BREAKTHROUGH!

**Programming Tasks (Mark Scheme)**

# Task 1 (2 marks)

## Coding

* Printing out the number of cards left in the deck correctly each turn **[1 mark]**

### Example Solution

print(self.\_\_Sequence.GetCardDisplay())

*# CODE ADDED*

print(**f**"There are {self.\_\_Deck.GetNumberOfCards()} cards left in the deck.")

*# END ADDITION*

print(self.\_\_Hand.GetCardDisplay())

## Testing:

* Printing out the number of cards left correctly between SEQUENCE and HAND **[1 mark]** 🡳



# Task 2 (5 marks)

## Coding:

* Changing GetChoice to show Peek (even if it doesn’t check GetPeekUsed) **[1 mark]**
* Changing PlayGame to accept ‘P’ and printing out the three cards in the deck (regardless of the format of the output) **[1 mark]**
* Adding the PeekUsed attribute with get/set methods to Lock **[1 mark]**

### Example Solution

Changes to GetChoice

**def** \_\_GetChoice(self):

        print()

*# CHANGE*

        if self.\_\_CurrentLock.GetPeekUsed():

          Choice = input("(D)iscard inspect, (U)se card:> ").upper()

        else:

          Choice = input("(D)iscard inspect, (U)se card, (P)eek:> ").upper()

*# END CHANGE*

        return Choice

Changes to PlayGame

          print(self.\_\_Discard.GetCardDisplay())

*# CODE ADDED*

        elif MenuChoice == "P":

          if not self.\_\_CurrentLock.GetPeekUsed():

            cards = ""

            for i in range(3):

              cards += self.\_\_Deck.GetCardDescriptionAt(i)+", "

            print("The next three cards are: "+ cards[:-2])

            self.\_\_CurrentLock.SetPeekUsed()

*# END ADDITION*

        elif MenuChoice == "U":

Changes to Lock

**def** \_\_init\_\_(self):

        self.\_Challenges = []

*# CODE ADDED*

        self.\_\_PeekUsed = False

*# END ADDITION*

…

*# CODE ADDED*

**def** GetPeekUsed(self):

      return self.\_\_PeekUsed

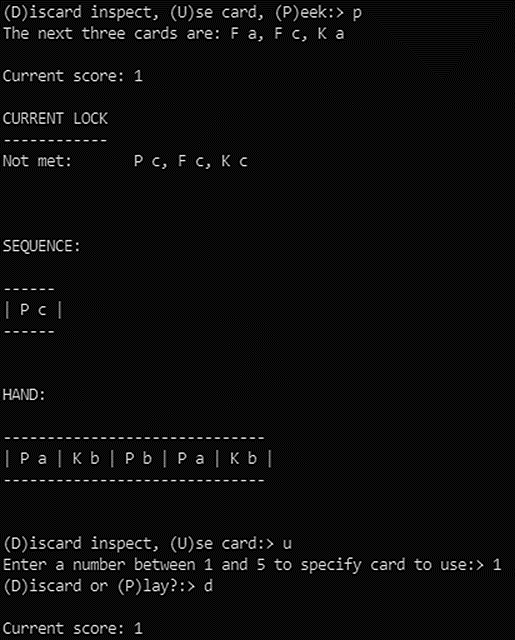
**def** SetPeekUsed(self):

      self.\_\_PeekUsed = True

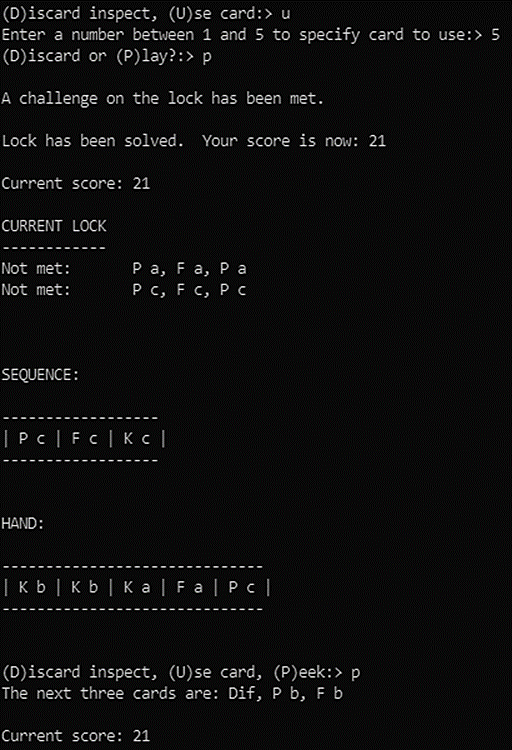
*# END ADDITION*

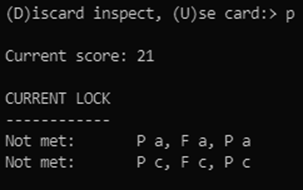
## Testing:

* Peek is an option, works correctly and then disappears **[1 mark]** 🡳



* Peek reappears for the next lock and works and then disappears and doesn’t work **[1 mark]** 🡳





# Task 3 (3 marks)

## Coding:

* Checking for correct condition to print out the error **[1 mark]**
* Printing out a sensible error message with the card or tool type that is in error **[1 mark]**

### Example Solution

Changes to PlayCardToSequence

              self.\_\_GetCardFromDeck(CardChoice)

*# CODE ADDED*

            else:

              print(**f**"ERROR: The card you are trying to play ({self.\_\_Hand.GetCardDescriptionAt(CardChoice – 1)}) is the same type as the last card in the sequence.")

