain blackjack and how to card count. Like this:

A screenshot of a computer

AI-generated content may be incorrect.

it’s supposed to be easy to swap between informative slides, and sort of look like a blackjack table.

For the main game at the end, where the users are meant to incorporate their skills, they should be able to keep a tally and predict what’s more likely to show.

A screenshot of a game

AI-generated content may be incorrect.

In this image here you can see there’s a total counter at the top, keeping track for you, It shows you the cards you got and the cards the dealer got, it’s got their score for you on the side.

A screenshot of a game

AI-generated content may be incorrect.

A Prompt of sorts like this will come up on the users screen. Asking what is more likely, following the score, A, Below 10, B, likely to be either, or C, Above 10.

The user will answer the question if they are correct, they will hear a ding and will get onto the next round of blackjack.

The red felt too much so the original concept was changed to a green, to look more realistic

A screenshot of a game

AI-generated content may be incorrect.

Data Types:

‘Main.py’

|  |  |  |  |
| --- | --- | --- | --- |
| Type: | Name | Description | example |
| Integer | button | Tracks which intro screen is active | 0, 1, 2, 3, 4 |
| Integer | font\_size | Dynamically calculated font size | 20, 24, 36 |
| Integer | screen\_width | Width of the screen | |  | | --- | |  |  |  | | --- | | 800 | |
| Integer | screen\_height | Height of the screen | 600 |
| Integer | image\_x, image\_y | Positioning images on screen | 100, 200, etc. |
| Integer | spacing | Vertical spacing between lines of text | 20, 30 |
| Integer | header\_x, header\_y, text\_x, text\_y, y\_start | Text/image positions | 150, 320, etc |
| Boolean | running | Controls whether the game loop continues | True, False |
| String | |  | | --- | | "Black Jack" | | |  | | --- | | Window title | | |  | | --- | | "Black Jack" | |
| String | |  | | --- | | header\_text |  |  | | --- | |  |  |  | | --- | |  | | Header for each intro screen | "Welcome to learning to Card Count..." |
| String | |  | | --- | | line |  |  | | --- | |  |  |  | | --- | |  | | Each line of body text | "This interactive game is designed to..." |
| String | |  | | --- | | "Next", "Back" | | Button text | "Next", "Back" |
| String | |  | | --- | | Font name | | Used in get\_scaled\_font() | "Times New Roman" |
| String | |  | | --- | | Image path | | Filepaths to images | "Task 1/images/bjlogo.png" |

Data Types

‘Game.py’

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Name | Description | Example |
| |  | | --- | |  |  |  | | --- | |  |  |  | | --- | |  |   Integer | WIDTH, HEIGHT | The width and height of the game window. | 800, 600 |
| Integer | card\_width, card\_height | The width and height of the cards. | card\_width = 115, card\_height = 155 |
| Integer | total(card\_list) | Function that updates the global count based on the cards in card\_list. | total(player\_hand) |
| Integer | hand\_value(hand) | Function that calculates and returns the value of a hand. | hand\_value(player\_hand) |
| **Boolean**   |  | | --- | |  | | **bust** | **Boolean indicating whether the player has busted (over 21).** | **bust = False** |
| **Boolean** | **show\_dealer** | **Boolean indicating whether to show the dealer's second card.** | **show\_dealer = True** |
| **Boolean** | **dealer\_standing** | **Boolean indicating whether the dealer has finished their turn.** | **dealer\_standing = False** |
| String | count | String representing the current count (e.g., "Total: 0"). | count = "Total: 0" |
| String | result | String holding the result of the round ("You Win!", "Dealer Busts!", etc.). | result = "You Win!" |
| String | quiz\_buttons | Dictionary holding Pygame Rect objects for the quiz answer buttons. | quiz\_buttons = {"less": Rect, "equal": Rect, "more": Rect} |
| String | font, font\_small | Pygame font objects used to render text on the screen. | font = pygame.font.SysFont('Arial', 36) |
| String | text\_surf, count\_surface, result\_surface, header, question | Pygame surfaces holding rendered text for display. | text\_surf = font.render('text', True, (255, 255, 255)) |
| String | state | A string representing the current state of the game ("game" or "quiz"). | state = "game" |
| Float | Round\_total | The current round value for card counting | round\_total = 0.0 |

