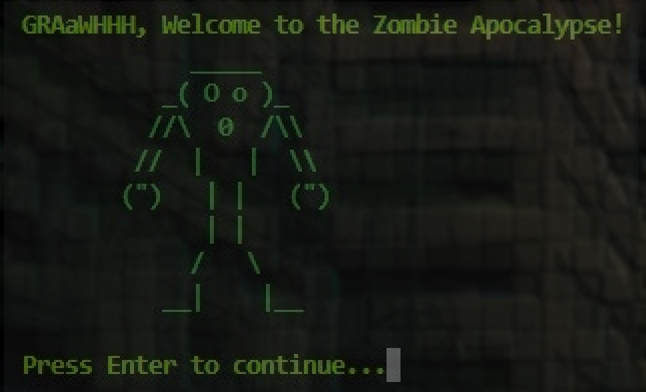
ZAPP – ZOMBIE APOCALYPSE PLANNING POKER



```
GRAaWHHH, Welcome to the Zombie Apocalypse!

            _____
          _( O o )_
         //\  0  /\\
        //  |   |  \\
       (")   | |   (")
             | |
            /   \
          __|    |__

Press Enter to continue...
```

SLIDE DECK

T1A3 – TERMINAL APP

by Jim Lister

Project overview

To Watch the video of this presentation please follow this link;

https://www.youtube.com/watch?v=WAXMrl4ehB4

The project:

I wanted to create a terminal app that would feature a game that would be familiar to those in the programming world. That way I would have the frame-work for a program that would potentially be useful in a programming or development environment. I chose "planning poker" or "scrum poker" as it fulfilled this requirement. To make it more interesting, I based it around the zombie apocalypse.

Project duration:

2 weeks - (Dec 5 - Dec 18)

Project overview

The purpose:

The purpose of building this terminal app is to gain a better understanding of the inner workings and processes of operating systems and hardware devices. Terminal applications can be a standard in information technology practices rather than a graphical user interface therefore developing one will increase technical sophistication in not just back-end coding but ability to use other terminal apps.

The goal:

The main goal is to become coherant in DRY python programming language and increase apptitude in all manner of applying python fundamentals into a real working program. The program must accept user input and run printed outputs. It must utlise variables and correctly assign them values. It must use conditional control flow structures and loops. It must import modules including external packages and it must include read and write file handling as well as error handling.

Project overview

My role:

Python Developer

Responsibilities:

- □ Planning & Implementation
- □ Test Driven Development
- □ Coding
- □ Testing & Debugging
- □ Error Handling
- □ Deployment

Planning and Implementation

- ❑ Ideation
- ❑ Definition of Application Features
- ❑ Implementation Plan
- ❑ Breakdown of Implementation Plan
- ❑ Execution of Implementation Plan

Ideation & Definition of Application's Features

Ideation:

As mentioned in the overview I wanted to create something that might be recognisable in the programming and development world. "Planning" or "scum" poker fit the bill and the challenges to be set in the Zombie Apocalypse just for fun.

Definition of Application Features.

List of program features;

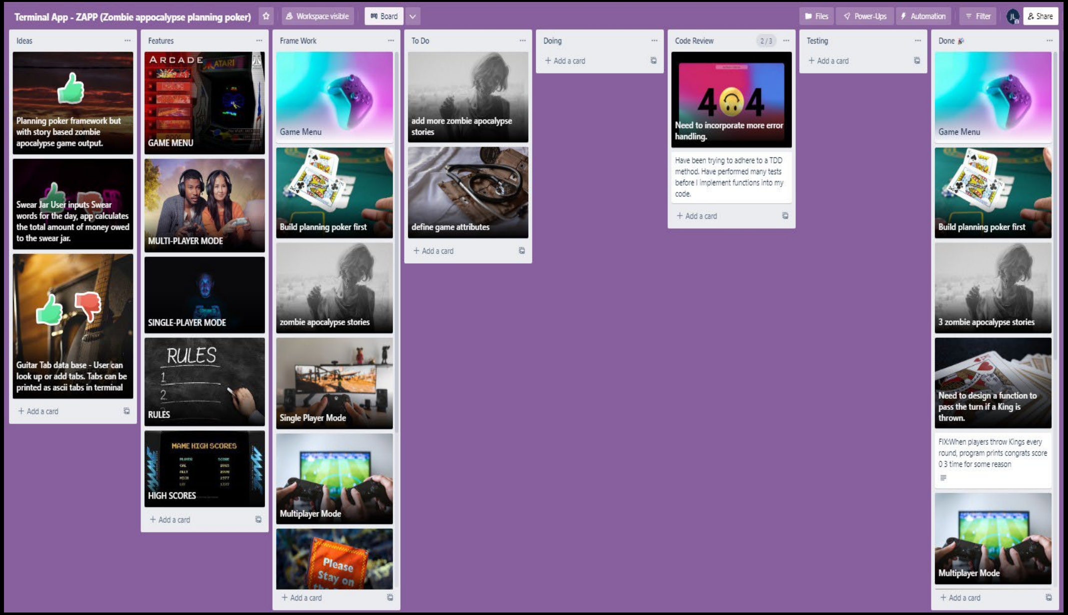
- ❑ Game Intro
- ❑ Game Menu
- ❑ Multi Player Mode
- ❑ Single Player Mode
- ❑ High Score
- ❑ Bonus Content

Implementation Plan

I created my implementation plan on Trello's project management platform. You can visit my Trello board by following the link;

https://trello.com/b/1Fb6TSx0/terminal-app-zapp-zombie-appocalypse-planning-poker

The main considerations within my plan were to attempt a TDD approach, design the planning poker game first as it would lay out the initial frame-work for the rest of the game and to initialize classes for the player and challenge objects to essentially make the number of players and number of challenges scalable.



Terminal App - ZAPP (Zombie appocalypse planning poker)

Ideas
- Planning poker framework but with story based zombie apocalypse game output.
- Swear Jar User inputs Swear words for the day, app calculates the total amount of money owed to the swear jar.
- Guitar Tab data base - User can look up or add tabs. Tabs can be printed as ascii tabs in terminal

Features
- GAME MENU
- MULTI-PLAYER MODE
- SINGLE-PLAYER MODE
- RULES
- HIGH SCORES

Frame Work
- Game Menu
- Build planning poker first
- zombie apocalypse stories
- Single Player Mode
- Multiplayer Mode

To Do
- add more zombie apocalypse stories
- define game attributes

Doing

Code Review 2/3
- Need to incorporate more error handling.
- Have been trying to adhere to a TDD method. Have performed many tests before I implement functions into my code.