*# END ADDITION*

        else:

## Testing:

* Showing the error message and the hand and sequence afterwards confirming that the card was neither played nor discarded **[1 mark]** 🡳



# Task 4 (6 marks)

## Coding:

* Printing out the correct message only when a mulligan is available **[1 mark]**
* Adding the MulliganUsed attribute to Breakthrough and initialising it to False **[1 mark]**
* Implementing the mulligan to add all the cards from the player’s hand, the discard pile and the sequence to the deck **[1 mark]**
* Shuffling up and dealing again (and discarding any difficulty cards drawn) **[1 mark]**

### Example Solution

Changes to GetChoice

**def** \_\_GetChoice(self):

        print()

*# CHANGE*

        if self.\_\_MulliganUsed:

          Choice = input("(D)iscard inspect, (U)se card:> ").upper()

        else:

          Choice = input("(D)iscard inspect, (U)se card, (M)ulligan:> ").upper()

*# END CHANGE*

        return Choice

Changes to Breakthrough

        self.\_\_LoadLocks()

*# CODE ADDED*

        self.\_\_MulliganUsed = False

*# END ADDITION*

Changes to PlayGame

if MenuChoice == "D":

    print(self.\_\_Discard.GetCardDisplay())

*# CODE ADDED*

elif MenuChoice == "M":

    if not self.\_\_MulliganUsed:

*# move cards from sequence to deck*

      for Count in range(self.\_\_Sequence.GetNumberOfCards()):

        self.\_\_MoveCard(self.\_\_Sequence, self.\_\_Deck, self.\_\_Sequence.GetCardNumberAt(0))

*# move cards from discard pile to deck*

      for Count in range(self.\_\_Discard.GetNumberOfCards()):

        self.\_\_MoveCard(self.\_\_Discard, self.\_\_Deck, self.\_\_Discard.GetCardNumberAt(0))

*# move cards from hand to deck*

      for Count in range(self.\_\_Hand.GetNumberOfCards()):

        self.\_\_MoveCard(self.\_\_Hand, self.\_\_Deck, self.\_\_Hand.GetCardNumberAt(0))

*# shuffle up and deal*

      self.\_\_Deck.Shuffle()

      for Count in range(5):

        while self.\_\_Deck.GetCardDescriptionAt(0) == "Dif":

          self.\_\_MoveCard(self.\_\_Deck, self.\_\_Discard, self.\_\_Deck.GetCardNumberAt(0))

        self.\_\_MoveCard(self.\_\_Deck, self.\_\_Hand, self.\_\_Deck.GetCardNumberAt(0))

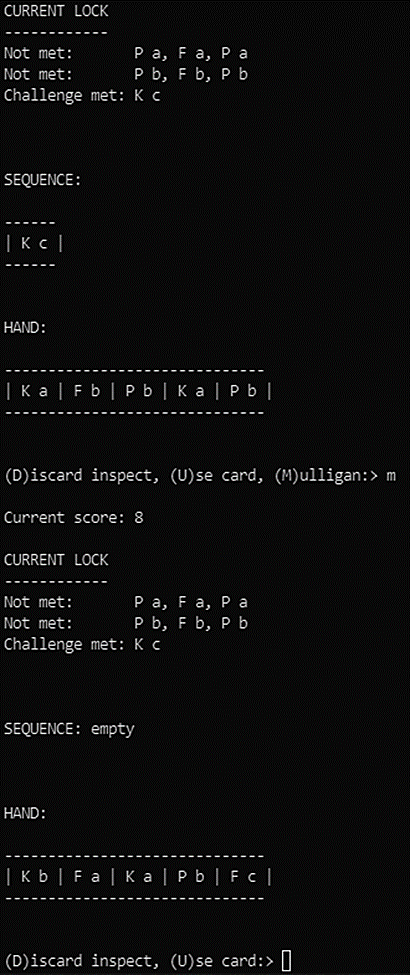
      self.\_\_MulliganUsed = True

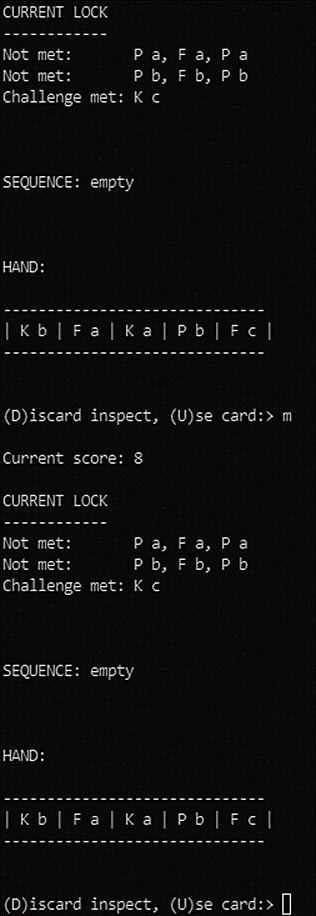
*# END ADDITION*

elif MenuChoice == "U":

## Testing:

* Showing a mulligan being used after solving a challenge **[1 mark]** 🡳





* Showing an attempt to use the mulligan again failing **[1 mark]** 🡲

# Task 5 (3 marks)

## Coding:

* Printing out quit as a menu option and including it in the selection statement in PlayGame **[1 mark]**
* Cleanly exiting the main game loop in PlayGame without using system.exit, goto or a similar ugly mechanism and successfully ending the program **[1 mark]**

### Example Solution

Changes to GetChoice

**def** \_\_GetChoice(self):

        print()

*# CHANGE*

        Choice = input("(D)iscard inspect, (U)se card, (Q)uit:> ").upper()

*# END CHANGE*

        return Choice

Changes to PlayGame

self.\_\_SetupGame()

*# CODE ADDED*

            hasQuit = False

*# END ADDITION*

            while not self.\_\_GameOver:

                self.\_\_LockSolved = False

                while not self.\_\_LockSolved and not self.\_\_GameOver and not hasQuit:

                    print()

                    print("Current score:", self.\_\_Score)

                    print(self.\_\_CurrentLock.GetLockDetails())

                    print(self.\_\_Sequence.GetCardDisplay())

                    print(self.\_\_Hand.GetCardDisplay())

                    MenuChoice = self.\_\_GetChoice()

                    if MenuChoice == "D":

                        print(self.\_\_Discard.GetCardDisplay())

*# CODE ADDED*

                    elif MenuChoice == "Q":

                      hasQuit = True

*# END ADDITION*

                    elif MenuChoice == "U":

                        CardChoice  = self.\_\_GetCardChoice()

                        DiscardOrPlay = self.\_\_GetDiscardOrPlayChoice()

                        if DiscardOrPlay == "D":

                            self.\_\_MoveCard(self.\_\_Hand, self.\_\_Discard,  
 self.\_\_Hand.GetCardNumberAt(CardChoice - 1))

                            self.\_\_GetCardFromDeck(CardChoice)

                        elif DiscardOrPlay == "P":