Data Structures

‘Main.py’

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Name | Description | Example |
| tuple | |  | | --- | | screen size | | Resolution tuple | (800, 600) |
| tuple | |  | | --- | | color |  |  | | --- | |  | | RGB colors for screen or text | |  | | --- | | (0, 128, 0), (255, 255, 255) | |
| tuple | |  | | --- | | position | | Used for blit() or Rect() | (image\_x, image\_y), (text\_x, text\_y) |
| tuple | button size & pos | Used in pygame.Rect() | (x, y, width, height) |
| |  | | --- | | list | | body\_text\_lines | A list of strings shown as tutorial content on each page. | ["This interactive game is designed to teach you", "the fundamentals of card counting..."] |

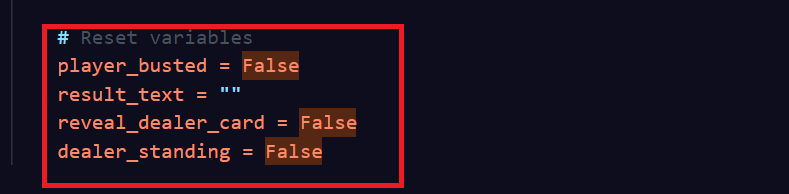
Data Structures

‘Game.py’

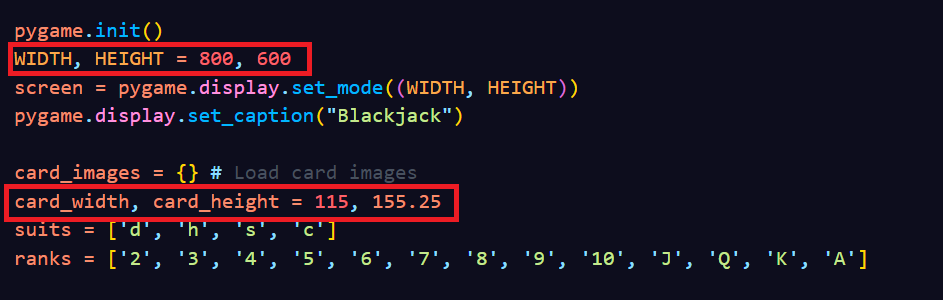
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Type | | Name | Description | | | Example |
| |  | | --- | | list |  |  | | --- | |  | | ranks | | | All possible card ranks | ['2', '3', ..., 'A'] | |
| list | suits | | | All four card suits | ['d', 'h', 's', 'c'] | |
| list | deck | | | Shuffled list of 52 card strings | ['5d', 'Qs', ..., 'Ah'] | |
| list | player\_hand | | | Cards currently held by the player | ['10c', '7d'] | |
| list | dealer\_hand | | | Cards currently held by the dealer | ['5h', 'Qd'] | |
| Dictionaries | card\_images | | | Maps card codes to their Pygame image surfaces | { '5h': <Surface>, 'Qs': <Surface>, ... } | |
| Dictionaries | buttons | | | Maps quiz answer labels to button rectangles | { 'less': <Rect>, 'equal': <Rect>, ... } | |
| Dictionaries3 | quiz\_buttons | | | Stores quiz button areas for click detection during the quiz screen | Same structure as buttons above | |

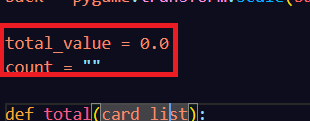
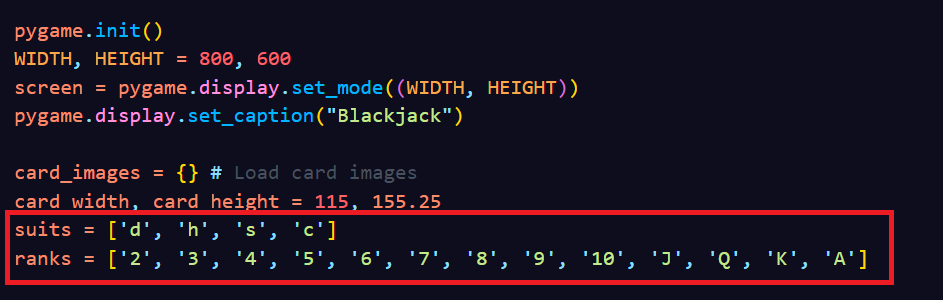
**Data Types**

Booleans:



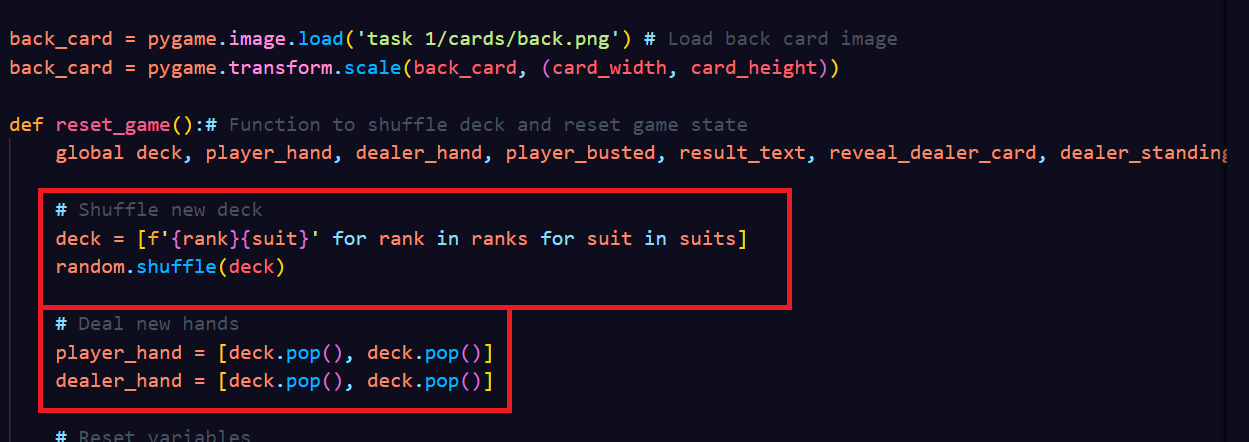
Integer:

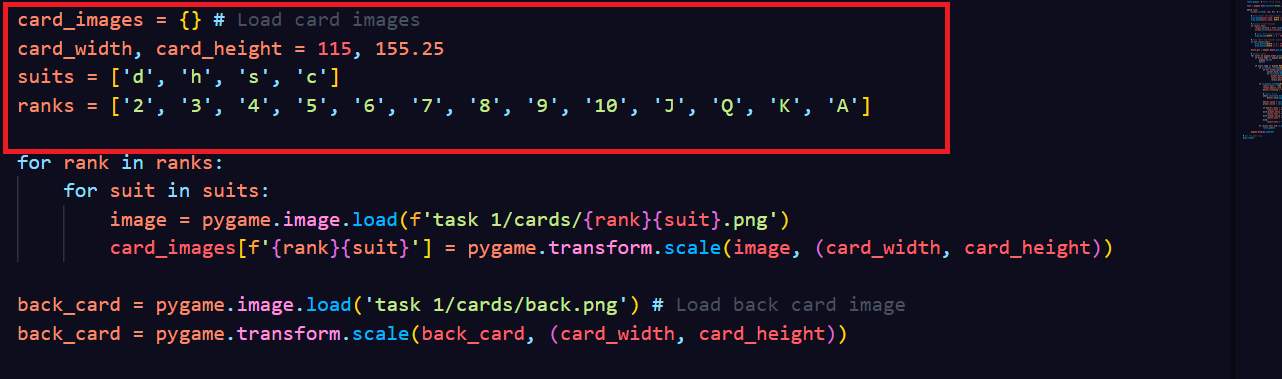
Strings:

****Float:

Data Structures:

Lists:

Dictionaries:



flow chart:

A diagram of a card game

AI-generated content may be incorrect.

**Requirements:**

The software will be a Pygame-based Blackjack game that uses card counting and a quiz to teach probability to stage 4 and 5 students. It will recreate blackjack rounds, calculate the running total using the Hi-Lo method, and give users with quiz questions to test and improve their memory and ability.

**User Specifications:**

**Target Users:**

Students in stage 4/5

**Goals:**

* Learn probability through a card game
* Understand concepts like odds, likelihood of drawing certain cards, dealer bust probability, etc.
* See how probability affects decision-making in games.

**Expectations:**

* Clear instructions and tutorials.
* Engaging and not overly difficult gameplay.
* Quiz questions that are easier to answer but require an understanding

**Design Approach:**

The Design Approach that was used within my project was Waterfall.

I felt a waterfall approach was best as I would have a structured layout of what my program will look like and how it will teach card counting.

**Algorithms**

**Card Value Calculation Algorithm:**

This Algorithm calculates the value of a hand while also giving aces the values of 11 or 1 depending on the scenario.

**Logic:**

* Sum the value of each card.
* Track how many Aces are counted as 11.
* If the total exceeds 21, convert Aces from 11 to 1 until under 21

**Type:** Conditional logic with a while loop

A computer screen shot of a code

AI-generated content may be incorrect.

**Card Counting Algorithm (total):**

Counts the Total, separating the face value of the cards into a value of either -1, 0 or 1 using the Hi-Lo car counting method.

**Logic:**

* Low cards (2–6) = -1
* High cards (10, J, Q, K, Aces) = +1
* Middle cards (7–9) = 0

**Type:** basic method that follows simple rules to count the cards.

A computer code with colorful text

AI-generated content may be incorrect.

**Game State Algorithm:**

Controls the mode of the software is in. whether its in quiz or in game.

**Logic:**

* Switching between "game" and "quiz" states.
* Detecting busts and determining win/loss/draw.
* Resetting the game state after a round.

**Type:** A state setter sets from either Game or Quiz.

A computer screen shot of a program code

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.  
A screen shot of a computer code

AI-generated content may be incorrect.

**Dealer Logic Algorithm:**

Decides on what the dealer does after the user stands.

**Logic:**

* Dealer hits until the hand value is at least 17.
* Then the dealer stops drawing.

**Type:** Simple Decision-making loop

A computer screen shot of text

AI-generated content may be incorrect.

**Quiz Answer checking:**

According to the total, depending on what the user chooses, will mark as correct.

**Logic:**

* If > 0 = high cards are more likely = answer should be "≥10".
* If count < 0 = more high cards have been played =low cards are more likely = answer should be "≤6".
* If count == 0 = Even Probability for next card = answer is "any card".

**Type:** Rule Based logic

A computer screen shot of text

AI-generated content may be incorrect.

**Card Creation and Shuffling:**

Creates a deck of card then shuffles it

**Logic:**

* Builds all combinations of rank + suit.
* Uses random.shuffle() to randomize order.

**Type:** Permutation

A screen shot of a computer code

AI-generated content may be incorrect.

**Debugging:**

**Methods of Debugging used:**

* Watches
* Interfaces Between Functions
* Debugging Output Statements

**Watches:**

Watches are something you can use to keep track of a certain variable and now what is happening with it.

Watches were used when I began to try implementing a total function. This was used because my total was spitting out random numbers that didn’t coordinate with the cards shown. And so, thing was put in to track what cards came out and make the total coordinate with it.

A black background with text and symbols

AI-generated content may be incorrect.

A computer screen with white text

AI-generated content may be incorrect.

(Example of trying to make sure card values added up before doing total)

**Interfaces Between Functions:**

Interface between functions is a interface of how data is shared between functions

Interfaces between functions were used within my code when I had the total working so I could make sure it was correct.



A blue background with white text

AI-generated content may be incorrect.

(Example of it telling the total working)

**Debugging Output Statements:**

A Debugging output statement is like a print line which tells you what’s happening within the code, so you know what’s actually going on

Debugging Output Statements were used to make sure the cards that are being printed in terminal are same in game, ensuring that its not just grabbing random cards so it can properly be adjusted towards the total.

A close up of a black background

AI-generated content may be incorrect.

**Errors Experienced**

What type of Errors are there?

* + Syntax
  + logic
  + runtime.

Withing my Code I faced all 3 errors.

**Syntax:**

During Coding this piece of educational software, I ran into many Syntax Errors. A Syntax Error is when you make a mistake within the code.

e.g., Print(“Hello”

This is a Syntax Error as it should be

Print(“Hello”)

And thus, won’t run.

This happened many times, where I may have forgotten a comma, typed a variable wrong, or just forgot things like ‘)’.



This is an example of one of the many times I made an Error. The reason this is an error is because I leave the Square Bracker ([) at the start of dealer\_hand left open. This was a simple and easy mistake to make as they were used again shortly after, [0], and due to that was accidently missed.

**Logic:**

A Logic error is when the code runs without crashing, but it doesn't do what you intended. And it is caused by a mistake in your thinking or plan, e.g., the code is "correct" in form but not in function. And it happens when the program runs fine, but the result is wrong.

**Example from my code:**

A screenshot of a computer screen

AI-generated content may be incorrect.

The reason this is a logic error is because I have flipped the signs by mistake.

“When the players hand value is greater than the dealers hand value. Player lost” is essentially what the first line is saying, where the Player won. And due to that mistake being made, it was followed through, and the same mistake arose with when dealers hand is higher than players.

**Runtime:**

A runtime error is when the code runs but then when something has happened, e.g., using an undefined variable, the software crashes. This had occurred numerous times within my code, for example, when I added my quiz. I forgot to define some of the functions and so when I ran the code, the game started and worked but then when I got to quiz It crashed.

**Variables:**

**Variables Withing ‘Main.py’**

* Pygame: The imported Pygame module used to create the game.
* Screen: The main game window (800x600) that is resizable.
* Clock: Controls the frame rate of the game.
* Button: Keeps track of the current tutorial page (0–4).
* Img: con image loaded from file for the window icon.
* image1: First tutorial image (intro page).
* image2: Second tutorial image (Blackjack rules).
* image3: Third and fourth tutorial image (card counting).
* Image4: Demo
* Image5: Demo2

**Function: get\_scaled\_font(size, font\_name)**

* size: The desired size of the font.
* font\_name: The font family to be used (default: Times New Roman).

**Function: get\_buttons(screen\_width, screen\_height)**

* screen\_width: Current width of the game window.
* screen\_height: Current height of the game window.
* button\_width: Width of the Next/Back buttons (1/6 of screen width).
* button\_height: Height of the buttons (1/12 of screen height).
* next\_btn: A pygame.Rect object for the "Next" button.
* back\_btn: A pygame.Rect object for the "Back" button.

**Function: drawIntro(screen, button, screen\_width, screen\_height)**

* Screen: The Pygame window to draw onto.
* button Current tutorial page being displayed.
* screen\_width Current width of the window.
* screen\_height: Current height of the window.
* font\_size: Font size calculated based on screen width.
* fontIntro: Scaled font object for rendering text.
* scaled\_image1: Rescaled version of image1 for display.
* scaled\_image2: Rescaled version of image2 for display.
* scaled\_image3: Rescaled version of image3 for display.
* scaled\_image4: Rescaled version of image4 for display.
* scaled\_image5: Rescaled version of image5 for display.
* image\_x: X-coordinate to center images.
* image\_y: Y-coordinate to place images vertically.
* header\_text : Title text for each tutorial page.
* body\_text\_lines; List of strings forming the body text paragraphs.
* Header: Rendered Pygame surface of the header text.
* header\_x: X-position to center the header.
* header\_y: Y-position to place the header.
* y\_start: Starting Y-position for drawing body lines.
* Spacing: Spacing between each line of body text.
* Line: Current string in the body text being rendered.
* I: Index of the line in the body text.
* Text: Rendered surface of a line of body text.
* text\_x : X-position to center a line of body text.
* text\_y: Y-position for a line of body text.
* next\_btn\_rect: Button rectangle for "Next" navigation.
* back\_btn\_rect: Button rectangle for "Back" navigation.
* next\_text: Rendered label for the "Next" button.
* back\_text: Rendered label for the "Back" button.

**Main Loop Variables**

* running: Boolean controlling whether the game loop is active.
* screen\_width: Width of the screen, updated dynamically.
* screen\_height: Height of the screen, updated dynamically.
* Event: Each event from the Pygame event queue (e.g. mouse click, quit).

**Variables Withing ‘Game.py’**

* WIDTH, HEIGHT: Window dimensions.
* screen: Pygame display surface.
* card\_images: Dictionary holding loaded and scaled card images.
* card\_width, card\_height: Dimensions of card images.
* suits, ranks: Lists of card suits and ranks.
* back: The back image of a card.
* total\_value: Float used for card counting.
* count: String showing the total card count.
* deck: List of shuffled card strings.
* player\_hand, dealer\_hand: Lists of card strings for hands.
* bust: Boolean for player bust.
* result: Game outcome string.
* show\_dealer: Boolean controlling if dealer’s full hand is shown.
* dealer\_standing: Boolean indicating if dealer is standing.

**total(card\_list)**

* card: Current card in the loop.
* rank: Extracted rank from card string.

**hand\_value(hand)**

* value: Total value of hand.
* aces: Number of aces in hand.
* card, rank: As above, used during evaluation.

**reset()**

* deck, player\_hand, dealer\_hand, bust, result, show\_dealer, dealer\_standing, count: (reinitialized here)

**hand\_draw(hand, start\_x, start\_y, ...)**

* total\_width: Total width of hand display.
* i, card: Index and current card in loop.
* x: X-coordinate of card being drawn.

**button(x, y, width, height, text)**

* font: Font object.
* text\_surf: Rendered text surface.

**clicked(x, y, width, height, mouse\_pos)**

* No local vars beyond parameters.

**draw\_quiz\_screen()**

* box\_width, box\_height, box\_x, box\_y: Dimensions and position of quiz box.
* font, header, question: Font and rendered text surfaces.
* button\_w, button\_h, spacing, start\_x, start\_y: For button layout.
* buttons: Dict of quiz button rects.
* font\_small: Font for button labels.

**game()**

* font: Game font.
* state: Game state ("game" or "quiz").
* quiz\_buttons: Button rects for quiz.
* mouse\_pos: Current mouse position.
* event: Pygame event object.
* new\_card: Card drawn during gameplay.
* dealer\_value, player\_value: Final values to determine result.

**System Model:**

**A screenshot of a computer

AI-generated content may be incorrect.**

A screenshot of a game

AI-generated content may be incorrect.

**A screenshot of a computer

AI-generated content may be incorrect.**

**Testing**

During the programming of my interactive piece of media I had numerous people test and try out my game.

Examples of things that was suggested to fix:

Card size. Make them bigger

Make Text Bigger

Move buttons to the top sides

Increase height and size on the “Total”

Add a textbox in Main so it’s easier to read

Change some Images

Start with one of the dealers cards as a back of a card

Have “wrong answer” come on screen during quiz

There was many more but that’s to name a few.

By gathering this feedback, I can enhance the look and feel of my software, improving both the UI and UX. This makes the experience more engaging and memorable, which helps the key educational concepts I’m aiming to teach stick with the user.Sources:

ChatGPT

<https://www.tutorialspoint.com/pygame/pygame_use_text_buttons.htm>,

<https://stackoverflow.com/questions/53685708/how-to-single-click-and-turn-to-another-page-in-pygame>,

<https://pygame.readthedocs.io/en/latest/4_text/text.html>,

<https://www.youtube.com/watch?v=ndtFoWWBAoE>,

<https://www.youtube.com/watch?v=ndtFoWWBAoE>,

<https://www.youtube.com/watch?v=W6Xl9Fhgsw0>,

<https://www.geeksforgeeks.org/python-display-images-with-pygame/>,

<https://www.youtube.com/watch?v=rHEnZfq_zEQ>