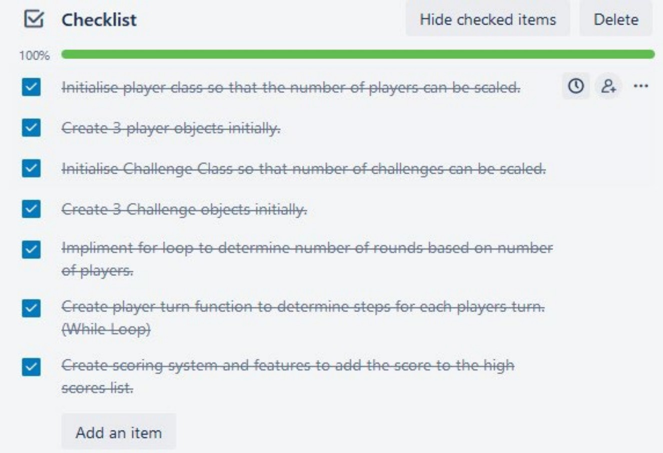
Testing

Done
- Game Menu
- Build planning poker first
- 3 zombie apocalypse stories
- Need to design a function to pass the turn if a King is thrown.
- FIX:When players throw Kings every round, program prints congrats score 0 3 time for some reason
- Multiplayer Mode

Break Down of Implimentation Plan

Feature 1. Multi-Player Mode;

- ❑ Develop First to determine overall frame-work for the games.
- ❑ Initialise Player Class so that number of players can be scaled.
- ❑ Define only 3 player objects initially in order to keep development small scale.
- ❑ Initialise Challenges Class so that number of challenges can be scaled.
- ❑ Define only 3 challenge objects initially in order to keep development scale down.
- ❑ Implement for loop in multiplayer function to determine number of rounds based on number of players.
- ❑ Design player turn function to determine steps for each player turn.(While loop)
- ❑ Create scoring system and feature to add the scores to the high scores list.



☑ **Checklist**

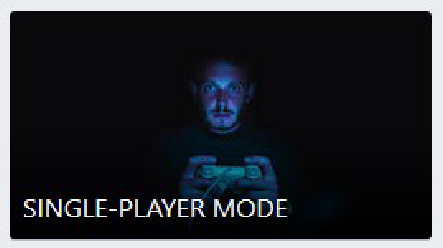
100%

- ☑ ~~Initialise player class so that the number of players can be scaled.~~
- ☑ ~~Create 3 player objects initially.~~
- ☑ ~~Initialise Challenge Class so that number of challenges can be scaled.~~
- ☑ ~~Create 3 Challenge objects initially.~~
- ☑ ~~Impliment for loop to determine number of rounds based on number of players.~~
- ☑ ~~Create player turn function to determine steps for each players turn. (While Loop)~~
- ☑ ~~Create scoring system and features to add the score to the high scores list.~~

T1A3 – TERMINAL APP

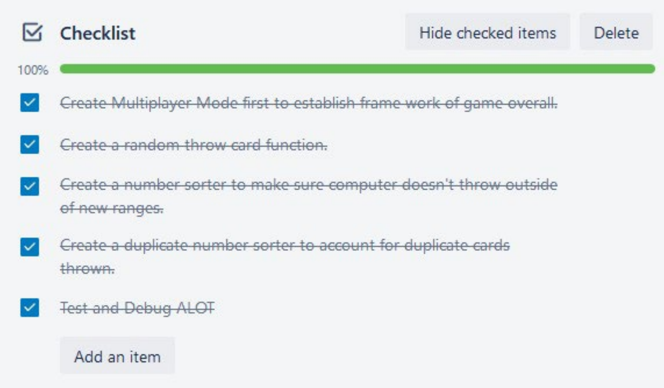
by Jim Lister

Break Down of Implimentation Plan Cont.



Feature 2. Single-Player Mode;

- □ Use initial frame-work from Multi-Player Mode.
- □ Develop a random throw card function.
- □ Program a number sorter to make sure computer doesn't throw outside of new ranges.
- □ Program a duplicate number sorter to account for duplicate cards thrown.
- □ Test and Debug ALOT

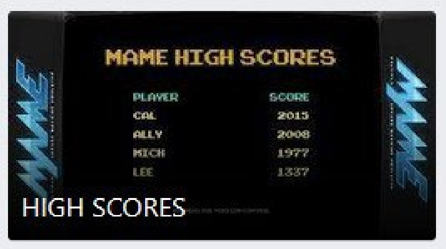


☑ Checklist

100%

- ☑ ~~Create Multiplayer Mode first to establish frame work of game overall.~~
- ☑ ~~Create a random throw card function.~~
- ☑ ~~Create a number sorter to make sure computer doesn't throw outside of new ranges.~~
- ☑ ~~Create a duplicate number sorter to account for duplicate cards thrown.~~
- ☑ ~~Test and Debug ALOT~~

Break Down of Implimentation Plan Cont.



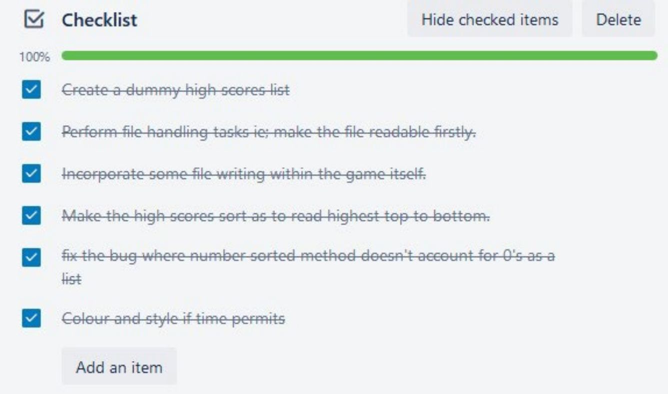
MAME HIGH SCORES

PLAYER	SCORE
CAL	2015
ALLY	2008
MICH	1977
LEE	1337

HIGH SCORES

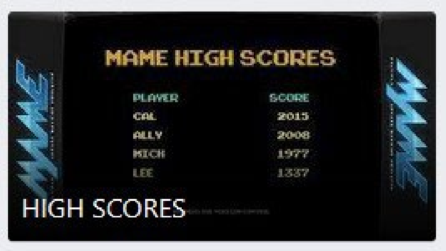
Feature 3. High Scores;

- □ Author a dummy high scores list in a text file.
- □ Perform file handling tasks ie; make the file readable.
- □ Incorporate some file writing within the game itself.
- □ Make the high scores sort as to read highest top to bottom.
- □ Fix the bug where numbers sorted method doesn't account for zeros as a list in lines.



- ☑ ~~Create a dummy high scores list~~
- ☑ ~~Perform file handling tasks ie; make the file readable firstly.~~
- ☑ ~~Incorporate some file writing within the game itself.~~
- ☑ ~~Make the high scores sort as to read highest top to bottom.~~
- ☑ ~~fix the bug where number sorted method doesn't account for 0's as a list~~
- ☑ ~~Colour and style if time permits~~

Break Down of Implimentation Plan Cont.