                            self.\_\_PlayCardToSequence(CardChoice)

                    if self.\_\_CurrentLock.GetLockSolved():

                        self.\_\_LockSolved = True

                        self.\_\_ProcessLockSolved()

*# ADDED the if and moved the existing code to the else*

                if hasQuit:

                  self.\_\_GameOver = True

                  self.\_\_Score += self.\_\_Deck.GetNumberOfCards()

                  print("Final score:", self.\_\_Score)

                else:

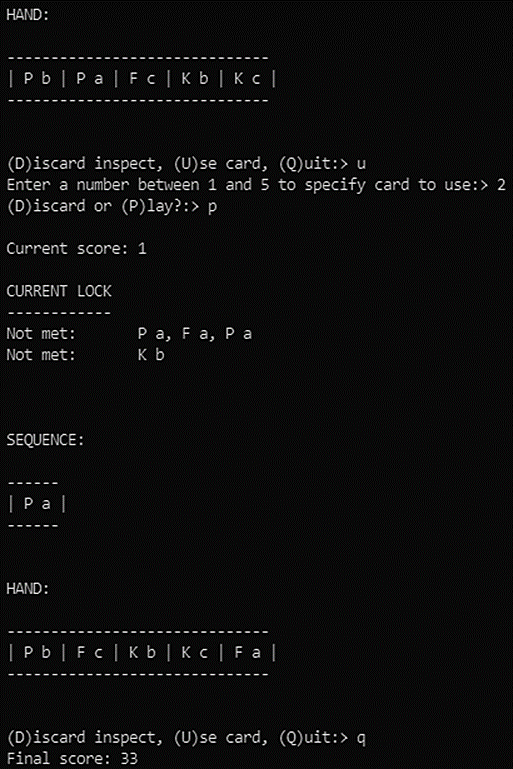
                  self.\_\_GameOver = self.\_\_CheckIfPlayerHasLost()

*# END CHANGE*

        else:

## Testing:

* Printing out a final score of 33 (if a Pick was played), 34 (if a File was played) or 35 (if a Key was played) **[1 mark]** 🡳



# Task 6 (5 marks)

## Coding:

* Adding the three attributes numPicks, numKeys and numFiles to the CardCollection class and initialising them to 0 **[1 mark]**
* Ensuring that at least one attribute is updated correctly when a card is added **[1 mark]**
* Ensuring that all three attributes are updated correctly when a card is removed **[1 mark]**
* Creating a DisplayStats method that will print out the percentage of each type of tool card remaining correctly (even if not to two decimal places). Note that ‘correctly’ means dividing the number of that tool by the total number of cards in the deck. **[1 mark]**

### Example Solution

Changes to the CardCollection class

*# CODE ADDED*

        self.\_\_numPicks = 0

        self.\_\_numFiles = 0

        self.\_\_numKeys = 0

*# END ADDITION*

**def** AddCard(self, C):

*# CODE ADDED*

      if C.GetDescription()[0] == "F":

        self.\_\_numFiles += 1

      elif C.GetDescription()[0] == "K":

        self.\_\_numKeys += 1

      elif C.GetDescription()[0] == "P":

        self.\_\_numPicks += 1

*# END ADDITION*

      self.\_Cards.append(C)

**def** RemoveCard(self, CardNumber):

      CardFound  = False

      Pos  = 0

      while Pos < len(self.\_Cards) and not CardFound:

        if self.\_Cards[Pos].GetCardNumber() == CardNumber:

          CardToGet = self.\_Cards[Pos]

          CardFound = True

          self.\_Cards.pop(Pos)

        Pos += 1

*# CODE ADDED*

      if CardToGet.GetDescription()[0] == "F":

        self.\_\_numFiles -= 1

      elif CardToGet.GetDescription()[0] == "K":

        self.\_\_numKeys -= 1

      elif CardToGet.GetDescription()[0] == "P":

        self.\_\_numPicks -= 1

*# END ADDITION*

      return CardToGet

*# CODE ADDED*

**def** DisplayStats(self):

      keyChance = self.\_\_numKeys/self.GetNumberOfCards() \* 100

      pickChance = self.\_\_numPicks/self.GetNumberOfCards() \* 100

      fileChance = self.\_\_numFiles/self.GetNumberOfCards() \* 100

      print(**f**"There is a {keyChance**:0.2f**}% chance that the next card will be a key, a {fileChance**:0.2f**}% chance that it will be a file and a {pickChance**:0.2f**}% chance that it will be a pick.")

*# END ADDITION*

Changes to GetCardFromDeck

                print(self.\_\_Hand.GetCardDisplay())

*# CODE ADDED*

                self.\_\_Deck.DisplayStats()

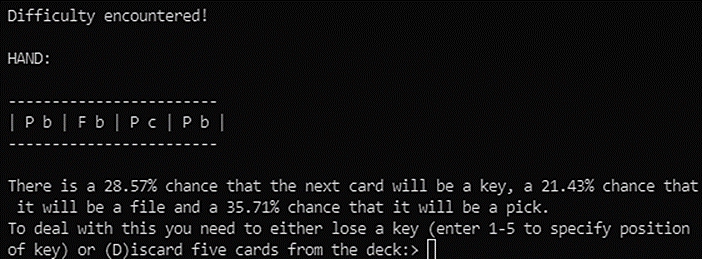
*# END ADDITION*

                print("To deal with this you need to either lose a key ", end='')

## Testing:

* Showing the percentage of at least one tool (even if incorrect) to two decimal places and printed out before asking which card the player would like to select or whether to discard from the deck.

Note that the percentages are unlikely to match the ones below. **[1 mark]** 🡳



# Task 7 (6 marks)

## Coding:

* Adding one multi-tool of each kind to the deck at creation time **[1 mark]**
* Adding one multi-tool of each kind to the deck whenever a lock is solved **[1 mark]**
* Adding the two SetCardToolkit methods that successfully allow a card’s toolkit to be changed from m **[1 mark]**
* Changing PlayCardToSequence to ask which toolkit the player would like whenever a multi-tool is played **[1 mark]**
* Calling SetCardToolkit for the correct card and toolkit from PlayCardToSequence **[1 mark]**

### Example Solution

Changes to CreateStandardDeck

**def** \_\_CreateStandardDeck(self):

*# CHANGE*

        NewCard = ToolCard("P", "m")

        self.\_\_Deck.AddCard(NewCard)

        NewCard = ToolCard("F", "m")

        self.\_\_Deck.AddCard(NewCard)

        NewCard = ToolCard("K", "m")

        self.\_\_Deck.AddCard(NewCard)

*# END CHANGE*

        for Count in range(5):

Changes to ProcessLockSolved

    self.\_\_MoveCard(self.\_\_Discard, self.\_\_Deck, self.\_\_Discard.GetCardNumberAt(0))

*# CHANGE*

        NewCard = ToolCard("P", "m")

        self.\_\_Deck.AddCard(NewCard)

        NewCard = ToolCard("F", "m")

        self.\_\_Deck.AddCard(NewCard)

        NewCard = ToolCard("K", "m")

        self.\_\_Deck.AddCard(NewCard)

*# END CHANGE*

        self.\_\_Deck.Shuffle()

Changes to PlayCardToSequence

**def** \_\_PlayCardToSequence(self, CardChoice):

*# CHANGE*

      if self.\_\_Hand.GetCardDescriptionAt(CardChoice -1)[2] == "m":

        toolkit = input("Which toolkit would you like to choose? ")

        self.\_\_Hand.SetCardToolkit(CardChoice -1, toolkit)

*# END CHANGE*

      if self.\_\_Sequence.GetNumberOfCards() > 0:

Creation of SetCardToolkit in CardCollection

**def** SetCardToolkit(self, pos, kit):

      if self.\_Cards[pos].GetDescription()[2] == "m":

        self.\_Cards[pos].SetCardToolkit(kit)

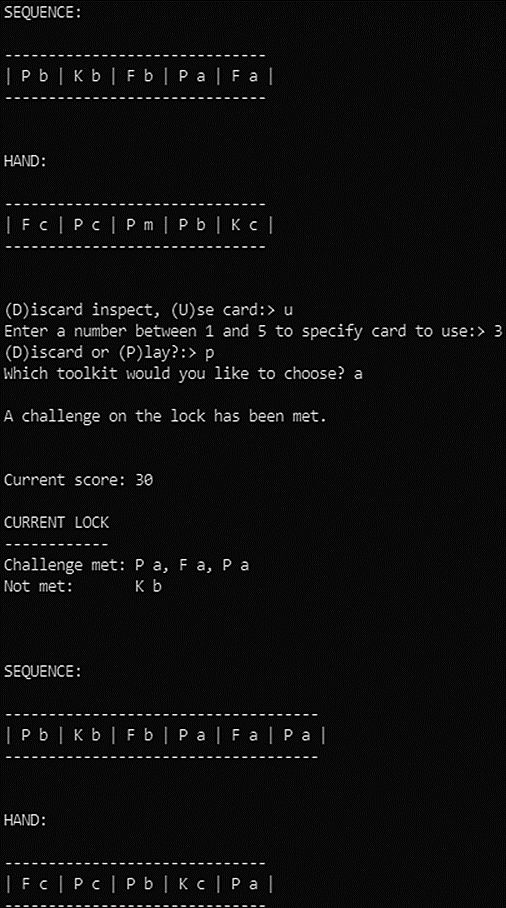
Creation of SetCardToolkit in ToolCard

**def** SetCardToolkit(self, kit):

        self.\_Kit = kit

## Testing:

* Showing the sequence updated with the card played of the toolkit chosen **[1 mark]** 🡳



# Task 8 (6 marks)

## Coding:

* Changing PlayGame to pass in the argument for the sequence to GetLockDetails and changing GetLockDetails to accept the new parameter **[1 mark]**
* Changing GetLockDetails to match a single card on the sequence to the first card of a challenge and change the message to partially met, also to not crash when there is an empty sequence **[1 mark]**
* Changing GetLockDetails to generate a partially met message when only the last card of the sequence matches the first card of a challenge **[1 mark]**
* Changing GetLockDetails to generate a partially met message when the last two cards of the sequence match the first two cards of a challenge. **[1 mark]**

### Example Solution

Changes to GetLockDetails

**def** GetLockDetails(self, sequence):

...

...

            else:

*# CHANGE*

                condition = C.GetCondition()

                if len(condition) == 3:

                  seqLen = sequence.GetNumberOfCards() - 1

                  if seqLen > 0 and condition[1] == sequence.GetCardDescriptionAt(seqLen) and condition[0] == sequence.GetCardDescriptionAt(seqLen-1):

                    LockDetails += "Partially met: "

                  elif seqLen >= 0 and condition[0] == sequence.GetCardDescriptionAt(seqLen):

                    LockDetails += "Partially met: "

                  else:

                    LockDetails += "Not met:       "

                else:

                  LockDetails += "Not met:       "

*# END CHANGE*

            LockDetails += self.\_\_ConvertConditionToString(C.GetCondition()) + "\n"

Changes to PlayGame

                    print("Current score:", self.\_\_Score)

*# CHANGE*

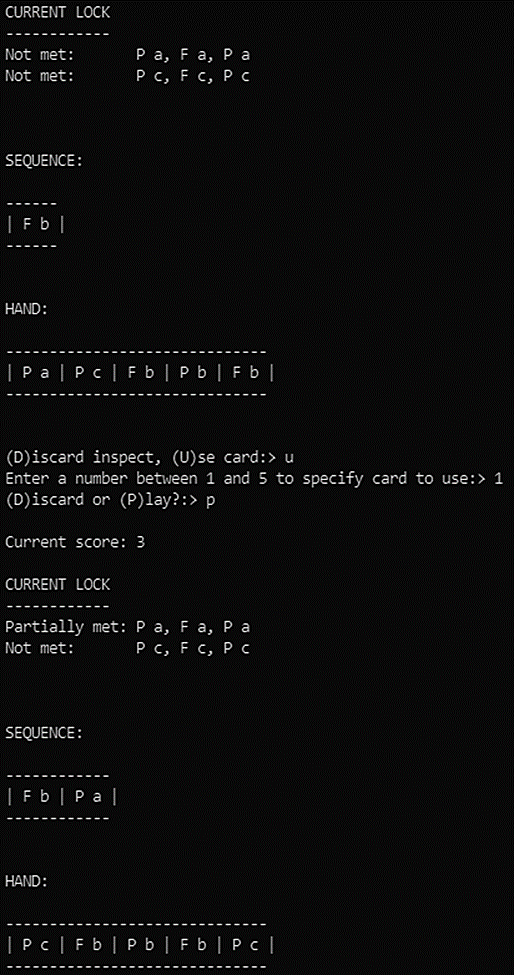
                    print(self.\_\_CurrentLock.GetLockDetails(self.\_\_Sequence))

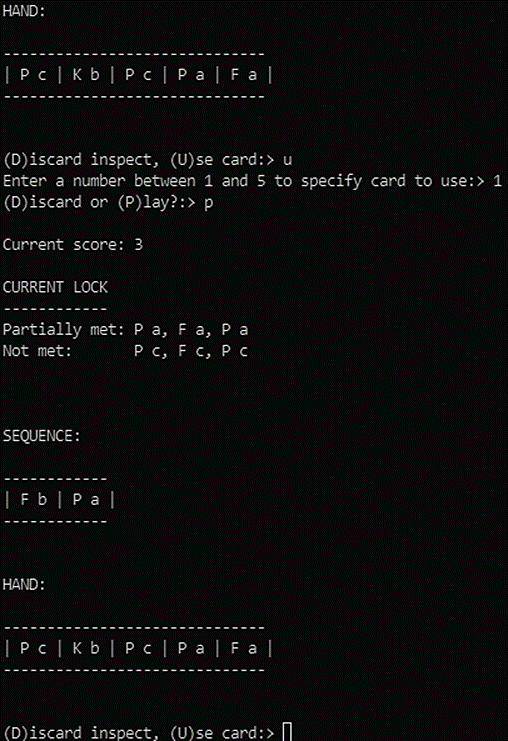
*# END CHANGE*

                    print(self.\_\_Sequence.GetCardDisplay())

## Testing:

* First card played to sequence doesn’t generate partially met (if it doesn’t match) and second card does **[1 mark]** 🡳





* Last two cards on the sequence matching first two of a challenge generates partially met **[1 mark]** 🡲

# Task 9 (5 marks)

## Coding:

* Adding a variable for bonusCounter and initialising it to 0 for each new lock **[1 mark]**
* Adding 1 to the variable each time a card is played or discarded **[1 mark]**
* Awarding the correct bonus once the lock is solved (including 0 if over 20 cards were played) and resetting the variable to 0 **[1 mark]**

### Example Solution

**def** PlayGame(self):

    if len(self.\_\_Locks) > 0:

        self.\_\_SetupGame()

        while not self.\_\_GameOver:

            self.\_\_LockSolved = False

*# CODE ADDED*

            bonusCounter = 0

*# END ADDITION*

            while not self.\_\_LockSolved and not self.\_\_GameOver:

                print()

                print("Current score:", self.\_\_Score)

                print(self.\_\_CurrentLock.GetLockDetails())

                print(self.\_\_Sequence.GetCardDisplay())

                print(self.\_\_Hand.GetCardDisplay())

                MenuChoice = self.\_\_GetChoice()

                if MenuChoice == "D":

                    print(self.\_\_Discard.GetCardDisplay())

                elif MenuChoice == "U":

                    CardChoice  = self.\_\_GetCardChoice()

                    DiscardOrPlay = self.\_\_GetDiscardOrPlayChoice()

*# CODE ADDED*

                    bonusCounter += 1

*# END ADDITION*

                    if DiscardOrPlay == "D":

                        self.\_\_MoveCard(self.\_\_Hand, self.\_\_Discard, self.\_\_Hand.GetCardNumberAt(CardChoice - 1))

                        self.\_\_GetCardFromDeck(CardChoice)

                    elif DiscardOrPlay == "P":

                        self.\_\_PlayCardToSequence(CardChoice)

                if self.\_\_CurrentLock.GetLockSolved():

                    self.\_\_LockSolved = True

                    self.\_\_ProcessLockSolved()

*# CODE ADDED*

                    bonus = max(0,20-bonusCounter)

                    self.\_\_Score += bonus

                    print("This locked awarded you",bonus,"bonus points.")

                    bonusCounter = 0

*# END ADDITION*

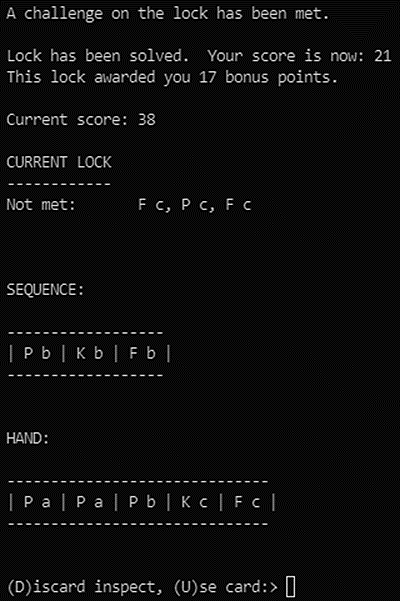
            self.\_\_GameOver = self.\_\_CheckIfPlayerHasLost()

    else:

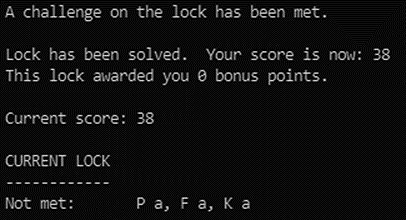
        print("No locks in file.")

## Testing:

* Solving a lock in under 20 cards and getting the correct bonus (which doesn’t have to match the score of 38 below)   
  **[1 mark]** 🡳



* Solving a lock in over 20 cards and getting 0 bonus. **[1 mark]** 🡳



# Task 10 (9 marks)

## Coding:

* Modifying SetupGame to have a 25% chance of adding a GeniusCard **[1 mark]**
* Modifying ProcessLockSolved to have a 25% chance of adding a GeniusCard **[1 mark]**
* Creating a GeniusCard class that inherits from Card, has a constructor with self as the only parameter and sets the CardType to Gen **[1 mark]**
* Asking the user to enter a challenge number or discard when a genius card is drawn **[1 mark]**
* Processing the GeniusCard correctly to solve the challenge chosen **[1 mark]**
* Processing the GeniusCard correctly to discard it **[1 mark]**
* Handling the discarding of a GeniusCard correctly if drawn while refilling the hand (i.e. move it to the discard pile and print a message) **[1 mark]**

### Example Solution

Creation of AddGeniusCardToDeck

**def** \_\_AddGeniusCardToDeck(self):

        self.\_\_Deck.AddCard(GeniusCard())

Creation of GeniusCard

*# CODE ADDED*

**class** GeniusCard(Card):

**def** \_\_init\_\_(self):

        self.\_CardType = "Gen"

        super().\_\_init\_\_()

**def** GetDescription(self):

        return self.\_CardType

**def** Process(self, CurrentLock, Choice):

        CurrentLock.SetChallengeMet(Choice-1,True)

*# END ADDITION*

Changes to SetupGame

self.\_\_AddDifficultyCardsToDeck()

*# CODE ADDED*

            if random.randint(1,4) == 1:

                self.\_\_AddGeniusCardToDeck()

*# END ADDITION*

            self.\_\_Deck.Shuffle()

Changes to ProcessLockSolved

            self.\_\_MoveCard(self.\_\_Discard, self.\_\_Deck, self.\_\_Discard.GetCardNumberAt(0))

*# CODE ADDED*

        if random.randint(1,4) == 1:

          self.\_\_AddGeniusCardToDeck()

*# END ADDITION*

        self.\_\_Deck.Shuffle()

Changes to GetCardFromDeck

                CurrentCard.Process(self.\_\_Deck, self.\_\_Discard, self.\_\_Hand, self.\_\_Sequence, self.\_\_CurrentLock, Choice, CardChoice)

*# CODE ADDED*

            elif self.\_\_Deck.GetCardDescriptionAt(0) == "Gen":

                CurrentCard = self.\_\_Deck.RemoveCard(self.\_\_Deck.GetCardNumberAt(0))

                print()

                print("Genius card encountered!")

                print("You can either use this card immediately to solve a challenge or discard it.")

                Choice = input(**f**"enter 1-{self.\_\_CurrentLock.GetNumberOfChallenges()} to solve a challenge or (D)iscard it so it can come up after reshuffling:> ").upper()

                if Choice.strip() == "D":

                    self.\_\_Discard.AddCard(CurrentCard)

                else:

                    CurrentCard.Process(self.\_\_CurrentLock, int(Choice))

*# END ADDITION*

        while self.\_\_Hand.GetNumberOfCards() < 5 and self.\_\_Deck.GetNumberOfCards() > 0:

            if self.\_\_Deck.GetCardDescriptionAt(0) == "Dif":

                self.\_\_MoveCard(self.\_\_Deck, self.\_\_Discard, self.\_\_Deck.GetCardNumberAt(0))

                print("A difficulty card was discarded from the deck when refilling the hand.")

*# CODE ADDED*

            elif self.\_\_Deck.GetCardDescriptionAt(0) == "Gen":

                self.\_\_MoveCard(self.\_\_Deck, self.\_\_Discard, self.\_\_Deck.GetCardNumberAt(0))

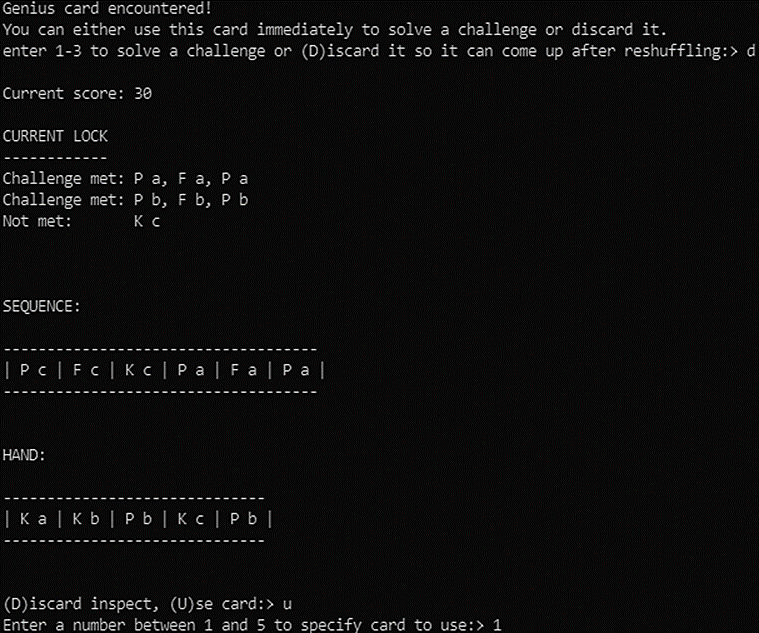
                print("A genius card was discarded from the deck when refilling the hand.")

*# END ADDITION*

            else:

## Testing:

* Using a GeniusCard successfully   
  **[1 mark]** 🡲
* Discarding a GeniusCard successfully **[1 mark]** 🡳



# Task 11 (12 marks)

## Coding:

* Adding the Credits attribute and initialising it to 10 **[1 mark]**
* Asking whether the player would like to buy a tool only when they have played or discarded a card and have at least 2 credits left **[1 mark]**
* Ensuring that keys are not listed if they have fewer than 3 credits remaining (even if they didn’t get the previous mark) **[1 mark]**
* Adding a tool card of the correct type and toolkit to the player’s hand in the fifth position **[1 mark]**
* Removing the tool card from the deck at the correct position **[1 mark]**
* Deducting the correct number of credits for buying a card **[1 mark]**
* Printing out a list of the correct number of each tool available and not printing tools where there are none available   
  **[1 mark]**
* Printing option 10 correctly at the end of the menu, once and once only **[1 mark]**
* Having an iteration statement to correctly calculate the number of tools of each type available **[1 mark]**
* Returning a list from PrintToolsAvailable that contains the index of a card with the chosen tool and kit **[1 mark]**

### Example Solution

Adding the Credits attribute

        self.\_\_LoadLocks()

*# CODE ADDED*

        self.\_\_Credits = 10

*# END ADDITION*

Changes to GetCardFromDeck

if self.\_\_Deck.GetNumberOfCards() > 0:

*# CODE ADDED*

            if self.\_\_Credits >= 2:

                Choice = input("Would you like to buy a tool (y/n)? ")

                if Choice.strip().lower() == "y":

                    ToolList = self.\_\_Deck.PrintToolsAvailable(True if self.\_\_Credits >= 3  
 else False)

                    CardChosen = int(input("Which tool would you like to buy? "))

                    if CardChosen != 10 and ToolList[CardChosen -1] != -1:

                        self.\_\_MoveCard(self.\_\_Deck, self.\_\_Hand,  
 self.\_\_Deck.GetCardNumberAt(ToolList[CardChosen-1]))

                        if CardChosen > 6:

                            self.\_\_Credits -= 3

                        else:

                            self.\_\_Credits -= 2

*# END ADDITION*

            if self.\_\_Deck.GetCardDescriptionAt(0) == "Dif":

New PrintToolsAvailable method in CardCollection

**def** PrintToolsAvailable(self, KeysAvailable):

        Tools = ["F a", "F b", "F c", "P a", "P b", "P c", "K a", "K b", "K c"]

        ToolList = [-1,-1,-1,-1,-1,-1,-1,-1,-1]

        ToolsAvailable = [0,0,0,0,0,0,0,0,0]

        for i in range(self.GetNumberOfCards()):

            if self.\_Cards[i].GetDescription() == "F a":

                ToolsAvailable[0] += 1

                if ToolList[0] == -1:

                    ToolList[0] = i

            elif self.\_Cards[i].GetDescription() == "F b":

                ToolsAvailable[1] += 1

                if ToolList[1] == -1:

                    ToolList[1] = i

            elif self.\_Cards[i].GetDescription() == "F c":

                ToolsAvailable[2] += 1

                if ToolList[2] == -1:

                    ToolList[2] = i

            elif self.\_Cards[i].GetDescription() == "P a":

                ToolsAvailable[3] += 1

                if ToolList[3] == -1:

                    ToolList[3] = i

            elif self.\_Cards[i].GetDescription() == "P b":

                ToolsAvailable[4] += 1

                if ToolList[4] == -1:

                    ToolList[4] = i

            elif self.\_Cards[i].GetDescription() == "P c":

                ToolsAvailable[5] += 1

                if ToolList[5] == -1:

                    ToolList[5] = i

            elif self.\_Cards[i].GetDescription() == "K a" and KeysAvailable:

                ToolsAvailable[6] += 1

                if ToolList[6] == -1:

                    ToolList[6] = i

            elif self.\_Cards[i].GetDescription() == "K b" and KeysAvailable:

                ToolsAvailable[7] += 1

                if ToolList[7] == -1:

                    ToolList[7] = i

            elif self.\_Cards[i].GetDescription() == "K c" and KeysAvailable:

                ToolsAvailable[8] += 1

                if ToolList[8] == -1:

                    ToolList[8] = i

        for i in range(len(Tools)):

            if ToolList[i] >= 0:

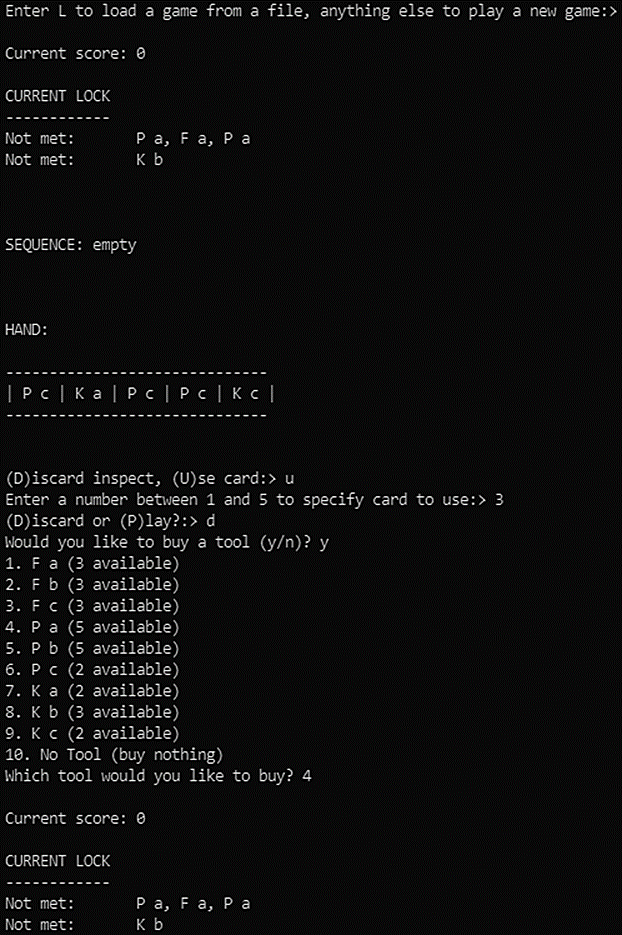
                print(**f**"{i+1}. {Tools[i]} ({ToolsAvailable[i]} available)")

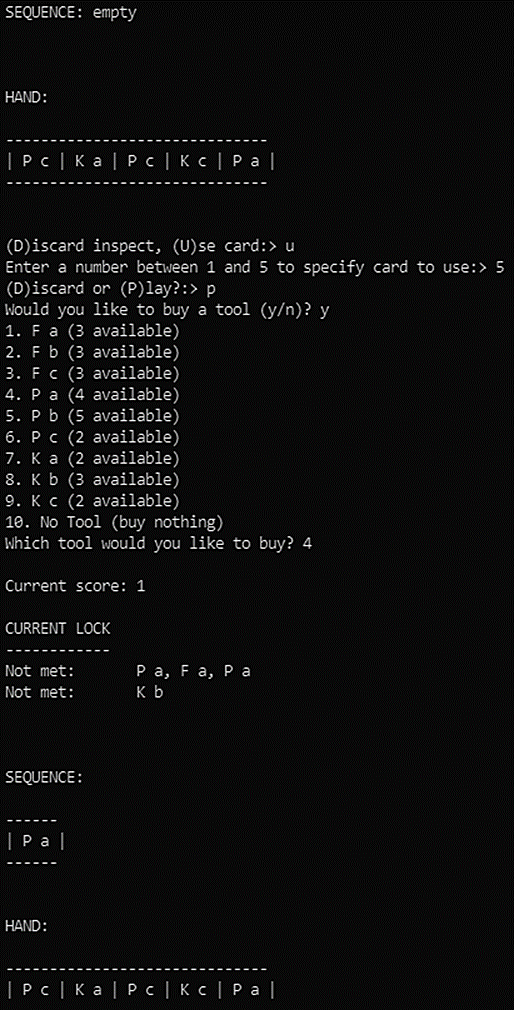
        print("10. No Tool (buy nothing)")

        return ToolList

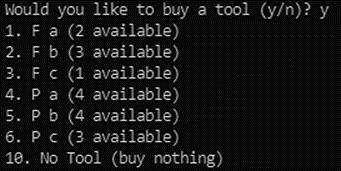
## Testing:

* Buying two tools **[1 mark]** 🡳





* Trying to buy a tool with 2 credits left **[1 mark]** 🡳



# Task 12 (4 marks)

## Coding:

* Selection statement to check whether the challenge just solved was at least three tool cards **[1 mark]**
* Iteration statement to run once for each tool card in the challenge just solved **[1 mark]**
* Call to MoveCard to move one tool card from the sequence to the discard pile inside the iteration statement **[1 mark]**

### Example Solution

if self.\_\_CurrentLock.CheckIfConditionMet(SequenceAsString):

*# CODE ADDED*

    if len(SequenceAsString) >= 13:

      for i in range(len(SequenceAsString) // 4):

        self.\_\_MoveCard(self.\_\_Sequence, self.\_\_Discard, self.\_\_Sequence.GetCardNumberAt(-1))

*# END ADDITION*

    return True

## Testing:

* Solve a challenge with one card, then with three cards **[1 mark]** 🡳



# Task 13 (8 marks)

## Coding:

* Changing GetChoice to correctly prompt you to (S)ave the game and PlayGame to call the new SaveGame method if ‘S’ was chosen **[1 mark]**
* Returning a string of the correct format for the save file from GetChallengesAsString method **[1 mark]**
* Returning a string of the correct format for the save file from GetChallengesMetAsString method **[1 mark]**
* Saving the current score to the save game file **[1 mark]**
* Saving the current lock to the save game file **[1 mark]**
* Saving the hand, sequence, deck and discard pile to the save game file **[1 mark]**
* Having a method or a loop that creates the string for a CardCollection in the correct format for the save game file **[1 mark]**

### Example Solution

Changes to GetChoice

**def** \_\_GetChoice(self):

        print()

*# CHANGE*

        Choice = input("(D)iscard inspect, (S)ave or (U)se card?:> ").upper()

*# END CHANGE*

        return Choice

Changes to PlayGame

                        print(self.\_\_Discard.GetCardDisplay())

*# CODE ADDED*

                    elif MenuChoice == "S":

                        self.\_\_SaveGame()

*# END ADDITION*

                    elif MenuChoice == "U":

Code for SaveGame

**def** \_\_SaveGame(self):

      saveName = "game1.txt"

      try:

        with open(saveName,"w") as saveFH:

          print(self.\_\_Score,file=saveFH)

          print(self.\_\_CurrentLock.GetChallengesAsString(),file=saveFH)

          print(self.\_\_CurrentLock.GetChallengesMetAsString(),file=saveFH)

          if self.\_\_Hand.GetNumberOfCards() > 0:

            hand = ",".join([**f**"{self.\_\_Hand.GetCardDescriptionAt(i)} {self.\_\_Hand.GetCardNumberAt(i)}" for i in range(self.\_\_Hand.GetNumberOfCards())])

          else:

            hand = ""

          print(hand,file=saveFH)

          if self.\_\_Sequence.GetNumberOfCards() > 0:

            seq = ",".join([**f**"{self.\_\_Sequence.GetCardDescriptionAt(i)} {self.\_\_Sequence.GetCardNumberAt(i)}" for i in range(self.\_\_Sequence.GetNumberOfCards())])

          else:

            seq = ""

          print(seq,file=saveFH)

          if self.\_\_Discard.GetNumberOfCards() > 0:

            discard = ",".join([**f**"{self.\_\_Discard.GetCardDescriptionAt(i)} {self.\_\_Discard.GetCardNumberAt(i)}" for i in range(self.\_\_Discard.GetNumberOfCards())])

          else:

            discard = ""

          print(discard,file=saveFH)

          if self.\_\_Deck.GetNumberOfCards() > 0:

            deck = ",".join([**f**"{self.\_\_Deck.GetCardDescriptionAt(i)} {self.\_\_Deck.GetCardNumberAt(i)}" for i in range(self.\_\_Deck.GetNumberOfCards())])

          else:

            deck = ""

          print(deck,file=saveFH)

        print("File saved.")

      except:

        print("File not saved")

Code for GetChallengesAsString

**def** GetChallengesAsString(self):

      challenges = ""

      for C in self.\_Challenges:

        if len(challenges) > 0:

          challenges += ";"

        challenges += self.\_\_ConvertConditionToString(C)

      return challenges

Code for GetChallengesMetAsString

**def** GetChallengesMetAsString(self):

      challenges = ""

      for C in self.\_Challenges:

        if len(challenges) > 0:

          challenges += ";"

        if C.GetMet():

          challenges += "Y"

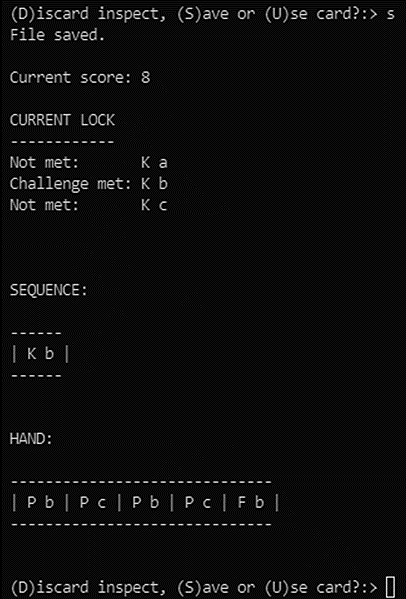
        else:

          challenges += "N"

      return challenges

## Testing:

* Saving game then loading game **[1 mark]** 🡳





# Task 14 (6 marks)

## Coding:

* