Team A

Stephen Connair
Chip Gaddis
Jaylyn Pham
Diego Rivera Cartegena
Leah Ryan

Fundamentals of Engineering 1181.02
Dr. Kecskemety
12/03/2024

Table of Contents

Table of Contents 2
Executive Summary 3
Project Management Documentation 4
1. Team Working Agreement 4
2. Individual Responsibility Agreement 6
Programming Responsibilities/ Roles 6
3. Project Schedule 7
4. Meeting Notes 8
a. Meeting 1 8
b. Meeting 2 8
c. Meeting 3 8
Business Plan 9
1. User Identification and Interviews 9
a. User Interview 1 9
b. User Interview 2 9
2. Electronic Advertisement 10
3. Pitch Video Link 11
Software Documentation 12
1. Introduction 12
2. User Manual 12
3. Program Description for Developers 14
4. Final Algorithm or Flowchart or Pseudocode 19
5. Final Program with Comments 23
6. Brief Discussion 43
7. Conclusion and Recommendation 45
References 47

Executive Summary

Group A – Stephen Connair, Chip Gaddis, Jaylyn Pham, Diego Rivera Cartagena, Leah Ryan

Software Design Project

Instructor – Kecskemety, GTA – Kling

12/01/2024

Background & Purpose:

The purpose the Software Design Project was to put students in a situation in which they would employ problem solving, collaboration, and creative skills. The method chosen to accomplish this was by creating one or more video games, a tangible, enjoyable way for students to observe their coding expertise grow and share their work with their peers. Team A created and learned from four games: Over Under 7, Blackjack, Matching, and Connect 4.

The scientific community will benefit from the outcome of this assignment because it has been a major steppingstone in the career paths of the students. It has been an opportunity for the students to develop not only a game, but also a passion for engineering. This will carry over to their next years of studies and will ultimately lead to creating driven engineers to add to the scientific workforce.

Results and Analysis:

As a result of the Software Design Project, the team grew more familiar with MATLAB's functionality. In particular, they learned how to play music with the (audioplayer) function, display interactive buttons with the (uicontrol) function, pause execution of the game with (uiwait), control figures with (title, xlabel, ylabel, and close), and more. Using SimpleGameEngine provided experience with exterior functions and how they can be implemented in coding. The team also grew a more intimate understanding of loops, if-else-if structures, and debugging. The team succeeded in creating a game collection with four functioning games that were both fun and educational for the user. While there are still some potential holes in the programs, mostly regarding invalid inputs causing program malfunctions, these could easily be fixed given more time.

Conclusion & Recommendations:

In conclusion, the Software Design Project drastically increased team's proficiency in MATLAB coding. Team A now has the skills to tackle coding other games in MATLAB. Given more time, some improvements could be made. Primarily, the team would iron out bugs related to inputs, they would perform a graphics overhaul on those games currently without custom graphics, and some game structures would be changed to create a more streamlined and interactive experience for the user. The rest of this document will cover in detail the process and components behind each of the four games the team created.

Project Management Documentation

1. Team Working Agreement

Team Working Agreement
Term (**Autumn 2024**)
Creation 08/27/2024; Revised 12/03/2024

1) Group Identification

Lab section # (see Carmen) - 6463
Table Letter - A
Instructor (GTA) – Adam King
Team Name (Optional) - Sphinxes

Team member info:

NAME:	**EMAIL:**	**PHONE:**	**OTHER:**
Diego Rivera Cartagena	Riveracartagena.1@buckeyemail.osu.edu	787-378-3846	
Chip Gaddis	Gaddis.60@osu.edu	330-285-4570	
Stephen Connair	Connair.83@osu.edu	937-814-6235	
Leah Ryan	ryan.1978@osu.edu	740-447-3389	
Jaylyn Pham	Pham.642@osu.edu	740-442-8300	

2) Primary Means of Communication and Expectations

State your team’s agreed upon means of communication and expectations for response. The agreed upon means of communication is via a group message in iMessage, and the expectation for response is within 48 hours or a reasonable amount of time to be able to complete the assignment.

3) Scheduling of Meetings (Schedule at least one meeting as part of constructing your team agreement.)

Day and time of regular meeting: Sundays at 4:00 pm
Location/format of meeting: Drackett Tower Common area
Agreed upon means of scheduling other meetings: Meetings will be schedules over text or email.

Meeting Schedule:

Date: Sundays	Time: 4pm-5pm	Location: Drackett Tower Common Room

Participating members (If not all.):
Diego Rivera Cartagena

Stephen Connair
Jaylyn Pham
Leah Ryan
Chip Gaddis
Agenda: (items in bulleted/numbered list)

4) General Responsibilities for All Team Members
This element of the team working agreement is the list of rules/agreements or the contract that all members agree to abide by.
Diego – Writer
Stephen – Organizer
Chip – Dungeon Master (does a little of everything)
Leah – Logic person
Jaylyn- Idea maker
These are the roles that we are at first agreeing to, but they may be subject to change. Each member of the team is expected to perform the duties associated with their role.

5) Specific Team Member Roles

- **Stephen:** Team leader
- **Jaylyn:** Video production coordinator
- **Leah:** Assistant to video production coordinator
- **Chip:** Digital media coordinator
- **Diego:** Lead programmer

6) Conflict Resolution
Each team should have a pre-agreed approach to addressing issues that may arise.
Example statement: When there are problems within our group pertaining to the general responsibilities or specific responsibilities, the following steps will be taken in this order until a resolution is found.

1. Enter conflict resolution assuming good intentions from teammates.
2. Discuss problems within the group to come to a working solution, trying to make sure all members have opportunity to participate in the discussion.
3. Hold a team discussion of the problem with a GTA or Professor to find a solution.
4. If 2 and 3 fail, ask a GTA or Professor to assist (arbitrate).

When issues arise within our group the following process will be followed:

1. Communicate the issue by either the group chat or in person in a polite manner
2. Talk the issues out to reach a solution which everyone in the group can agree on
3. If we talk it out, see if it’s okay with everyone to agree to disagree.
4. If not, look for help with the instructor.

7) Expectations of Faculty and GTA’s
Suggested Statement:
If a team member fails to live up to this agreement, the situation may be reported to the staff, but the team will still be responsible for submitting a completed assignment. Staff will be available to meet with teams to resolve issues.

If a situation arises where a team member negatively impacts the ability to complete the assignment, then the Sphinxes will hand in what they can but will meet with staff to discuss the next steps.

8) Team Signatures
Signatures:
Diego Rivera Cartagena
Stephen Connair
Leah Ryan
Chip Gaddis
Jaylyn Pham

2. Individual Responsibility Agreement

Programming Responsibilities/ Roles

Team Member	Initial Assigned Tasks	End Result
Stephen Connair	Over-Under-7	Over-Under-7
Diego Rivera Cartagena	Blackjack	Blackjack
Leah Ryan	Memory graphics	Simple Card Game (50%)
Jaylyn Pham	Memory code	Simple Card Game (50%)
Chip Gaddis	Connect 4 (smart AI)	Connect 4 (smart AI)

Table 1. Programming Responsibilities/ Roles

Documentation Responsibilities/ Roles

Team Member	Initial Assigned Tasks	End Result
Stephen Connair	Welcome Page Introduction Discussion Conclusions and Recommendations	Some roles adjusted, but all tasks %100 completed
Diego Rivera Cartagena	User Manual Final Program with Comments	Some roles adjusted, but all tasks %100 completed
Leah Ryan	Draw.io Project Management (all) Final Algorithm, Flowchart, or Pseudocode Program Description for Developers	Some roles adjusted, but all tasks %100 completed
Jaylyn Pham	Pitch Video with Documentation Final Program with Comments (Simple Card Game)	Some roles adjusted, but all tasks %100 completed

Chip Gaddis	User identification and Interview Electronic/Print Advertisement	Some roles adjusted, but all tasks %100 completed

Table 2. Documentation Responsibilities/ Roles

1. Project Schedule

Table 3. Project Schedule

Task	Start Date	Finish Date	Due Date	Primary Person	Secondary Person	Est. Hours	% Completed
Attend Meetings Sunday at 4:00 p.m. as needed	**11/06/24**	**11/24/24**	**11/28/24**	**All**	**All**	**1**	**100**
Blackjack code	**11/06/24**	**11/20/24**	**11/21/24**	**Diego Rivera Cartagena**	**All**	**5**	**100**
Over-Under-7 Code	**11/06/24**	**11/20/24**	**11/21/24**	**Stephen Connair**	**All**	**3**	**100**
Memory Code	**11/06/24**	**11/20/24**	**11/21/24**	**Jaylyn Pham and Leah Ryan**	**All**	**3**	**100**
Interviews 1	**11/14/24**	**11/14/24**	**12/04/24**	**Chip**	**All**	**.25**	**100**
Powerpoint Presentation Draft	**11/09/24**	**11/12/24**	**11/14/24**	**Jaylyn Pham**	**Leah Ryan**	**2.5**	**100**
Powerpoint Presentation	**11/12/24**	**11/17/24**	**11/28/24**	**Jaylyn Pham**	**Leah Ryan**	**2.5**	**100**
Interview 2	**11/21/2024**	**11/30/24**	**12/04/24**	**Chip Gaddis**	**All**	**.25**	**100**
Electronic/Print advertisement for the game – (flyer, brochure etc.)	**11/21/2024**	**11/30/24**	**12/04/24**	**Chip Gaddis**	**All**	**1**	
Notebook Compilation and Executive Summary	**12/1/2024**	**12/03/2024**	**12/04/24**	**Stephen Connair**	**All**	**5**	

2. Meeting Notes

a. Meeting 1

Date/Time: 11/06/24, 8:30pm
Members Present: All
Topics/ Agenda: SDP Notebook Check-in and Graphics preview
Action Items with names assigned:

- **Chip:** Program Connect 4 graphics
- **Stephen:** Program Over Under 7 graphics
- **Jaylyn:** Start PowerPoint Presentation
- **Leah:** Program Matching graphics
- **Diego:** Program Blackjack graphics

b. Meeting 2

11/21/24, 8:00am
Members Present: All
Topics/Agenda: discuss the completed program with game select screen
Action Items with names assigned:

- **Chip:** Demonstrate Connect 4 and learn how to access it
- **Stephen:** Demonstrate Over Under 7 and teach teammates how to utilize the program
- **Jaylyn:** Demonstrate Matching and learn how to access it
- **Leah:** Demonstrate Matching and learn how to access it
- **Diego:** Demonstrate Matching and learn how to access it

c. Meeting 3

11/26/24, 10:05am
Members Present: All
Topics/Agenda: Video advertisement and electronic advertisement
Action Items with names assigned:

- **Chip:** Create Connect 4 advertisement audio and electronic advert
- **Stephen:** Create intro/outro and Over Under 7 advertisement audio
- **Jaylyn:** Create partial Matching audio and put together advert
- **Leah:** Complete Matching audio
- **Diego:** Create Blackjack advertisement audio

Business Plan

1. User Identification and Interviews

Intended Audience: People of all ages and backgrounds! Sphinx Game Suite has fun for everyone. Enjoy the thrill of betting in Over Under Seven and Blackjack, challenge your mind with matching, or try competing with a friend in Connect 4!

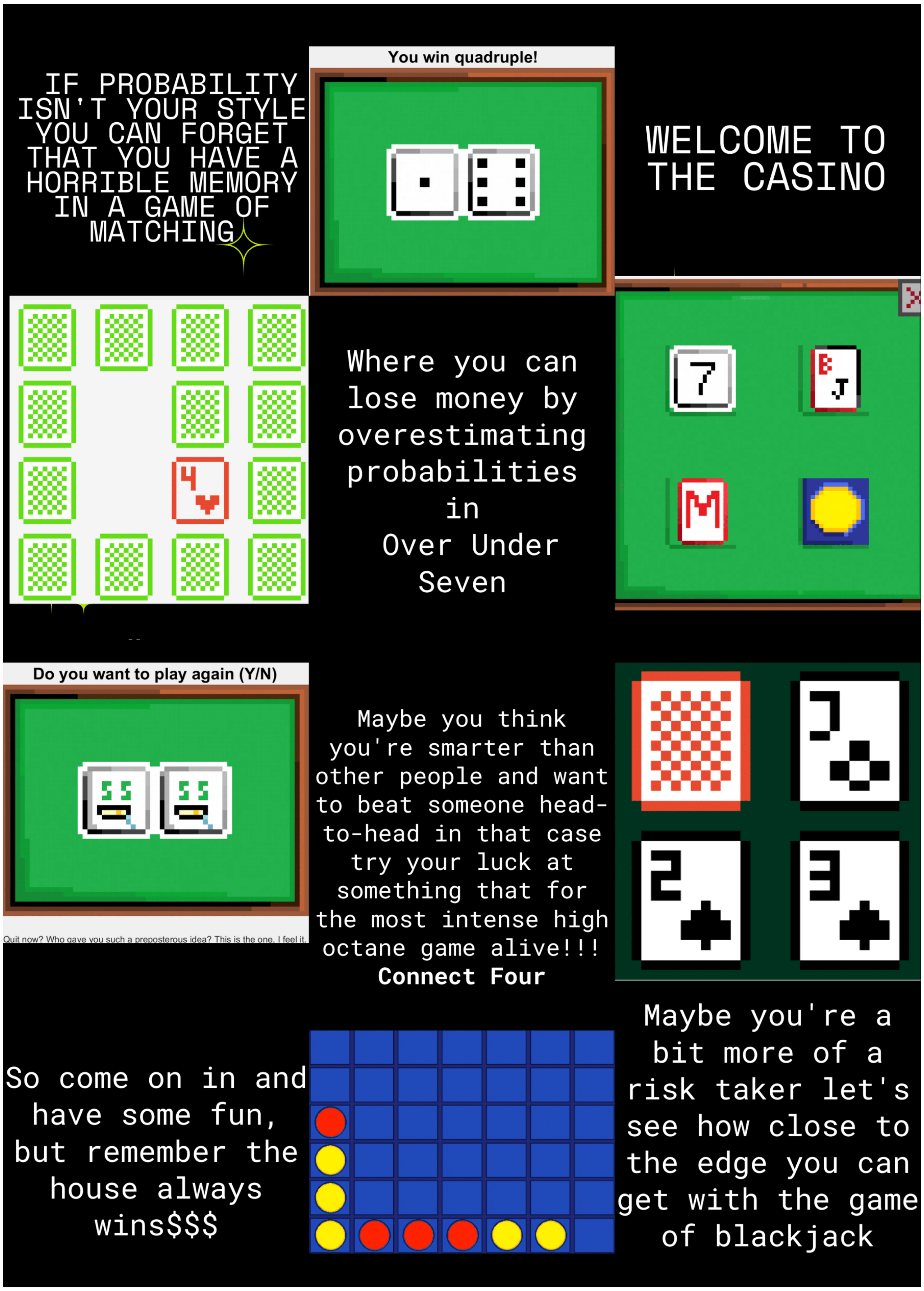
a. User Interview 1

1. What are some games you have played recently?
    a. Plants vs zombies
2. What are some games you might enjoy playing in MATLAB?
    a. Call of Duty
3. In your opinion, what is the optimal length a game should take to play?
    a. 5 minutes
4. What aspects of a game make it enjoyable to play?
    a. Interactive with the user. Good graphics and music are also great to pull the user in.

b. User Interview 2

1. What are some games you have played recently?
    a. Splatoon 3
2. What are some games you might enjoy playing in MATLAB?
    a. I would enjoy playing poker in MATLAB
3. In your opinion, what is the optimal length a game should take to play?
    a. 30 to 60 minutes
4. When playing this game, which aspects did you enjoy?
    a. The music and visuals
5. Based on playing this game, what changes would you recommend?
    a. Make sure the play again is visible and not hidden

2. Electronic Advertisement


IF PROBABILITY ISN'T YOUR STYLE YOU CAN FORGET THAT YOU HAVE A HORRIBLE MEMORY IN A GAME OF MATCHING
You win quadruple!
WELCOME TO THE CASINO
Where you can lose money by overestimating probabilities in Over Under Seven
Do you want to play again (Y/N)
Quit now? Who gave you such a preposterous idea? This is the one. I feel it.
Maybe you think you're smarter than other people and want to beat someone head-to-head in that case try your luck at something that for the most intense high octane game alive!!!
Connect Four
So come on in and have some fun, but remember the house always wins$$$
Maybe you're a bit more of a risk taker let's see how close to the edge you can get with the game of blackjack

3. Pitch Video Link

The Sphinx’s pitch video can be accessed at: https://youtu.be/g30I01XQ7yM

Software Documentation

1. Introduction

In the software Design Project, each team was tasked with creating a game using the programming language, MATLAB. Group A, The Sphinxes, opted to make four games and compile them into a single game selection screen, coining it as “The Sphinx Game Suite.” The four games in the suite are as follows: Over Under 7, Blackjack, Matching, and Connect 4. The team has produced documentation for each of the four games. Documentation will include a User Manual, Program Description for Developers, Final Algorithm, Flowchart, or Pseudocode, Discussion, and Recommendations.

2. User Manual

<u>Over Under 7:</u>

Objective: the objective is to gain money by correctly predicting the outcome of dice rolls.

How to play:

1. Start
    - First, enter the amount of money you want to start with. This is how much money you will have available to wager
2. Make your wager
    - Each round, you decide how much money to wager. This can be any amount less than or equal to the total funds you have available.
3. Make your prediction
    - Predict whether the sum of the two dice will be over (more than), under (less than), or exactly on 7
4. Roll the Dice
    - The game rolls two virtual dice and the sum is revealed. If your prediction is correct on over or under, you win the amount of money you wagered. If it is correct on exactly 7, you win 4x the amount of money you wagered. If your prediction was incorrect, you lose the money you wagered.
5. Throughout the game, you can expect to see fun, educational facts about probabilities, economics, and other topics.
6. Keep playing or quit
    - After each round you will be prompted to either quit then and walk away with your current funds or play another round.
7. Game over
    - If you run out of money, you are forced to quit

<u>Blackjack:</u>

In this implementation of Blackjack into a MATLAB program done by Diego Rivera Cartagena the things the user should know are the following. In this manual it will be described what is going on behind the scenes on each screen that pops up for the user. The first screen once you start playing the game is where the user the amount of money he is betting. The user should know that he is always going all in so he cannot choose that amount of money and then choose a smaller portion to bet. Once the user picks a number the program then proceeds to set the initial balance. The next screen is where the user sees his hand and one of the dealer's cards while the other one is face down. To be able to make this screen the program shuffled the deck and then gave the dealer and the user its hands where it makes calculations to see the value of their hands. In this screen the user must hit left click to hit or right click to stand.

After the user makes his choice, the program calculate the program then checks if the user won, tied or loss. If the user tied, then it shows a screen telling the user he tied and he then continues playing. If the user won, it shows a screen telling the user he won and telling him he must continue playing until he loses. Then the screen where it tells the user he lost along with a message based on the initial balance. Finally, after the you lost screen, it gives the user a screen where he chooses to play again or not.

Matching:

To start the game, there is a 4x4 array of cards facing down being displayed. From here the user must use the mouse input to choose the card desired. After selecting one card, the user must select a second card to match the first selection. If the two selected cards do not match, they flip face down. On the other side if the two selected cards do match, they will stay flipped over and displayed. From here the user repeats this process until all the matches are found. When completed the user will be asked if they would like to play again.

Connect 4:

How is connect four played? Connect four is a two-player game. One of the players has red chips and the other player has yellow chips. They take turns placing chips and one of seven columns. The chips fall to the bottom of the board or to the first unoccupied space. The goal is to get four of the chips in a in a line either vertically or horizontally or diagonally. In this online connect four game is played by typing the numbers one through seven.

Program Description for Developers

Over Under 7:

Program Overview

The Over Under Seven program is a MATLAB based interactive dice game in which the player wagers money and predicts whether the sum of the two dice rolls will be under, over, or exactly on seven. The program makes use of built-in MATLAB functions such as fprintf(), input(), and randi() as well as commands from a "simple game engine" that was provided to us by the engineering department. Loops were used to manage validation of inputs, and to make the game re-playable until the player runs out of funds.

Program Flow

The program begins by displaying instructions that are stored in a function and prompting the player for how much money they want to start with. A while loop ensures the input for the starting funds is valid by rejecting the input and asking for a new one until it is valid.

Next, the program enters a Gameplay loop that loops while the player requests to keep playing. The player is asked how much they want to wager (this is portion of the program is stored in a function) and whether they think the roll will be over, under or exactly on seven.

Next, based on the roll and prediction, the funds the player has are updated using a function containing an if-elseif-else structure to determine if they won or lost.

Finally, depending on how much money the player has left, they can either continue playing or end the game. If the player is out of funds, a game over screen (stored in a function) is displayed and the program exits.

Variable Names and Uses

Variable	Use
play	Controls whether the player goes back into the gameplay loop or exits the game.
ifund	Variable for how much money the player has to start. It is also used as the current fund variable as the game progresses.
wager	Variable for how much the player wants to bet.
prediction	Variable for what the player predicts the roll will be: over, under, or exactly on seven.
sumroll	Variable that contains the sum of the two dice values
roll	Array containing the individual dice rolls (2 elements)
sceneX	Game engine objects used to display game boards for various stages
Board	Matrix that pulls data from sprite sheet to show images to the player
allowed	Variable used in the wagerLoop function to determine if input is valid

Provided MATLAB Game Commands Used

simpleGameEngine

Initializes the game engine and allows for customization of image dimensions and colors based off a sprite sheet image.

drawScene

Displays the game board layout based on the previously provided board matrix and sprite image.

getKeyboardInput

Stops the script from executing until the player presses a key. It was used for transitioning between scenes.

Example: getKeyboardInput(sceneX);

title and xlabel

Add text elements to the game window for instructions, prompts, or feedback.

Functions and Their Roles

Main Function

Handles the core gameplay loop, interacting with the supporting functions.

win_lose_scrn

Determines the outcome of the round based on the player's prediction and dice roll. Updates the player's funds and displays win/lose messages.

wagerLoop

Validates the wager amount to ensure it is positive and within the player's available funds.

scene1Func

Initializes the game by displaying the opening scene and prompting the player for their starting funds.

instrFunc

Prints the game instructions to the command window.

gameOverScreen

Displays a game over screen when the player runs out of funds.

References

"SoftwareDesignProject_Procedure.pdf" Available at: https://carmen.osu.edu/ [Accessed Nov. 2024]

"SDP Lab 1_Presentation.pdf" Available at: https://carmen.osu.edu/ [Accessed Nov. 2024]

"SDP Resources" Available at: https://carmen.osu.edu/ [Accessed Nov. 2024]

Gilat, A., *MATLAB: An Introduction with Applications 5th Edition*. Hoboken, NJ: Wiley, 2015.

U.S. Department of the Treasury, "The debt to the penny and who holds it," [Online]. Available: https://fiscaldata.treasury.gov. [Accessed Nov. 18, 2024].

Federal Reserve Bank of New York, "Household debt and credit report," 2024. [Online]. Available: https://www.newyorkfed.org/microeconomics/hhdc.html. [Accessed Nov. 18, 2024].

Blackjack:

Program Description:
This section provides a technical description of the program's functionality, highlighting the core logic and structure of the Blackjack game in MATLAB.

Variables:

1. **playerBalance**: Keeps track of the player's current balance after each round.
2. **bet**: The amount wagered by the player in each round.
3. **playerHand**: A vector containing the cards held by the player.
4. **realPlayerHand**: A vector representing the actual values of the player's cards.
5. **dealerHand**: A vector containing the cards held by the dealer.
6. **realDealerHand**: A vector representing the actual values of the dealer's cards.
7. **playAgain**: A boolean variable that stores whether the player wants to play again after a round.
8. **singleAce**: A boolean used to determine if an ace should be counted as 11 or 1 in the hand.

MATLAB Game Commands Used:

- figure: Creates a figure window for displaying game information.
- uicontrol: Creates user interface controls such as buttons and text on the figure.
- uiwait: Pauses the execution of the game, awaiting user input.
- uiresume: Resumes execution after the user interacts with the interface.
- close: Closes the figure window after the game or interaction is completed.

The program uses a combination of user interface elements and logic to simulate the Blackjack game. The game involves creating windows to display the player's and dealer's hands, providing buttons for player actions (hit, stand), and updating the player's balance based on the outcome of each round.

References

"SoftwareDesignProject_Procedure.pdf" Available at: https://carmen.osu.edu/ [Accessed Nov. 2024]

"SDP Resources" Available at: https://carmen.osu.edu/ [Accessed Nov. 2024]

Gilat, A., *MATLAB: An Introduction with Applications 5th Edition*. Hoboken, NJ: Wiley, 2015.

Matching:

Matching is a game that requires memory to complete the task of pairing the matching cards. It was created simpleGameEngine and retro_card.png which were given in class. It features a 4-row by 4-column grid, consisting of 8 pairs of cards that were randomly picked. To create our graphics, we used the drawScene command. We utilized the function "cardValues = cardValues(randperm(16))" to shuffle the cards, ensuring that the cards would be randomized with a different layout each time the game is ran. Players would be able to interact with the game by selecting the cards with their mouse to flip them over and reveal the card itself. The command used to complete this was getmouseInput. When clicked, the face of the card is revealed to show its value while the second card clicked should be an attempt to match the first cards value. If the cards are not a match, the cards are then flipped over to show the backs instead of the card's values. If the cards are a successful match, the cards are then removed from the display. Once all the cards have been matched, the game is then ended and will display a congratulatory message which also informs the player how many attempts they took to match all the cards.

a. MemoryGame=stores simpleGameEngine
b. playAgain=controls whether the games should restart. If determined true, the game will loop to play again until the player decides to quit
c. intro=stores the text object to show the introductory instructions or information at the beginning.
d. cardBack=stores the random index of the cards when flipped over and is randomly picked from values 4 to 10
e. displayArray=the 4x4 matric which displays the cards utilized in the game
f. numPairs=shows the number of unique card pairs used in the game. Currently set to 8 to make 8 card pairs that fit into the 4x4 grid.
g. uniqueCards=holds a set of 8 card values that are randomly selected from 21 to 72
h. cardValues=array that stores the values of all the cards utilized in the game
i. cardArray=reshapes the cardValues array into a 4x4 grid pattern
j. numMatches=counts the number of successful matches made by the player
k. playagainInput=stores the players input when asked if they would like to play again

Provided game commands=
Sprite sheet=utilized for our graphics to visualize our cards
simpleGameEngine=stores multiple different functions that are into one
drawScene=displays the sprite sheets or our cards
getMouseInput=allows the player to use their mouse to click to interact with the game
retro_card.png=in simpleGameEngine and utilized for the sprite sheet.

References

"SoftwareDesignProject_Procedure.pdf" Available at: https://carmen.osu.edu/ [Accessed Nov. 2024]
"SDP Resources" Available at: https://carmen.osu.edu/ [Accessed Nov. 2024]
Gilat, A., *MATLAB: An Introduction with Applications 5th Edition*. Hoboken, NJ: Wiley, 2015.

<u>**Connect 4:**</u>

- This code utilizes a number of variables including:
- my_scene
  - this helps to draw the scenes
- empty_sprite
  - this one is used to identify an empty space
- red_sprite
  - this one represents one of the players
    - we change the color
- black_sprite
  - this one represents another player
- conue
  - this one helps me to determine if they want to keep playing the game
- bord
  - this one helps me to determine where all the pieces are and helps keep track of the board it also helps determine when there is a winner
- noWiner
  - this one helps me determine who the winner is
- i
  - this one just keeps track of accounting for loops it also helps me determine whose turn it is
- board_display
  - this one shows the board and keeps track of what to show on the board
- good
  - this one determines if the move was a legal move
- move
  - this one determines what the move was
- player
  - this one tells you which player turn it is
- j
  - this one just helps management and numbers it's mostly used in for loops and nested for loops
- win
  - this helps determine if someone has one
- WIN
  - this would also determines if someone one but the person wasn't working very well.

Connect for only has a few game commands

- move=input("make you moves");
  - All this does is determine what move you made
- conue=input("\n Do you want play agen (Y/N): ","s");
  - This just asks if you want to keep playing

References
“SoftwareDesignProject_Procedure.pdf” Available at: https://carmen.osu.edu/ [Accessed Nov. 2024]
“SDP Resources” Available at: https://carmen.osu.edu/ [Accessed Nov. 2024]
Gilat, A., *MATLAB: An Introduction with Applications 5th Edition*. Hoboken, NJ: Wiley, 2015.

3. Final Algorithm or Flowchart or Pseudocode

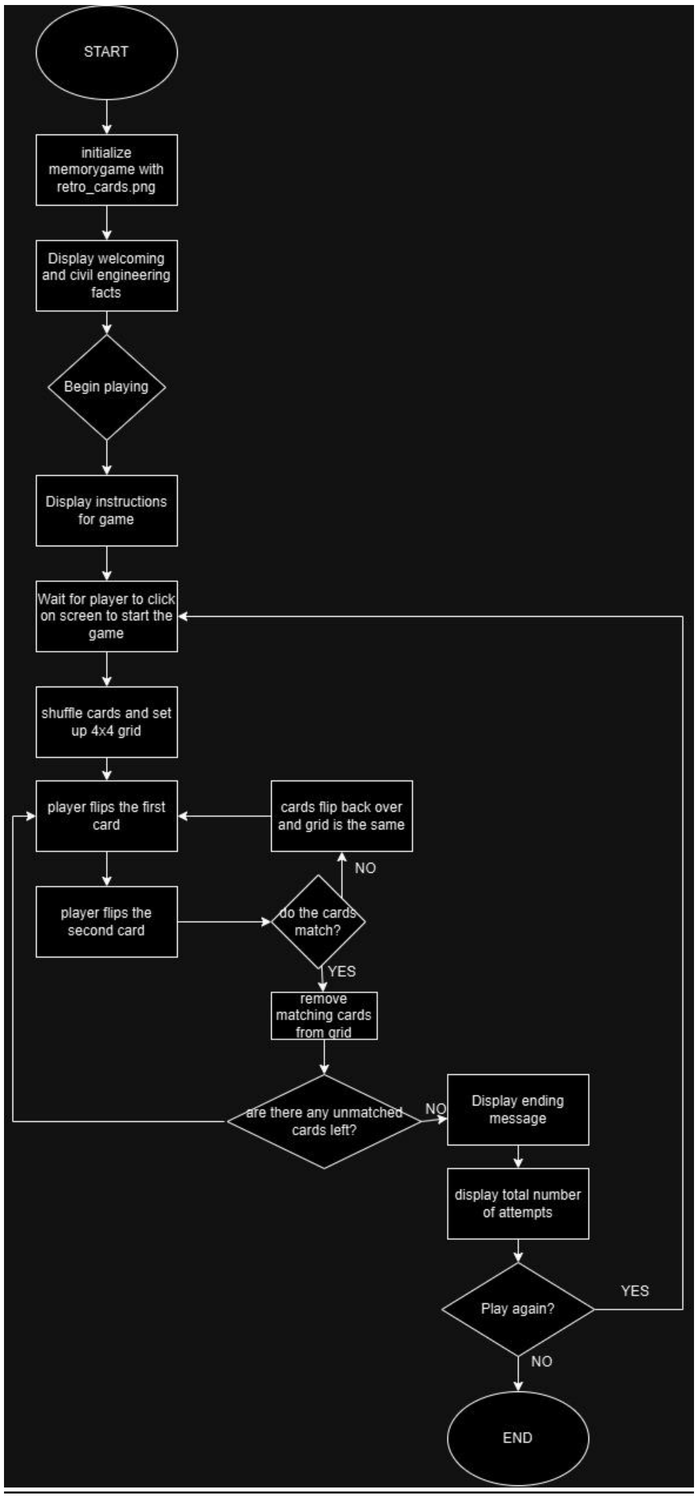
Over Under 7:

```
1. Initiate game setup
    a. Set up the game environment
        i. Dock figures
        ii. Disable associated warnings
    b. Print instruction statement
    c. Get input for starting funds
        i. If input is invalid, print an error and return to 1c
    d. Initiate a variable indicating whether or not to initiate game loop
2. Enter a gameplay loop
    a. Get input for wager amount
        i. If input is invalid, print an error and return to 2a
    b. Get input for whether the roll will be over under or exactly on 7
        i. If input is invalid, print an error and return to 2b
    c. Create a vector with two random integer values to be dice values
    d. Calculate the sum of the dice values
    e. Evaluate the player’s prediction
        i. If the player guessed over or under correctly, add their wager to their funds
           and continue on
        ii. If the player guessed exactly on correctly, add 4x their wager to their funds
           and continue on
        iii. If the player guessed incorrectly, subtract the wager amount from their funds
           and continue on
    f. Print the dice roll values, whether the player won or lost, and their remaining funds
        i. If the player is out of money, display a game over screen and continue to step
           3
        ii. If the player has remaining funds, get input for whether or not they want to
           keep playing
            A. If the player wants to stop, continue to step 3
            B. If the player wants to keep playing, return to step 2a
3. Close the game
```

<u>**Blackjack:**</u>

1. Initialize Game:
    - Set initial player balance.
    - Prompt user for the bet amount.
2. Deal Cards:
    - Deal two cards to the player and dealer each.
    - Display the cards to the user.
3. Player's Turn:
    - Prompt the player to either "hit" or "stand".
    - If "hit", add a card to the player's hand.
    - If "stand", move to the dealer's turn.
4. Dealer's Turn:
    - The dealer must hit until their hand is at least 17.
    - If the dealer busts (hand value > 21), the player wins.
5. Determine Winner:
    - If the player’s hand exceeds 21, they lose.
    - If the dealer’s hand exceeds 21, the player wins.
    - If the player’s hand is higher than the dealer's, the player wins.
    - If the hands are equal, it's a tie.
6. End of Round:
    - Update the player’s balance based on the round’s outcome.
    - Ask the player if they want to play again.
    - If "yes", reset the game and return to step 2. If "no", end the game.

Matching:


START
initialize memorygame with retro_cards.png
Display welcoming and civil engineering facts
Begin playing
Display instructions for game
Wait for player to click on screen to start the game
shuffle cards and set up 4x4 grid
player flips the first card
cards flip back over and grid is the same
NO
player flips the second card
do the cards match?
YES
remove matching cards from grid
are there any unmatched cards left?
NO
Display ending message
display total number of attempts
Play again?
YES
NO
END

<u>Connect 4:</u>

- write instructions and interesting facts
- a loop that determines if anyone has won and if they want to play again and if the game is still going
    - get the players' move
    - determine if the player is player one or player two
        - a loop that checks to see if the move was a legal move
            - check to see if the move is a legal move and if the move can be placed in each one of the columns repeat for each column until there are no rows left
            - if none of the rows are free right that is not a valid move and asked the player to make another move
    - draw the board
    - see if someone has won the game yet
        - check to see if someone one in the horizontal direction
            - for 6 loops
                - for 4 loops
                    - check if there are four pieces in a row
                    - if there are that player won
        - check to see if someone won in the vertical direction
            - for 3 loops
                - for 7 loops
                    - check if there are four pieces in a collum
                    - if there are that player won
        - check to see if someone won in the diagonal direction
            - for 3 loops
                - for 4 loops
                    - check if anyone one in the diagonal direction in from the bottom left corner to the top right corner
                    - if they did, they won
            - for 3 loops
                - for 4 loops
                    - check if anyone one in the diagonal direction in from bottom right corner to the top left corner
                    - if they did, they won
    - see who wanted
        - and prints that they won
        - ask if they want to continue
    - if they wanted to continue
        - restart the loop and reset the board

4. Final Program with Comments

Over Under 7 Script:

```
clc
clear
%dock generated figures
set(0, 'DefaultFigureWindowStyle', 'docked');
%disable unnecessary warning about image magnification
warning('off', 'images:imshow:magnificationMustBeFitForDockedFigure');
%initialize play variable to be used later
play = 'Y';

%use instruction function for instructions
instrFunc

ifund = scene1Func();
%error loop for invalid inputs
while ifund < 0
    fprintf('Invalid input. Must input a positive number\n')
    %shift focus to command window
    commandwindow;
    ifund = input('How much money would you like to start with?: ');
end


%gameplay loop
while(play == 'Y' && ifund>0)
    % Initialize scene3, wager amount
    scene3 = simpleGameEngine('noice.png',16,16,15);
    % Create board background
        Board = [7,8,9,10;
            11,12,13,14;
            15,16,17,18];
    drawScene(scene3,Board);
    % Add text elements and get input for the wager amount
    title('How much do you wager?', 'FontSize', 20, 'FontWeight', 'bold');
    xlabel('Input the number value and press enter to continue')

    %call get valid wager input function
    wager = wagerLoop(ifund);

    % Add text elements and get input for the prediction
    title('Will the roll be over, under, or exactly on 7? (O, U, E)', 'FontSize', 20, 'FontWeight',
'bold');
    xlabel('Press any key to continue')
    %shift focus to command window
    commandwindow;
    prediction = input('Will the roll be under, over, or exactly on 7 (U, O, E)?: ', 's');
    prediction = upper(prediction);

    %error loop for invalid inputs
    while prediction ~= 'U' & prediction ~= 'O' & prediction ~= 'E' % single & because double was throwing
errors instead of my message when input was a multi digit number
        fprintf('Invalid input. Must input "U", "O", or "E"\n')
        %shift focus to command window
        commandwindow;
        prediction = input('Will the roll be under, over, or exactly on 7 (U, O, E)?: ', 's');
        prediction = upper(prediction);
    end
    close

    % Initialize scene5, Winner/Loser status
    scene5 = simpleGameEngine('noice.png',16,16,15);
    % Random roll of 2 dice (1 row by 2 columns) - only allow values up to 6
    roll = randi(6, 1, 2);
    % Create board background
    Board = [7,8,9,10;
```

```
        11, roll, 14;
        15,16,17,18];
    drawScene(scene5,Board);
    sumroll = sum(roll);

% if-elseif-else structure function for win\lose
    ifund = win_lose_scrn(prediction, sumroll, ifund, wager);

    %Get input to move on
    getKeyboardInput(scene5);
    close

    fprintf('Your remaining funds are: %i\n', ifund)

    %if-elseif-else structure for game over
    if ifund == 0
        %call game over screen function
        gameOverScreen()

    elseif ifund ~= 0

        %initialize scene7, Play again or not
        scene7 = simpleGameEngine('noice.png',16,16,15);
        % Create board background
        Board = [7,8,9,10;
            11,20,20,14;
            15,16,17,18];
        drawScene(scene7,Board);
        %add descriptive text elements
        title('Do you want to play again (Y/N)', 'FontSize', 20, 'Fontweight', 'bold');
        xlabel('Quit now? Who gave you such a preposterous idea? This is the one, I feel it.');

        %shift focus to command window
        commandwindow;
        %ask user if they want to play again
        play = input('Do you want to play again (Y/N)?: ', 's');
        play = upper(play);
        %throw error message for invalid inputs and ask again
        while play ~= 'Y' & play ~= 'N'
            fprintf('Invalid input. Must input "Y" or "N"\n')
            %shift focus to command window
            commandwindow;
            play = input('Do you want to play again (Y/N)?: ', 's');
            play = upper(play);
        end
        close
    end

end

%ensure all figures have closed
close all
```

Over Under 7 Functions:

```
function instrFunc()

%prints set of instructions for the user
fprintf('\n*** Welcome to Over/Under Seven! ***\n');
fprintf('Here are the rules:\n');
fprintf('1. You start with an initial amount of money that you choose.\n');
fprintf('2. In each round, you can wager an amount up to your current funds.\n');
fprintf('3. Predict whether the sum of two dice rolls will be:\n');
fprintf('    - "U" for Under 7\n');
fprintf('    - "O" for Over 7\n');
fprintf('    - "E" for Exactly 7\n');
fprintf('4. Payouts:\n');
fprintf('    - Under or Over 7: Win = wager amount added; Lose = wager amount deducted.\n');
fprintf('    - Exactly 7: Win = 4 times your wager!\n');
fprintf('5. The game ends when you run out of money or choose to quit.\n');
fprintf('\nGood luck!\n\n');

end
```

```
function ifund = scene1Func()

    % Initialize scene1, intro
    scene1 = simpleGameEngine('noice.png',16,16,15);
    % Create board background
        Board = [7,8,9,10;
            11,12,13,14;
            15,16,17,18];
    drawScene(scene1,Board);
    % Add text elements
    title('Over/Under Seven', 'FontSize', 20, 'FontWeight', 'bold');
    xlabel('Press any key to continue');
    %Get input to move on
    getKeyboardInput(scene1);

    % Add text elements and get input for starting money
    title('How much money would you like to start with?', 'FontSize', 20, 'FontWeight', 'bold');
    xlabel('Input the number value and press enter to continue')
    %shift focus to command window
    commandwindow;
    ifund = input('How much money would you like to start with?: ');
end
```

```
function wager = wagerLoop(ifund)
    %loop to stop betting more money than you have
    allowed = 0;
    while allowed == 0
        %shift focus to command window
        commandwindow;
        wager = input('How much do you wager?: ');

        %error loop for invalid inputs
        while wager < 0
            fprintf('Invalid input. Must input a positive number\n')
            %shift focus to command window
            commandwindow;
            wager = input('How much do you wager?: ');
        end

        if wager > ifund
            fprintf('Insufficient funds, pick a new wager\n')
            allowed = 0;
        else
            allowed = 1;
        end
    end
end
```

```
function ifund = win_lose_scrn(prediction, sumroll, ifund, wager)
    % Function to evaluate game outcome based on prediction and dice roll
    % Inputs:
    %   prediction - Player's prediction ('U', 'O', 'E')
    %   sumroll    - Sum of the dice rolls
    %   ifund      - Current funds of the player
    %   wager      - Player's wager
    %
    % Output:
```

```
    %   ifund      - Updated funds after win/lose

    % Check prediction and update funds accordingly
    if prediction == 'U' && sumroll < 7
        ifund = ifund + wager;
        title('You win!', 'FontSize', 20, 'FontWeight', 'bold');
        xlabel('Click to continue');
    elseif prediction == 'U' && sumroll >= 7
        ifund = ifund - wager;
        title('You lose :(', 'FontSize', 20, 'FontWeight', 'bold');
        xlabel('Press any key to continue');
    elseif prediction == 'O' && sumroll > 7
        ifund = ifund + wager;
        title('You win!', 'FontSize', 20, 'FontWeight', 'bold');
        xlabel('Press any key to continue');
    elseif prediction == 'O' && sumroll <= 7
        ifund = ifund - wager;
        title('You lose :(', 'FontSize', 20, 'FontWeight', 'bold');
        xlabel('Press any key to continue');
    elseif prediction == 'E' && sumroll == 7
        ifund = ifund + 4 * wager;
        title('You win quadruple!', 'FontSize', 20, 'FontWeight', 'bold');
        xlabel('Press any key to continue');
    elseif prediction == 'E' && sumroll ~= 7
        ifund = ifund - wager;
        title('You lose :(', 'FontSize', 20, 'FontWeight', 'bold');
        xlabel('Press any key to continue');
    end
end
```

```
function gameOverScreen()
        % Initialize scene6, Game Over Screen
        scene6 = simpleGameEngine('noice.png',16,16,15);
        % Create board background
        Board = [7,8,9,10;
            11,19,19,14;
            15,16,17,18];
        drawScene(scene6,Board);
        %add descriptive text elements
        title('Womp Womp ** GAME OVER ** Womp Womp', 'FontSize', 20, 'FontWeight', 'bold');
        xlabel('See kids? gambling is bad. Now go do the hokey pokey and turn yourself around.');
        %Get input to move on
        pause(5)
        close;
end
```

Blackjack Script:

```
clear;
clc;

playAgain = true;
while playAgain

playerBalance = openingScreen();
[singleAce,currentCard, playerHand, realPlayerHand, dealerHand, realDealerHand, showDealer, finalOutput,
blkjSzn, empty_sprite, blank_line, empty_card1, empty_card2, backD, backP, dealerLine, playerLine] =
initialisation(); %intitialize variables



while playerBalance > 0 % keep repeating game until player does not have any money left

    % first turn functions
    [deck] = randperm(72 - 21 + 1) + 21 - 1; % build deck
    [bet] = playerBalance; % get player bet

    [playerHand, currentCard, deck] = addCard(currentCard, deck, playerHand);
    [playerHand, currentCard, deck] = addCard(currentCard, deck, playerHand); % add two cards to player
hand
    [realPlayerHand] = cardFinder(playerHand); % convert reference cards to actual cards

    [dealerHand, currentCard, deck] = addCard(currentCard, deck, dealerHand);
    [dealerHand, currentCard, deck] = addCard(currentCard, deck, dealerHand); % add two cards to dealer
hand
    [realDealerHand] = cardFinder(dealerHand); % convert reference cards to actual cards


    [realDealerHand] = aceConversion(realDealerHand, singleAce); % conver aces to 11 when needed
    [realPlayerHand] = aceConversion(realPlayerHand, singleAce); % conver aces to 11 when needed

    currentStatus(playerHand, dealerHand,realPlayerHand, realDealerHand, showDealer, finalOutput,
playerBalance, blkjSzn, empty_sprite, blank_line, empty_card1, empty_card2, backD, backP, dealerLine,
playerLine, bet); % output current status

    % player choice function
    [choice] = playerChoice(blkjSzn);

    while isequal(choice, 'hit') % while loop checks for player move

        [playerHand, currentCard, deck] = addCard(currentCard, deck, playerHand); % add 1 more card to the
player hand
        [realPlayerHand] = cardFinder(playerHand); % convert cards
        [realPlayerHand] = aceConversion(realPlayerHand, singleAce); % convert aces when needed
        currentStatus(playerHand, dealerHand, realPlayerHand, realDealerHand, showDealer, finalOutput,
playerBalance, blkjSzn, empty_sprite, blank_line, empty_card1, empty_card2, backD,backP, dealerLine,
playerLine, bet)

        if sum(realPlayerHand) > 21 % player busted
            break
        end

        [choice] = playerChoice(blkjSzn);
    end

    % dealer functions
    if sum(realPlayerHand) <= 21

            % dealer choice options
            while sum(realDealerHand) <= 16 % dealer hits while his hand is less or equal than 16
                [dealerHand, currentCard, deck] = addCard(currentCard, deck, dealerHand); % dealer hits
function
                [realDealerHand] = cardFinder(dealerHand); % convert cards
            end

            [realDealerHand] = aceConversion(realDealerHand, singleAce); %  convert Ace
            showDealer = true; % change boolean ot show dealer card
            currentStatus(playerHand, dealerHand, realPlayerHand, realDealerHand, showDealer, finalOutput,
playerBalance, blkjSzn, empty_sprite, blank_line, empty_card1, empty_card2, backD, backP, dealerLine,
playerLine, bet); % output current status
            showDealer = false; % change boolean again for next round
    end

    % player win status, bet calculations
    [playerBalance, playerHand, realPlayerHand, dealerHand, realDealerHand] =
winCalculator(realPlayerHand, realDealerHand, playerBalance, bet);

end

% final outputs
finalOutput = true; % final status
currentStatus(playerHand, dealerHand, realPlayerHand, realDealerHand, showDealer, finalOutput,
playerBalance, blkjSzn, empty_sprite, blank_line, empty_card1, empty_card2, backD, backP, dealerLine,
playerLine,bet)

playAgain = again(playAgain);
end
```

Blackjack Functions:

```
function [hand] = aceConversion(hand, singleAce)
% converts ace to 11 if necessary

    for i = 1 : length(hand)
            if hand(i) == 1 && singleAce == false % converts only one ace if there are more than two in a
hand
                hand(i) = 11;
                singleAce = true;
            end
            if sum(hand) > 21 && hand(i) == 11
                hand(i) = 1;
            end
    end
end
```

```
function [hand, currentCard, deck] = addCard(currentCard, deck, hand)
% adds card to hand

if currentCard >= 48 % if the current card is more thant the 48th shuffle the deck again to ensure it doe
snot end
    deck = shuffleDeck();
    currentCard = 1;
end

% read in the length of the hand and append new card to it
hand(length(hand)+1) = deck(currentCard);
% update current card
currentCard = currentCard + 1;

end
```

```
function playAgain = again(playAgain)

    close all;
    % Create a figure window
    fig = figure('Name', 'Again?', 'NumberTitle', 'off');

    % Add a title at the top
    titleText = uicontrol('Style', 'text', ...
        'String', 'Waste Money Again?', ...
        'FontSize', 40, ...
        'FontWeight', 'bold', ...
        'ForegroundColor', 'magenta', ...
        'BackgroundColor', get(fig, 'Color'), ...
        'Position', [475, 300, 500, 140]); % [x, y, width, height]

    % Add buttons for the three options
    buttonAgain = uicontrol('Style', 'pushbutton', ...
        'String', 'YES', ...
        'FontSize', 15, ...
        'Position', [650, 250, 150, 50], ...
        'Callback', @(src, event) setAgain(true));


    buttonExit = uicontrol('Style', 'pushbutton', ...
        'String', 'NO', ...
        'FontSize', 15, ...
        'Position', [650, 150, 150, 50], ...
        'Callback', @(src, event) setAgain(false));

    % Variable to store the selected balance

        function setAgain(choice)
            % Callback to set the balance and close the figure
            playAgain = choice;
            uiresume(fig); % Resume execution after button click
        end

    % Pause execution until a button is clicked
    uiwait(fig);

    % Assign the selected balance to the output

    % Close the figure
    close(fig);

end
```

```
function [realHand] = cardFinder(hand)
    % Converts card indices to face values and suits for a deck of cards.
    % hand: Array of card indices (21 to 72)
    % realHand: Numeric values (1-10) for blackjack
    % suit: Cell array of suits ('Hearts', 'Diamonds', 'Clubs', 'Spades')

    n = length(hand);
    realHand = zeros(1, n); % Initialize hand values

    for i = 1:n
        card = hand(i);

        if card >= 21 && card <= 30
            realHand(i) = card - 20;
        elseif card >= 31 && card <= 33
            realHand(i) = 10;
        elseif card >= 34 && card <= 43
            realHand(i) = card - 33;
        elseif card >= 44 && card <= 46
            realHand(i) = 10;
        elseif card >= 47 && card <= 56
            realHand(i) = card - 46;
        elseif card >= 57 && card <= 59
```

```
                realHand(i) = 10;
                elseif card >= 60 && card <= 69
                realHand(i) = card - 59;
            elseif card >= 70 && card <= 72
                realHand(i) = 10;
            end
        end
    end
```

```
function [] = currentStatus(playerHand, dealerHand, realPlayerHand, realDealerHand, showDealer,
finalOutput, playerBalance, blkjSzn, empty_sprite, blank_line, empty_card1, empty_card2, backD, backP,
dealerLine, playerLine,bet)
% outputs the dealers and the players hand

    if finalOutput == false % choose for final or not output
        if showDealer == false % choose for final dealer output in full
            playerLen = length(playerHand);
                dealerLine = [empty_card2, dealerHand(2), 1, 1, 1, 1];

                if playerLen == 6
                    playerLine = [playerHand(1), playerHand(2), playerHand(3), playerHand(4),
playerHand(5), playerHand(6)];
                    backP = [empty_card1, empty_card1, empty_card1, empty_card1, empty_card1,
empty_card1];
                elseif playerLen == 5
                    playerLine = [playerHand(1), playerHand(2), playerHand(3), playerHand(4),
playerHand(5), 1];
                    backP = [empty_card1, empty_card1, empty_card1, empty_card1, empty_card1, 1];
                elseif playerLen == 4
                    playerLine = [playerHand(1), playerHand(2), playerHand(3), playerHand(4), 1, 1];
                    backP = [empty_card1, empty_card1, empty_card1, empty_card1, 1, 1];
                elseif playerLen == 3
                    playerLine = [playerHand(1), playerHand(2), playerHand(3), 1, 1, 1];
                    backP = [empty_card1, empty_card1, empty_card1, 1, 1, 1];
                elseif playerLen == 2
                    playerLine = [playerHand(1), playerHand(2), 1, 1, 1, 1];
                    backP = [empty_card1, empty_card1, 1, 1, 1, 1];
                end
                drawScene(blkjSzn, [backD; backP], [dealerLine; playerLine])
                title('Left Click to Hit / Right Click to Stand', 'FontSize', 20, 'Color', 'r');
                fprintf('You have %0.0f\n\n', sum(realPlayerHand))


        else
            playerLen = length(playerHand);
            dealerLen = length(dealerHand);

                if dealerLen == 6
                    dealerLine = [dealerHand(1), dealerHand(2), dealerHand(3), dealerHand(4),
dealerHand(5), dealerHand(6)];
                    backD = [empty_card1, empty_card1, empty_card1, empty_card1, empty_card1,
empty_card1];
                elseif dealerLen == 5
                    dealerLine = [dealerHand(1), dealerHand(2), dealerHand(3), dealerHand(4),
dealerHand(5), 1];
                    backD = [empty_card1, empty_card1, empty_card1, empty_card1, empty_card1, 1];
                elseif dealerLen == 4
                    dealerLine = [dealerHand(1), dealerHand(2), dealerHand(3), dealerHand(4), 1, 1];
                    backD = [empty_card1, empty_card1, empty_card1, empty_card1, 1, 1];
                elseif dealerLen == 3
                    dealerLine = [dealerHand(1), dealerHand(2), dealerHand(3), 1, 1, 1];
                    backD = [empty_card1, empty_card1, empty_card1, 1, 1, 1];
                elseif dealerLen == 2
                    dealerLine = [dealerHand(1), dealerHand(2), 1, 1, 1, 1];
                    backD = [empty_card1, empty_card1, 1, 1, 1, 1];
                end
                if playerLen == 6
                    playerLine = [playerHand(1), playerHand(2), playerHand(3), playerHand(4),
playerHand(5), playerHand(6)];
                    backP = [empty_card1, empty_card1, empty_card1, empty_card1, empty_card1,
empty_card1];
                elseif playerLen == 5
                    playerLine = [playerHand(1), playerHand(2), playerHand(3), playerHand(4),
playerHand(5), 1];
                    backP = [empty_card1, empty_card1, empty_card1, empty_card1, empty_card1, 1];
                elseif playerLen == 4
                    playerLine = [playerHand(1), playerHand(2), playerHand(3), playerHand(4), 1, 1];
                    backP = [empty_card1, empty_card1, empty_card1, empty_card1, 1, 1];
                elseif playerLen == 3
                    playerLine = [playerHand(1), playerHand(2), playerHand(3), 1, 1, 1];
                    backP = [empty_card1, empty_card1, empty_card1, 1, 1, 1];
                elseif playerLen == 2
                    playerLine = [playerHand(1), playerHand(2), 1, 1, 1, 1];
                    backP = [empty_card1, empty_card1, 1, 1, 1, 1];
                end
                drawScene(blkjSzn, [backD; backP], [dealerLine; playerLine])
                title('Left Click to Hit / Right Click to Stand', 'FontSize', 15, 'Color', 'r');
                fprintf('Dealer has %0.0f\n', sum(realDealerHand))
                fprintf('You have %0.0f\n\n', sum(realPlayerHand))
        end

    else % final outputs and graph maker
        if playerBalance == 0
            fprintf('You went from $%d to $0.\nMake better decisions next time\n\n', bet);
        end
    end
end
```

```
function [singleAce,currentCard, playerHand, realPlayerHand, dealerHand, realDealerHand, showDealer,
finalOutput, blkjSzn, empty_sprite, blank_line, empty_card1, empty_card2, backD, backP, dealerLine,
playerLine] = initialisation()

set(0, 'DefaultFigureWindowStyle', 'docked');
warning('off', 'images:imshow:magnificationMustBeFitForDockedFigure');

% outputes the rules of the game and intialises some variablesi
blkjSzn = simpleGameEngine('retro_cards.png', 16, 16, 30, [1, 50, 32]);
empty_sprite = 1;
blank_line = ones(empty_sprite, 6);
empty_card1 = 2;
empty_card2 = 4;
backD = [empty_card1, empty_card1, 1, 1, 1, 1] ;
backP = [empty_card1, empty_card1, 1, 1, 1, 1];
dealerLine = [empty_card2, empty_card2, 1, 1, 1, 1];
playerLine = [empty_card2, empty_card2, 1, 1, 1, 1];

% make a figure for the title and allow the user to pick his balance
%drawScene(blkjSzn, grid);
% Center the text on the game screen
%title('BLACKJACK', 'FontSize',15,'Color','r');


% intialize variables
singleAce = false;
currentCard = 1;
playerHand = [];
realPlayerHand = [];
dealerHand = [];
realDealerHand = [];
showDealer = false;
finalOutput = false;
end
```

```
function loss(bet)
% Creates a loss screen with a dynamic message based on the user's balance

% Create a larger figure window
fig = figure('Name', 'Blackjack - Loss Screen', ...
    'NumberTitle', 'off', ...
    'MenuBar', 'none', ...
    'ToolBar', 'none');

% Add the "You Lost" message at the top
lostText = uicontrol('Style', 'text', ...
    'String', 'You Lost!', ...
    'FontSize', 50, ... % Larger font size for the message
    'FontWeight', 'bold', ...
    'ForegroundColor', 'red', ...
    'BackgroundColor', get(fig, 'Color'), ...
    'Position', [475, 350, 450, 80]); % [x, y, width, height]

% Determine the dynamic message based on playerBalance
if bet < 1000
    messageText = 'You could have made a Solar-Powered Charger with that Money';
elseif bet < 100000
    messageText = 'You could have made your own drone with that Money';
elseif bet >= 100000
    messageText = 'You could have given a whole community renewable energy with that Money';
else
    messageText = 'Default message'; % Fallback case
end

% Add the dynamic message below the "You Lost!" text
messageDisplay = uicontrol('Style', 'text', ...
    'String', messageText, ...
    'FontSize', 15, ... % Adjusted font size
    'FontWeight', 'normal', ...
    'ForegroundColor', 'black', ...
    'BackgroundColor', get(fig, 'Color'), ...
    'Position', [500, 275, 375, 50]); % [x, y, width, height]

% Add a "Continue" button to close the window
continueButton = uicontrol('Style', 'pushbutton', ...
    'String', 'Continue', ...
    'FontSize', 15, ... % Slightly larger button text
    'Position', [650, 200, 100, 50], ... % Centered button
    'Callback', @(src, event) uiresume(fig)); % Resume execution when clicked

% Pause to display the screen
uiwait(fig);

% Close the figure after the button is clicked
close(fig);
end
```

```
function playerBalance = openingScreen()
% Creates a figure for the Blackjack game where the user selects a starting balance

set(0, 'DefaultFigureWindowStyle', 'docked');
warning('off', 'images:imshow:magnificationMustBeFitForDockedFigure');

% Create a figure window
fig = figure('Name', 'Blackjack', 'NumberTitle', 'off', ...
    'MenuBar', 'none', 'ToolBar', 'none');


% Add a title at the top
titleText = uicontrol('Style', 'text', ...
    'String', 'BLACKJACK', ...
    'FontSize', 50, ...
    'FontWeight', 'bold', ...
    'ForegroundColor', 'red', ...
    'BackgroundColor', get(fig, 'Color'), ...
    'Position', [475, 350, 450, 80]); % Center horizontally

% Add the "You Won!" message at the top
money = uicontrol('Style', 'text', ...
    'String', 'Choose how much money you have You are always going all in!', ...
    'FontSize', 15, ... % Larger font size for a bigger window
    'FontWeight', 'bold', ...
    'ForegroundColor', 'black', ...
    'BackgroundColor', get(fig, 'Color'), ...
    'Position', [500, 275, 375, 50]); % [x, y, width, height]

% Add buttons for the three options
button100 = uicontrol('Style', 'pushbutton', ...
    'String', '$100', ...
    'FontSize', 15, ...
    'Position', [650, 200, 100, 50], ... % Center button
    'Callback', @(src, event) setBalance(100));

button1000 = uicontrol('Style', 'pushbutton', ...
    'String', '$1000', ...
    'FontSize', 15, ...
    'Position', [650, 150, 100, 50], ... % Center button
    'Callback', @(src, event) setBalance(1000));

button100000 = uicontrol('Style', 'pushbutton', ...
    'String', '$100000', ...
    'FontSize', 15, ...
    'Position', [650, 100, 100, 50], ... % Center button
    'Callback', @(src, event) setBalance(100000));

% Variable to store the selected balance
selectedBalance = [];

    function setBalance(amount)
        % Callback to set the balance and close the figure
        selectedBalance = amount;
        uiresume(fig); % Resume execution after button click
    end

% Pause execution until a button is clicked
uiwait(fig);

% Assign the selected balance to the output
playerBalance = selectedBalance;

% Close the figure
close(fig);


end
```

```
function [choice] = playerChoice(blkjSzn)
% this function prompts the user for a choice and decides wether it is
% valid or not

    [r, c, b] = getMouseInput(blkjSzn);

    if b == 1
        choice = 'hit';
    else
        choice = 'stand';
    end
```

```
end
```

```
function tie()
% Creates a screen to display a tie message

% Create a figure window
fig = figure('Name', 'Blackjack - Tie Screen', ...
    'NumberTitle', 'off', ...
    'MenuBar', 'none', ...
    'ToolBar', 'none');

% Add the "It's a Tie!" message at the top
tieText = uicontrol('Style', 'text', ...
    'String', 'It''s a Tie!', ...
    'FontSize', 50, ...
    'FontWeight', 'bold', ...
    'ForegroundColor', 'cyan', ...
    'BackgroundColor', get(fig, 'Color'), ...
    'Position', [475, 350, 450, 80]); % [x, y, width, height]

% Add the "Play again..." message at the bottom
playAgainText = uicontrol('Style', 'text', ...
    'String', 'Play Again!', ...
    'FontSize', 15, ...
    'FontWeight', 'normal', ...
    'ForegroundColor', 'black', ...
    'BackgroundColor', get(fig, 'Color'), ...
    'Position', [500, 275, 400, 50]); % [x, y, width, height]

% Add a "Continue" button to close the window
continueButton = uicontrol('Style', 'pushbutton', ...
    'String', 'Continue', ...
    'FontSize', 10, ...
    'Position', [650, 200, 100, 50], ...
    'Callback', @(src, event) uiresume(fig)); % Resume execution when clicked

% Pause to display the screen
uiwait(fig);

% Close the figure after the button is clicked
close(fig);
end
```

```
function win()
% Creates a larger screen to display a winning message

% Create a larger figure window
fig = figure('Name', 'Blackjack - Win Screen', ...
    'NumberTitle', 'off', ...
    'MenuBar', 'none', ...
    'ToolBar', 'none');

% Add the "You Won!" message at the top
winText = uicontrol('Style', 'text', ...
    'String', 'You Won!', ...
    'FontSize', 50, ... % Larger font size for a bigger window
    'FontWeight', 'bold', ...
    'ForegroundColor', 'green', ...
    'BackgroundColor', get(fig, 'Color'), ...
    'Position', [475, 350, 450, 80]); % [x, y, width, height]

% Add the "Keep playing..." message at the bottom
keepPlayingText = uicontrol('Style', 'text', ...
    'String', 'Keep playing until you lose all your money', ...
    'FontSize', 15, ... % Adjusted font size
    'FontWeight', 'normal', ...
    'ForegroundColor', 'black', ...
    'BackgroundColor', get(fig, 'Color'), ...
    'Position', [500, 275, 400, 50]); % [x, y, width, height]

% Add a "Continue" button to close the window
continueButton = uicontrol('Style', 'pushbutton', ...
    'String', 'Continue', ...
    'FontSize', 10, ... % Slightly larger button text
    'Position', [650, 200, 100, 50], ... % Centered button
    'Callback', @(src, event) uiresume(fig)); % Resume execution when clicked

% Pause to display the screen
uiwait(fig);

% Close the figure after the button is clicked
close(fig);
end
```

```
function [playerBalance, playerHand, realPlayerHand, dealerHand, realDealerHand] =
winCalculator(realPlayerHand, realDealerHand, playerBalance, bet, blkjSzn, empty_sprite, blank_line,
empty_card1, empty_card2, backD, backP, dealerLine, playerLine)
% this function check for current status of player and dealers hand an
% doutputs accordingly the winnings
    playerBalance = playerBalance - bet;
    % calculate winnings and different bet outcomes
    if sum(realPlayerHand) > 21
        loss(playerBalance + bet);
    elseif sum(realDealerHand) > 21
        win();
        playerBalance = playerBalance + 2*bet;
    elseif sum(realPlayerHand) > sum(realDealerHand)
        win();
        playerBalance = playerBalance + 2*bet;
    elseif sum(realPlayerHand) == sum(realDealerHand)
        tie();
        playerBalance = playerBalance + bet;
    else
        loss(bet);
    end
    fprintf('Your current balance is: %0.0f\n', playerBalance);

    % reinitiate som variable for the next round
    playerHand = [];
    realPlayerHand = [];
    dealerHand = [];
    realDealerHand = [];
end
```

Matching Script:

```
clc
clear

% Welcoming and introducing the player to the game.
fprintf('\n Welcome to Our Matlab Memory Game!!')
fprintf('\n Here are some fast facts about Civil Engineering!')
fprintf('\n 1. It is the Oldest Engineering Discipline.')
fprintf('\n 2. Broad Field: It encompasses various sub-disciplines, including structural, environmental,
geotechnical, transportation, and water resources engineering')
fprintf('\n 3. Essential for Society: Civil engineers design, construct, and maintain the infrastructure
we rely on daily, such as roads, bridges, buildings, water supply systems, and more.')
fprintf('\n 4. Famous Structures: Civil engineers have been responsible for iconic structures like the
Eiffel Tower, Hoover Dam, and the Golden Gate Bridge.')
fprintf('\n 5. Sustainable Development: Modern civil engineering emphasizes sustainable development,
focusing on creating eco-friendly and resilient infrastructure.')

% Introduce the game engine with the retro card set
MemoryGame = simpleGameEngine('retro_cards.png', 16, 16, 4, [245, 245, 245]);

% Loop to keep playing until the user decides to quit
playAgain = true;  % Flag for restarting the game
while playAgain
    % Game introduction
    % Draw starting scene
    drawScene(MemoryGame, ones(4, 4));
    % Produce intro text
    intro = text(25, 125, ["Match all the cards in as few attempts as you can", ...
        "Only card value matters, so don't worry about suit", ...
        "Click on the scene to begin playing!"]);
    % Wait until user clicks to start game and get rid of intro text
    getMouseInput(MemoryGame);
    intro.Visible = 'off';

    % %%%%GAME LOOP%%%%%
    % Declaration of what the back of the card looks like. This is done by
    % randomly selecting an index from 3-10
    cardBack = randi(7) + 3;

    % Declaration of the play grid. This array will be manipulated to display
    % various things. It'll start off blank and the elements of it can be
    % replaced with various cards
    displayArray = ones(4, 4) * cardBack;

    % Randomly declare the card values underneath the cardBacks.
    % This is done by creating a random assortment of indexes from 21 - 72.
    numPairs = 8; % Number of unique card pairs
    uniqueCards = randperm(numPairs) + 20; % Generate 8 unique card values
    cardValues = [uniqueCards, uniqueCards]; % Duplicate for pairs
    cardValues = cardValues(randperm(16)); % Shuffle the 16 cards
    cardArray = reshape(cardValues, [4, 4]); % Arrange into a 4x4 grid

    % Declare variable for how many matches the user has made
    numMatches = 0;

    % Loop until every card is removed
    while any(find(displayArray > 1, 1))
        % Display current scene state
        drawScene(MemoryGame, displayArray);
        % Flip a card at mouse position
        displayArray = flipCard(displayArray, cardArray, MemoryGame);
        % Display current scene state
        drawScene(MemoryGame, displayArray);
        % Flip another card
        displayArray = flipCard(displayArray, cardArray, MemoryGame);
        % Display current scene state then wait a second
        drawScene(MemoryGame, displayArray);
        pause(1);
        % Increment number of matches
        numMatches = numMatches + 1;
        % Update array to reflect if the user successfully matched anything
        displayArray = confirmMatches(displayArray, cardBack);
        % Display end result to user before looping again
        drawScene(MemoryGame, displayArray);
    end

    % Game over - all cards matched
    fprintf('\nCongratulations! You matched all the cards in %d attempts!', numMatches);

    % Ask the user if they want to play again
    playAgainInput = input('\nDo you want to play again? (y/n): ', 's');
    if lower(playAgainInput) ~= 'y'
        playAgain = false; % Exit the main loop
        fprintf('\nThanks for playing! Goodbye!\n');
    end
end
%clear figure
clear all
```

Matching Functions:

```
function [testedArray] = confirmMatches(targetArray, cardBack)
%%%The purpose of this function is to test if the two selected cards are the
%%same value. If they aren't they the cards flip back over otherwise, they
%disappear.
%Convert all face up cards to hearts
convertedArray = convertCardsToValue(targetArray);
%Get size of array
[rows, cols]=size(convertedArray);
amount = rows * cols;
%Loop through each index of the Arrau
for i = 1: amount
 %Only test a card if it's face up
 if(convertedArray(i)>20)
 %Loop through every card after the card being tested
 for j = i+1:amount
 %%If the loop finds a card of the same value as the
%testee then make both cards blank
if (convertedArray(i) == convertedArray(j))
 convertedArray(i) = 1;
 convertedArray(j) = 1;
 targetArray(i) = 1;
 targetArray(j) = 1;
 end
 end
 end
end
%Loop through targetArray
for i = 1:amount
 %If card is still face up then flip it back over
 if(targetArray(i) >= 21)
 targetArray(i) = cardBack;
 end
end
%Return the testedArray
testedArray= targetArray;
end
```

```
function convertedArray = convertCardsToValue(targetArray)
%Change the value of every face up card to it's hearts equivalent
```

```
%Get size of array
[rows,cols]=size(targetArray);
%Loop through every index
for i=1:rows
 for j=1:cols
 %%If a number is greater than 33 subtract 13 from it until it's
 %below 33. This converts all the cards into hearts
 while(targetArray(i,j)>33)
 targetArray(i,j) = targetArray(i,j) - 13;
 end
 end
end
%Return the new array
convertedArray = targetArray;
end
```

```
function [flippedArray] = flipCard(targetArray, cardArray,gameEngine)
%Flip a value in a array from the cardback value to it's true value
%Get position of mouse input until it selects a card which is a card back
[row,col] = getMouseInput(gameEngine);
while targetArray(row,col) == 1 || targetArray(row,col) > 20
[row,col] = getMouseInput(gameEngine);
end
%Change card value to it's true value
targetArray(row,col)=cardArray(row,col);
%Return changed array
flippedArray = targetArray;
end
```

Connect 4 Script:

```
clc
clear
% Initialize scene
my_scene = simpleGameEngine('ConnectFour (1).png',86,101);

set(0,'DefaultFigureWindowStyle','docked');
warning('off','images:magnificationMustBeFitForDockedFigure');

% Set up variables to name the various sprites
empty_sprite = 1;
red_sprite = 2;
black_sprite = 3;
conue="n";
%this will make a bord that is easyer to work with
bord=[1:7;8:14;15:21;22:28;29:35;36:42];
bord(:)=1;
%these will help with the loop
noWiner=1;
i=1;

% Display empty board
board_display = empty_sprite * ones(6,7);
drawScene(my_scene,board_display)
%this should play the game
fprintf("How to play connect four")
fprintf("\n Two players are required")
fprintf("\n You take turns choosing the column that you would like to place your peace in (1-7)")
fprintf("\n The piece will appear in the bottom row in less that row is occupied it will then keep looking
")
fprintf("\n for the first unoccupied row and occupy that")
fprintf("\n When all rows are occupied you cant place a piece there")
fprintf("\n The goal of the game is to place for pieces either horizontally")
fprintf("\n or vertically or diagonally without being blocked by your opponent")
fprintf("\n \n \n The fastest plan to ever fly was the X15 at Mach 6.7")
fprintf("\n When metals heat up they expand this is important ")
fprintf("\n to acknowledge when making things that will experience extreme temperature fluctuations")
fprintf("\n When designing things it is important to remember what you are trying to make ")
fprintf("\n and not how cool it is to do something.\n")
while noWiner==1
    %this takes key bord imputs and makes moves

    move=input("make you moves");
    %this swiches the players
    if rem(i,2)==0
        player=red_sprite;
    else
        player=black_sprite;
    end
    %this is how to interact with the bord
   bord= gameBord(move,player,bord);
    %this desplays the bord
   board_display=bord;
   drawScene(my_scene,board_display)
   %bord
   %this says who wins
   noWiner= Win(bord,player);
   if noWiner==2
       fprintf("Player 1 wins!!!")
       conue=input("\n Do you want play agen (Y/N): ","s");
   elseif noWiner==3
       fprintf("Player 2 wins!!!")
       conue=input("\n Do you want play agen (Y/N): ","s");
   end
   if conue=="y"||conue=="Y"
       noWiner=1;
```

```
        i=0;
        bord=[1:7;8:14;15:21;22:28;29:35;36:42];
        bord(:)=1;
        conue="n";
    end
    i=i+1;
end

clc
close all
```

Connect 4 Functions:

```
function [WIN] = dieTest(bord,player)
%UNTITLED5 Summary of this function goes here
%   Detailed explanation goes here
win=1;
for i=3:6
    for j=1:4

        if (bord(i,j)==player) && (bord(i-1,j+1)==player)&& ((bord(i-2,j+2)==player) && (bord(i-
3,j+3)==player))
            win=player;
            break
        end
    end
    if win==player
        break
    end
end
for i=1:3
    for j=1:4

        if (bord(i,j)==player) && (bord(i+1,j+1)==player)&& ((bord(i+2,j+2)==player) &&
(bord(i+3,j+3)==player))
            win=player;
            break
        end
    end
    if win==player
        break
    end
end
WIN=win;
end
```

```
function [affbord] = gameBord(move,player,bord)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
good=1;

while good==1
    if move <=7 && move>=1
        if bord(6,move)==1
            bord(6,move)=player;
            good=2;
        elseif bord(5,move)==1
            bord(5,move)=player;
             good=2;
        elseif bord(4,move)==1
            bord(4,move)=player;
             good=2;
        elseif bord(3,move)==1
            bord(3,move)=player;
```

```
             good=2;
        elseif bord(2,move)==1
            bord(2,move)=player;
             good=2;
        elseif bord(1,move)==1
            bord(1,move)=player;
             good=2;
        else
            fprintf("Not a valid move")
            move=input("\n make you moves");
        end
    else
        fprintf("Not a valid move")
        move=input("\n make you moves");
    end
affbord = bord;
end
```

```
function [WIN] = horzTest(bord,player)
%UNTITLED5 Summary of this function goes here
%   Detailed explanation goes here
win=1;
for i=1:6
    for j=1:4

        if (bord(i,j)==player) && (bord(i,j+1)==player)&& ((bord(i,j+2)==player) && (bord(i,j+3)==player))
            win=player;
            break
        end
    end
end
WIN=win;
end
```

```
function importfile(fileToRead1)
%IMPORTFILE(FILETOREAD1)
%  Imports data from the specified file
%  FILETOREAD1:  file to read

%  Auto-generated by MATLAB on 10-Nov-2024 20:39:44

% Import the file
newData1 = importdata(fileToRead1);

% Create new variables in the base workspace from those fields.
vars = fieldnames(newData1);
for i = 1:length(vars)
    assignin('base', vars{i}, newData1.(vars{i}));
end
```

```
function [WIN] = verTest(bord,player)
%UNTITLED5 Summary of this function goes here
%   Detailed explanation goes here
win=1;
for i=1:3
    for j=1:7

        if (bord(i,j)==player) && (bord(i+1,j)==player)&& ((bord(i+2,j)==player) && (bord(i+3,j)==player))
            win=player;
            break
        end
    end
    if win==player
```

```
        break
    end
end
WIN=win;
end
```

```
function [Winner] = Win(bord,player)
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here
Winner=horzTest(bord,player);
if Winner==1
    Winner=verTest(bord,player);
end
if Winner==1
    Winner=dieTest(bord,player);
end
```

Brief Discussion

Over Under 7:

Over Under 7 had many difficulties at the outset. The learning curve to understand SimpleGameEngine was quite large for me and it took a lot of effort to become proficient in it.

I really started with the graphics preview and progressed from there. My vision for the game changed drastically as I realized what was and was not possible within SimpleGameEngine. Initially, I wanted a single-screen interactive window, but I soon realized I did not have the technical know-how to achieve that goal, so I instead opted for a scene-based game. With this thought in mind, I was able to sketch out a new plan and execute it, using different scenes for each game state.

Once I had learned the basics of SimipleGameEngine, I was able to construct a partially complete game utilizing inputs between scenes for the graphics preview. After that, I created custom graphics, learned how to dock figures automatically, and add music, which gave the game an overall more polished Lastly, I added the logic behind the game: prediction outcomes, play again loop, educational facts, etc. From there, I kept making minor tweaks to improve usability and visual appeal.

When faced with obstacles, I consulted teammates, instructors, the MATLAB “help” function or the internet. In most cases, a second pair of eyes was all I needed to spot my coding mistakes, which were sometimes as simple as case errors, but sometimes logical mistakes.

Blackjack:

The method in which this program was constructed was to first make the program work logically in the command window and then apply the simple game engine to give graphics for the game. The obstacles faced with this approach was just changing the logic so that it would work with the graphics. For example, to be able to calculate the values of the hands the original logic for the command window was straight forward but then with simple game engine it needed a different approach to be able to account for the sprites. The program was developed by trial-and-error, meaning that a lot of testing was done so that it was able to run successfully.

Matching:

During the game’s testing, we would run our code and take note of any issues we ran into. For example, we originally utilized a 52 deck of cards that were utilized in a 4 x 13 display. As we were testing our game, we found that the game’s difficulty was harder than intended and as a result took a lot of time. To fix this, we used a 4 x 4 array of cards to decrease the difficulty and time required to play our game. Our MATLAB code progressed greatly as we continued to improve our game. A specific example being our input. When we first created our graphics preview, the game was not running and relied on input from the space bar to flip over cards. As we progressed, we were able to utilize mouse input to allow the user to flip and select specific cards that they chose. Previously mentioned, an obstacle we faced was the lowering the difficulty and duration of the game. We struggled mainly on trying to cut down the number of cards but allowing the cards to still match. We overcame this specific issue by using “numpairs = 8” to ensure there were 8 pairs of cards that matched 1:1. We also fixed our issue by changing the display array to 4,4 instead of 4,13.

Connect 4:

Originally the plan was to try and make an AI so that you could play against connect 4 when you were alone. However realizing the time constraints for this project, I was not able to fulfill that idea. Mostly everything went to plan, but originally when I was trying to make connect 4, I was not using enough loops or functions and had written way more code than was needed, making it confusing. I was able to cut down my code by a couple hundred lines when I realized that I could use a loop in a certain location. Also, I ended up using a lot more variables than I started out with again, that cut down my code by hundreds of lines. The biggest problem I had was testing for if the solution had been reached. It kept making it so that the game broke. This was because I was trying to test each spot in the board, and this made it so that it would test spots that were not in the board. The solution I found to this is kind of interesting. Instead of doing a for loop like this for i=1:3 I did it like this for i=3:6. This made it so that it was checking only the locations that were on the board but also checking enough of the location.

5. Conclusion and Recommendation

Over Under 7:

Creating Over Under 7 proved to be effective in developing MATLAB programming skills. Additionally, better knowledge of the game itself was gained, progressing from zero prior knowledge of how to play to an intimate familiarity with its rules and gameplay. As for MATLAB knowledge, the SDP firmed up the concepts behind loops, logical structures, plot creation, indices, matrices, etc.

It proved to be an enjoyable process to create custom graphics and messages to be displayed throughout the game. It was also enjoyable to think through the logic behind all the loops and if-else-if statements to create a cohesive program. Debugging proved to be relatively easy since each portion of the game was tackled and tested sequentially, making it simiple to zone in on the errors.

Moving forward in development would most likely involve and overhaul of the game. After creating the game select screen for the team's games, the viability of adding buttons became evident. Therefore, buttons would be implemented to create more streamlined visuals for the user. Additionally, as development progressed, the use of functions would be considered more prominently. In the current program, functions were an afterthought, but they could be better used if they were planned to be an integral part of the program from the start.

Blackjack:

The result obtained were a blackjack game that was simplified to be able to make a good program. The recommendations would be to be able to allow the user to choose exactly how much to bet.

Matching:

Throughout the making of this game, we faced many challenges. But, with the help of the rest of our group we go through the finish line. Going into the creation of this game, we sort of took it easy, especially in the beginning. This quickly started to catch up to us as we began the actual programming stage. Our initial preview was completely different from our final product, and if the team could go back, we would have tried a little harder to make the process smoother. One of our main issues was to create a mouse input instead of a keyboard. This fix was solved by using the simple game engine. One of our biggest problems was adjusting the array of cards. Initially we adjusted the number of cards, but in return the cards did not all have matches. This was our most time-consuming part of the game. Thankfully we had help from the other members of our team. In the end, working through difficult tasks and problems with the code made it all worth it. Seeing our teams' games come together as one is an awesome experience to have.

Connect 4:

In conclusion, I think my team did a pretty good job however I do think that commentating and making sure the code looks pretty is more important than I ever thought it was before. If you are trying to edit someone else’s code, it’s very important to be able to read it so it is important to make sure the comments make sense and the code is legible. Given more time, I would have tried to make it so that you could play against the computer. I was thinking about trying to implement minimax algorithm. However, I ran out of time and, I do not know how to do that. I also feel that having smaller goals that were reached more consistently would have been helpful and to have talked more with the group and discussed progress would have been an important step that my group unfortunately he was not able to do as much. In total I think communication is one of the pillars to group work and something that we need to work on more.

References

"SoftwareDesignProject_Procedure.pdf" Available at: https://carmen.osu.edu/ [Accessed Nov. 2024]

"SDP Lab 1_Presentation.pdf" Available at: https://carmen.osu.edu/ [Accessed Nov. 2024]

"SDP Resources" Available at: https://carmen.osu.edu/ [Accessed Nov. 2024]

Gilat, A., *MATLAB: An Introduction with Applications 5th Edition*. Hoboken, NJ: Wiley, 2015.

U.S. Department of the Treasury, "The debt to the penny and who holds it," [Online]. Available: https://fiscaldata.treasury.gov. [Accessed Nov. 18, 2024].

Federal Reserve Bank of New York, "Household debt and credit report," 2024. [Online]. Available: https://www.newyorkfed.org/microeconomics/hhdc.html. [Accessed Nov. 18, 2024].