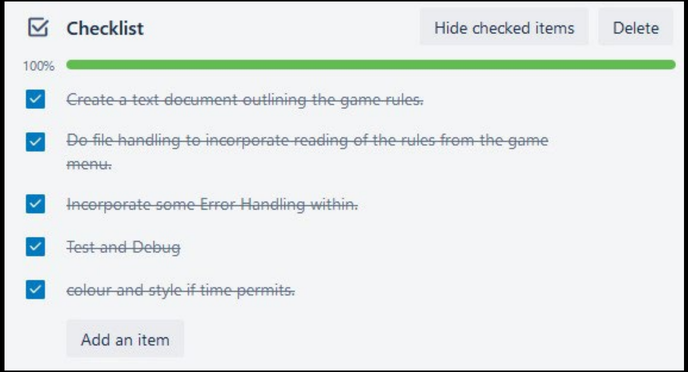
MAME HIGH SCORES

PLAYER	SCORE
CAL	2015
ALLY	2008
MICH	1977
LEE	1337

HIGH SCORES

Feature 4. Rules;

- ❑ Author a text document outlining the game rules.
- ❑ Perform file handling to incorporate reading of the rules from the game menu.
- ❑ Incorporate some Error Handling within.



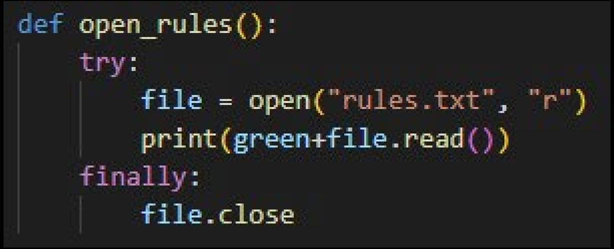
☑ ~~Create a text document outlining the game rules.~~

☑ ~~Do file handling to incorporate reading of the rules from the game menu.~~

☑ ~~Incorporate some Error Handling within.~~

☑ ~~Test and Debug~~

☑ ~~colour and style if time permits.~~



```
def open_rules():
    try:
        file = open("rules.txt", "r")
        print(green+file.read())
    finally:
        file.close
```

Break Down of Implimentation Plan Cont.

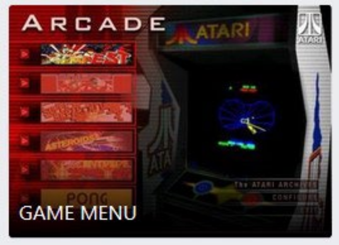
Feature 5. Bonus Content;

I designed my program to basically account for any user input either being accepted or returning invalid selection statements, so it's impossible for the user to throw an error and break the program. I added a bonus feature to demonstrate my understanding of Error Handling using the try/except statements.

- ❑ Design a function that deliberately throws a ValueError. User input asks for a name, but in-fact requires an integer.

- ❑ Instead of breaking the program except statement executes a read file function that displays a secret message from a text file. So, it incorporates a little bit more file handling as well so it's 2 birds with one stone.

Break Down of Implimentation Plan Cont.



Feature 6. Game Menu;

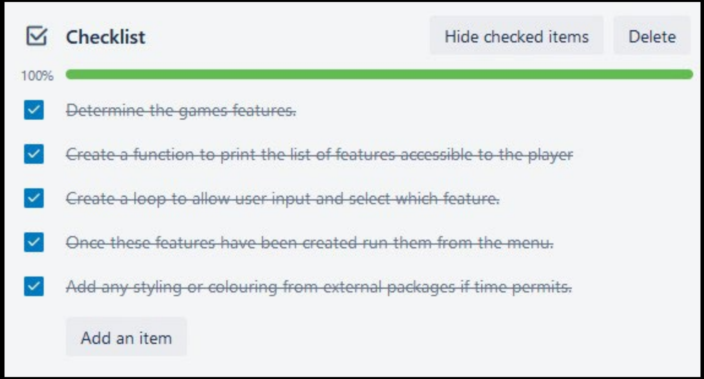
❑ Determine the games features.

❑ Create a function to print the list of features accessible to the player.

❑ Design a loop to allow user input and select which feature to execute.

❑ Once the features have been created run them from the menu.

❑ Add any styling or colouring from external packages if time permits.



☑ ~~Determine the games features.~~

☑ ~~Create a function to print the list of features accessible to the player~~

☑ ~~Create a loop to allow user input and select which feature.~~

☑ ~~Once these features have been created run them from the menu.~~

☑ ~~Add any styling or colouring from external packages if time permits.~~

Feature 7. Game Intro;

❑ Illustrate some ascii art to be utilised in introductory sequence if time permits.

Execution of Implementation Plan

As I expected, the execution of the plan did evolve somewhat over the course of the project but overall was extremely helpful in order to organize my processes so that I could complete the tasks within the given time frame.

Once I got the hang of Trello it became an instrumental part of the planning process and a crucial method to track my progress. Learning about Trello was an unexpected benefit about the project overall and I hope to use this project management feature in the future.

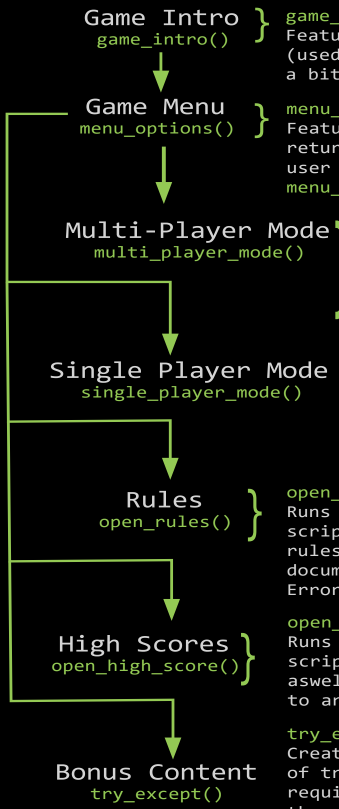
Overview of Terminal Application (R9)

- ❑ List of Features
- ❑ Overall Structure
- ❑ Logic & Control Flow
- ❑ Demonstration of Terminal App

List of Features

- ❑ 1. Game Intro
- ❑ 2. Game Menu
- ❑ 3. Multi-player Mode
- ❑ 4. Single Player Mode
- ❑ 5. Rules
- ❑ 6. High Scores
- ❑ 7. Bonus Content

Overall Structure



game_intro()
Features some styling from external package “colored” (used throughout project) and an ASCII image just for a bit of fun.

menu_options()
Features an ART function from external package “art” that returns some ASCII text. Runs onwhile loop that waits for user input based off the prompts defined in the menu_options() function.

multi_player_mode()
Runs in a nested for loop to determine round based on which_challenges() and number_of_players() functions and calls player_turn(). Ultimately runs conditional statements to check if thrown cards match or do not match. [list_of_inputs]

player_turn()
Runs in a while loop and waits for user input which is throw a card from the range.

number_of_players()
function that encapsulates list of class objects.
player1.name = “Survivor 1”

which_challenge()
score()
value()
function that encapsulates list of class objects.
challenge1.challenge
challenge1.score
challenge1.value for example

single_player_mode()
Runs in a nested for loop to determine round based on which_challenge() and number_of_players() functions and calls player_turn() & random_player_turn(). Ultimately runs conditional statements to check if thrown cards match or do not match. [list_of_inputs]

random_player_turn()
Runs in a while loop and which this time chooses a card at random generated by throw_card()function.

□ throw_card() also runs a function called run_next() which checks the thrown cards for duplicates, sorts them, determines the outside range and removes the cards outside of the range for the next round. The functions withinrun_next() are;

□ check_for_duplicates()
□ card_sorter()
□ duplicate_card_sorter()

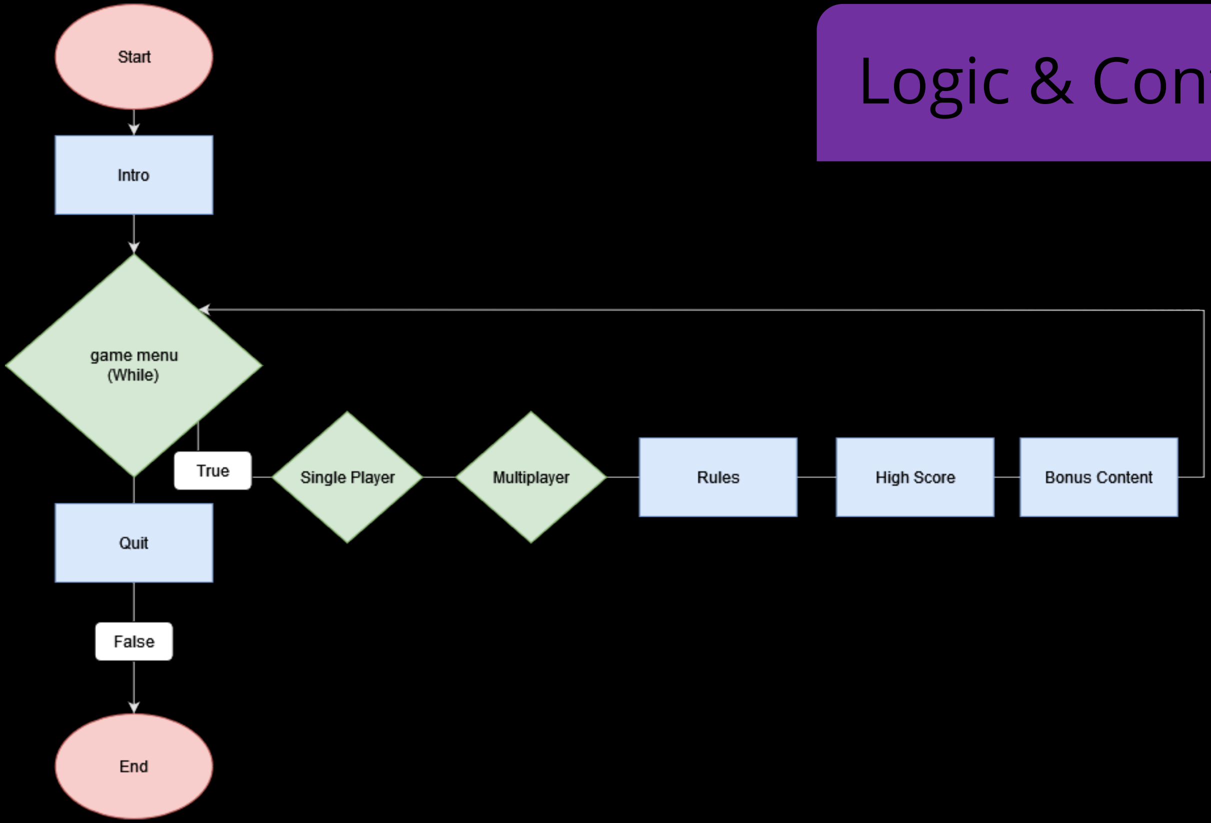
open_rules()
Runs some file handling script to open the game rules in an external text document. Includes some Error Handling.

□ enter_high_score() is the final function run in either mode and it performs some file handling reading and writing. As well as conditional statements to make sure the high score sorts from top to bottom.

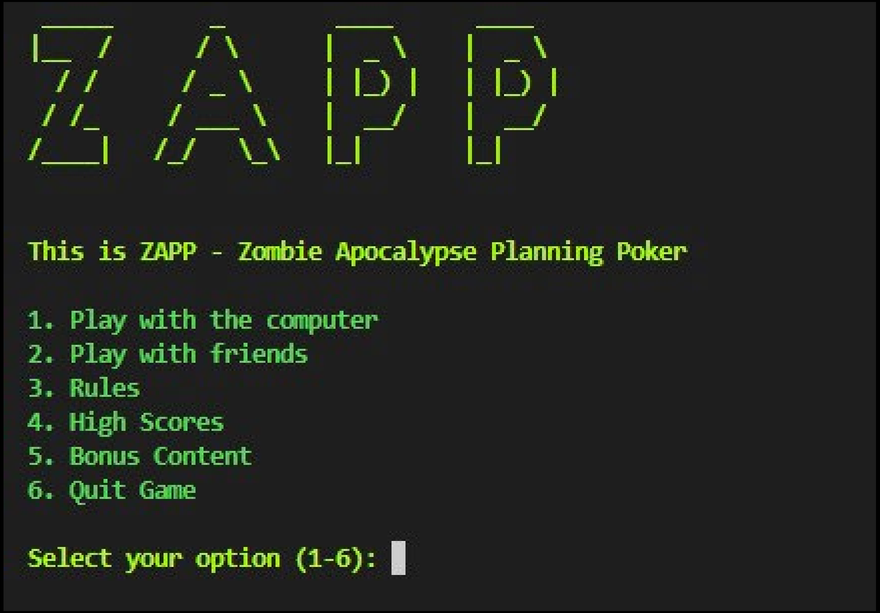
open_high_score()
Runs some file handling script to open the score aswell as a sorted function to arange them hieghest to lowest.

try_except()
Created to demonstrate understanding of try/except error handling. The input requires a number but if you enter a string the program handles the error with a hidden feature. Also includes some file handling.

Logic & Control Flow



Demonstration of Terminal App



```
 _____     _     ____   ____
|__  /    / \   |  _ \ |  _ \
  / /    / _ \  | |_) || |_) |
 / /_   / ___ \ |  __/ |  __/
/____| /_/   \_\|_|    |_|


This is ZAPP - Zombie Apocalypse Planning Poker

1. Play with the computer
2. Play with friends
3. Rules
4. High Scores
5. Bonus Content
6. Quit Game

Select your option (1-6):
```

*If you’re not watching the whole presentation as a video, watch the terminal app demo by following the link below;

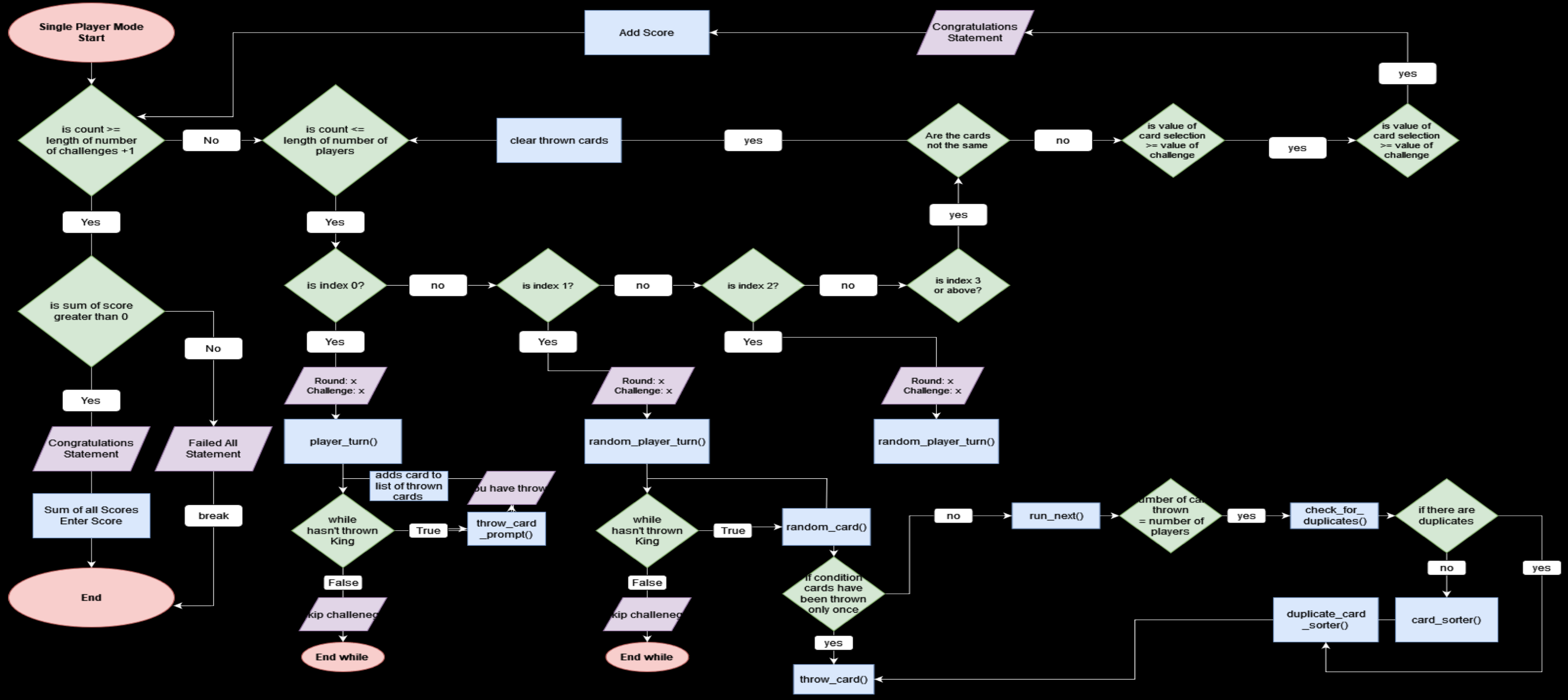
https://www.youtube.com/watch?v=COIVYZz_xPA

Overview of Code (R10)

- ❑ Code Logic & Control Flow
- ❑ Review of development/build process
- ❑ Testing & TDD process
- ❑ Challenges

Code Logic and Control Flow

Single Player Mode



Code Logic and Control Flow Cont.

Player Class

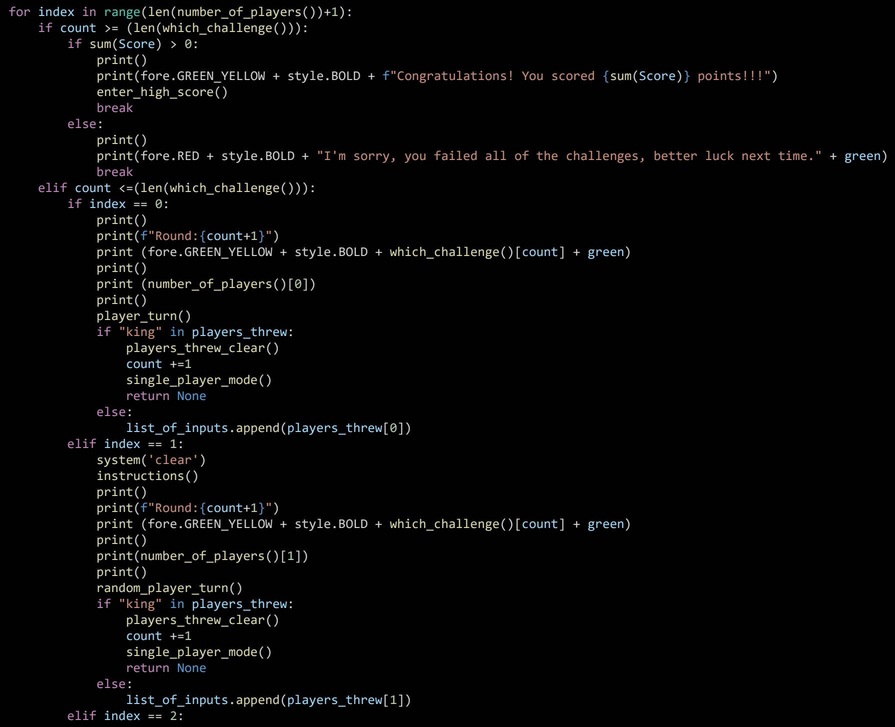
```
class Player:
    def __init__(self, name):
        self.name = name

from player_class import Player

def number_of_players():
    player1 = Player("Survivor 1")
    player2 = Player("Survivor 2")
    player3 = Player("Survivor 3")

    player_list = [player1.name, player2.name, player3.name]

    return (player_list)
```

Challenges Class

```
class challenges:
    def __init__(self, challenge, value, is_complete, score):
        self.challenge = challenge
        self.value = value
        self.is_complete = is_complete
        self.score = score


from challenges import challenges

challenge1 = challenges("There's a survivor surrounded by a horde at the downtown mall.",
150, False, 1000 )
challenge2 = challenges("There's an overturned supply truck on highway 99.", 50, False,
1000)
challenge3 = challenges("There's military personnel shooting innocent survivors in the
city central hospital.", 400, False, 5000)
def which_challenge():
    challenge_list = [challenge1.challenge, challenge2.challenge, challenge3.challenge]
    return (challenge_list)

def score():
    score_list = [challenge1.score, challenge2.score, challenge3.score]
    return (score_list)

def value():
    value_list = [challenge1.value, challenge2.value, challenge3.value]
    return (value_list)
```

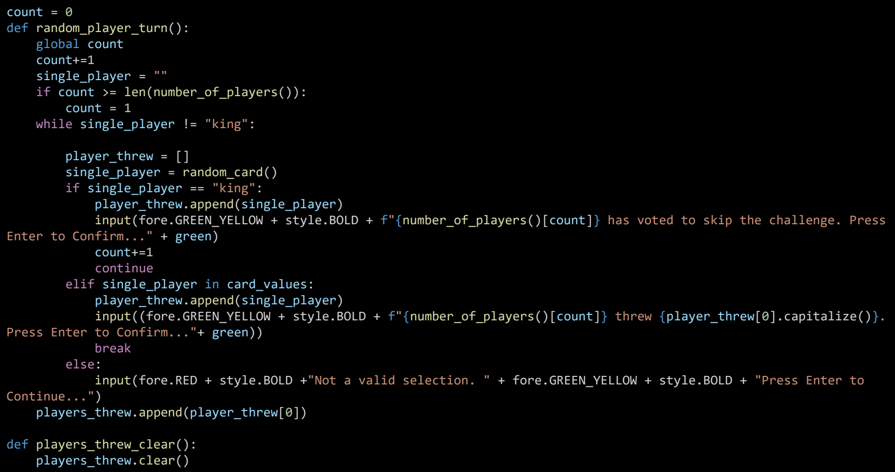
Code Logic and Control Flow Cont.

single_player_mode() for loop



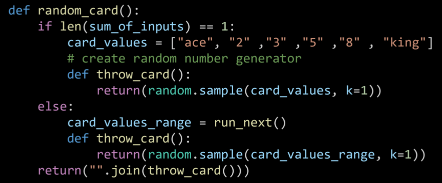
```python
for index in range(len(number_of_players())+1):
    if count >= (len(which_challenge())):
        if sum(Score) > 0:
            print()
            print(fore.GREEN_YELLOW + style.BOLD + f"Congratulations! You scored {sum(Score)} points!!!")
            enter_high_score()
            break
        else:
            print()
            print(fore.RED + style.BOLD + "I'm sorry, you failed all of the challenges, better luck next time." + green)
            break
    elif count <=(len(which_challenge())):
        if index == 0:
            print()
            print(f"Round:{count+1}")
            print (fore.GREEN_YELLOW + style.BOLD + which_challenge()[count] + green)
            print()
            print (number_of_players()[0])
            print()
            player_turn()
            if "king" in players_threw:
                players_threw_clear()
                count +=1
                single_player_mode()
                return None
            else:
                list_of_inputs.append(players_threw[0])
        elif index == 1:
            system('clear')
            instructions()
            print()
            print(f"Round:{count+1}")
            print (fore.GREEN_YELLOW + style.BOLD + which_challenge()[count] + green)
            print()
            print(number_of_players()[1])
            print()
            random_player_turn()
            if "king" in players_threw:
                players_threw_clear()
                count +=1
                single_player_mode()
                return None
            else:
                list_of_inputs.append(players_threw[1])
        elif index == 2:
```

random_player_turn() while loop



```python
count = 0
def random_player_turn():
    global count
    count+=1
    single_player = ""
    if count >= len(number_of_players()):
        count = 1
    while single_player != "king":

        player_threw = []
        single_player = random_card()
        if single_player == "king":
            player_threw.append(single_player)
            input(fore.GREEN_YELLOW + style.BOLD + f"{number_of_players()[count]} has voted to skip the challenge. Press 
Enter to Confirm..." + green)
            count+=1
            continue
        elif single_player in card_values:
            player_threw.append(single_player)
            input((fore.GREEN_YELLOW + style.BOLD + f"{number_of_players()[count]} threw {player_threw[0].capitalize()}. 
Press Enter to Confirm..."+ green))
            break
        else:
            input(fore.RED + style.BOLD +"Not a valid selection. " + fore.GREEN_YELLOW + style.BOLD + "Press Enter to 
Continue...")
    players_threw.append(player_threw[0])

def players_threw_clear():
    players_threw.clear()
```

random_card() conditional statements



```python
 def random_card():
     if len(sum_of_inputs) == 1:
         card_values = ["ace", "2" ,"3" ,"5" ,"8" , "king"]
         # create random number generator
         def throw_card():
             return(random.sample(card_values, k=1))
     else:
         card_values_range = run_next()
         def throw_card():
             return(random.sample(card_values_range, k=1))
     return("".join(throw_card()))
```

Code Logic and Control Flow Cont.

run_next() conditional statements

```
def run_next():
    thrown_cards = sum_of_inputs[-1]
    if len(thrown_cards) == len(number_of_players()):
        check_for_duplicates()
        if check_for_duplicates() == True:
            return duplicate_card_sorter()

        else:
            return card_sorter()
```

check_for_duplicates() conditional statements

```
def check_for_duplicates():
    thrown_cards = sum_of_inputs[-1]
    def remove_duplicates(x):
        return list(dict.fromkeys(x))
    thrown_cards_unique = remove_duplicates(thrown_cards)
    if len(thrown_cards) != len(thrown_cards_unique):
        return True
    else:
        return False
```

card_sorter() function

```
def card_sorter():
    thrown_cards = sum_of_inputs[-1]
    card_index={
                "ace": 0,
                "2": 1,
                "3": 2,
                "5": 3,
                "8" : 4,
            }
    card_index2={
                "ace": 0,
                "2": 1,
                "3": 2,
                "5": 3,
                "8" : 4
            }
    del card_index2[thrown_cards[0]]
    del card_index2[thrown_cards[1]]
    del card_index2[thrown_cards[2]]

    card_index2 = {i:j for i,j in card_index.items() if i not in card_index2}
    sorted_cards = sorted(card_index2.items(), key=lambda x:x[1])
    sorted_cards_slice = sorted_cards[1:-1]
    resultDictionary = dict((x, y) for x, y in sorted_cards_slice)
    card_range = {i:j for i,j in card_index2.items() if i not in resultDictionary}
    card_tuple = sorted(card_index.items(), key=lambda x:x[1])
    index_list = list(card_index)
    outside_list = list(card_range)
    card_range1= index_list.index(outside_list[0])
    card_range2= index_list.index(outside_list[1])
    remaining_cards = card_tuple[card_range1:card_range2+1]
    remaining_cards_dict = dict((x,y) for x, y in remaining_cards)
    remaining_cards_list = list(remaining_cards_dict)
    return remaining_cards_list
```

Append_remaining_card_values() function

```
def append_remaining_card_values(x):
    remaining_cards_values.append(card_sorter())
    return x
```

duplicate_card_sorter() function

```
def remove_duplicates(x):
    return list(dict.fromkeys(x))

def get_unique_cards():
    thrown_cards = sum_of_inputs[-1]
    thrown_cards_unique = remove_duplicates(thrown_cards)
    return thrown_cards_unique
```

Testing and TDD development Process

So, as I have mentioned I tried to adopt a test-driven development approach to the entire project. Essentially, I developed all my code in test environments which not only helped to break down problems into smaller challenges but turned out to be a really good way to learn all of the python fundamentals
through trial and error.

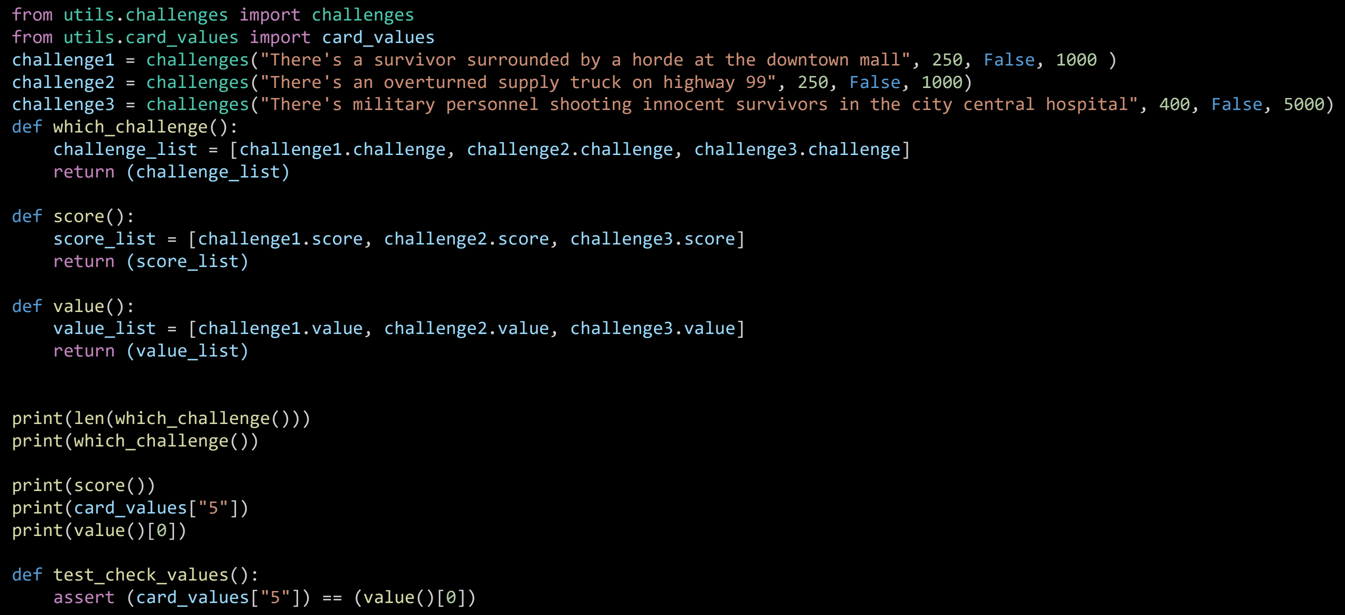
So, I suppose you could break down my methodology into 3 types of testing. I utilised assertion testing with pytest. I performed quite a bit of manual testing. Also, some testing using print statements which I found super useful in some of the more challenging parts of the project.

Examples of Assertion Tests in Pytest;

```
def test_round_checker():
    sum_of_inputs = [["King", "Ace", "2"],["King", "Ace", "2"],["King",
                      "Ace", "2"]]
    assert (len(sum_of_inputs))==(len(number_of_players()))
    print(len(sum_of_inputs))
    print(round)


def test_card_values():
    card_values={
        "Ace": 50,
        2: 100,
        3: 250,
        5: 500,
        8: 750,
        "king": 1000
    }
    assert("Ace" not in card_values.items()
    assert(2 in card_values)


 numbers = [1,2,3,4,5,6,7,8,9,10]

 def test_add_high_score():
     assert sum(numbers) == 55
```

Testing and TDD development Process Cont.

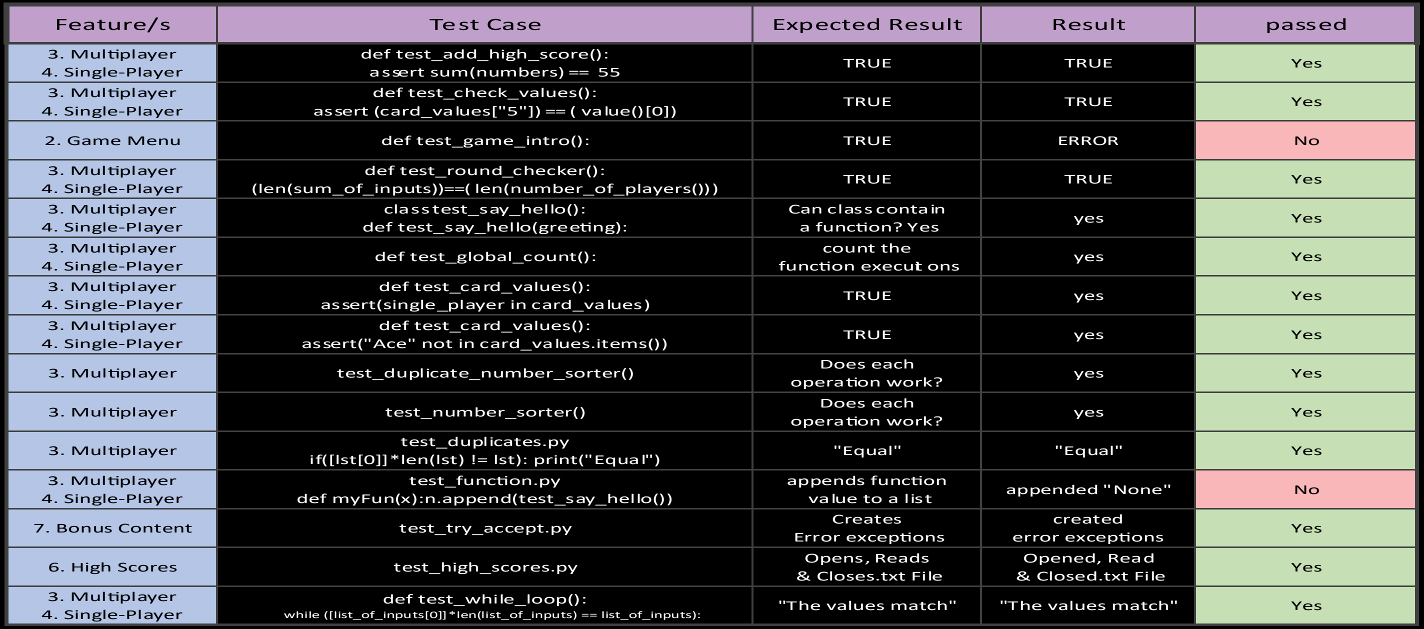
Examples of Assertion Tests in Pytest Cont.;



```python
from utils.challenges import challenges
from utils.card_values import card_values
challenge1 = challenges("There's a survivor surrounded by a horde at the downtown mall", 250, False, 1000 )
challenge2 = challenges("There's an overturned supply truck on highway 99", 250, False, 1000)
challenge3 = challenges("There's military personnel shooting innocent survivors in the city central hospital", 400, False, 5000)
def which_challenge():
    challenge_list = [challenge1.challenge, challenge2.challenge, challenge3.challenge]
    return (challenge_list)

def score():
    score_list = [challenge1.score, challenge2.score, challenge3.score]
    return (score_list)

def value():
    value_list = [challenge1.value, challenge2.value, challenge3.value]
    return (value_list)


print(len(which_challenge()))
print(which_challenge())

print(score())
print(card_values["5"])
print(value()[0])

def test_check_values():
    assert (card_values["5"]) == (value()[0])
```

Testing and TDD development Process Cont.

Manual Testing Spread-Sheet



Feature/s	Test Case	Expected Result	Result	passed
3. Multiplayer 4. Single-Player	def test_add_high_score(): assert sum(numbers) == 55	TRUE	TRUE	Yes
3. Multiplayer 4. Single-Player	def test_check_values(): assert (card_values["5"]) == (value()[0])	TRUE	TRUE	Yes
2. Game Menu	def test_game_intro():	TRUE	ERROR	No
3. Multiplayer 4. Single-Player	def test_round_checker(): (len(sum_of_inputs))==(len(number_of_players()))	TRUE	TRUE	Yes

3. Multiplayer
4. Single-Player
	class test_say_hello():
	def test_say_hello(greeting):
		Can class contain a function? Yes	yes	Yes

3. Multiplayer 4. Single-Player	def test_global_count():	count the function executions	yes	Yes

3. Multiplayer
4. Single-Player
	def test_card_values():
	assert(single_player in card_values)
		TRUE	yes	Yes

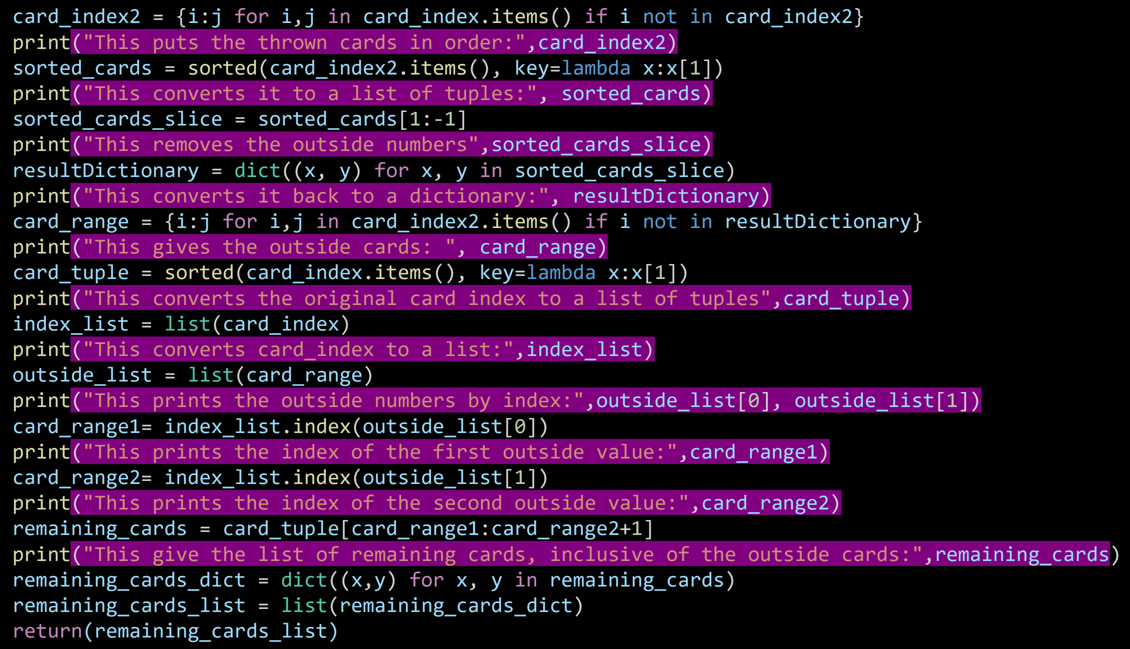
3. Multiplayer
4. Single-Player
	def test_card_values():
	assert("Ace" not in card_values.items())
		TRUE	yes	Yes

3. Multiplayer	test_duplicate_number_sorter()	Does each operation work?	yes	Yes
3. Multiplayer	test_number_sorter()	Does each operation work?	yes	Yes
3. Multiplayer	test_duplicates.py if([lst[0]]*len(lst) != lst): print("Equal")	"Equal"	"Equal"	Yes
3. Multiplayer 4. Single-Player	test_function.py def myFun(x):n.append(test_say_hello())	appends function value to a list	appended "None"	No
7. Bonus Content	test_try_accept.py	Creates Error exceptions	created error exceptions	Yes
6. High Scores	test_high_scores.py	Opens, Reads & Closes.txt File	Opened, Read & Closed.txt File	Yes

3. Multiplayer
4. Single-Player
	def test_while_loop():
	while ([list_of_inputs[0]]*len(list_of_inputs) == list_of_inputs):
		"The values match"	"The values match"	Yes

Testing and TDD development Process Cont.

Testing using print statements – card_sorter()



```
card_index2 = {i:j for i,j in card_index.items() if i not in card_index2}
print("This puts the thrown cards in order:",card_index2)
sorted_cards = sorted(card_index2.items(), key=lambda x:x[1])
print("This converts it to a list of tuples:", sorted_cards)
sorted_cards_slice = sorted_cards[1:-1]
print("This removes the outside numbers",sorted_cards_slice)
resultDictionary = dict((x, y) for x, y in sorted_cards_slice)
print("This converts it back to a dictionary:", resultDictionary)
card_range = {i:j for i,j in card_index2.items() if i not in resultDictionary}
print("This gives the outside cards: ", card_range)
card_tuple = sorted(card_index.items(), key=lambda x:x[1])
print("This converts the original card index to a list of tuples",card_tuple)
index_list = list(card_index)
print("This converts card_index to a list:",index_list)
outside_list = list(card_range)
print("This prints the outside numbers by index:",outside_list[0], outside_list[1])
card_range1= index_list.index(outside_list[0])
print("This prints the index of the first outside value:",card_range1)
card_range2= index_list.index(outside_list[1])
print("This prints the index of the second outside value:",card_range2)
remaining_cards = card_tuple[card_range1:card_range2+1]
print("This give the list of remaining cards, inclusive of the outside cards:",remaining_cards)
remaining_cards_dict = dict((x,y) for x, y in remaining_cards)
remaining_cards_list = list(remaining_cards_dict)
return(remaining_cards_list)
```

Review of Development Build Process

My development build process worked out well in that it felt like as I was developing each feature, the previous part of the program was instrumental in the development of the next. Adopting a TDD approach proved very helpful as I was able to develop most of my code in a testing environment. As problems arose, I was consistent in solving them quickly however I did get stuck trying to solve the random card to be thrown within the new range issue and went down some pretty wild rabbit holes, but I got there in the end.

I learned so much about python and programming in general that I feel I will be a lot more concise on future projects. By the end of the project my understanding of the DRY fundamentals was cemented even if my code began a little WET. Due to the time constraints, I didn't feel comfortable refactoring all of my code, but I mostly cleaned up all the stray elements and made sure everything was within functions to be called instead of repeating code wherever possible.

Take Aways

- ❑ Impact
- ❑ What I learned

Takeaways

Impact:

The terminal app hopefully showcases my new abilities to firstly, write in python programming language and adhere somewhat to fundamental pep 8 styling. Understand python functionality and code logic by writing concise functions containing nested loops and conditional statements. Understand control flow and utilising the correct methods for the right applications. Adhere to a TDD approach and DRY coding fundamentals.

What I learned:

While developing the terminal app I learned a tonne about python coding language, it's functionality and extent of it's capabilities.

I learned about objects and the classes that define them. Variables, functions and data types, conditional statements, break, continue, pass and return statemens, while & for loops etc…

I learned about external packages and importing them as well as organising my code into functions and modules and importing those also. Error Handling, File Handling, as well as some bash scripting to write my executable file.

Thank you!

Thank you for your time reviewing my work on the portfolio website project.
CA Student Number: 13868
email:1368@coderacademy.edu.au
alternate email: jim@jamjuicecreative.com.au
Phone number: 0450 321 332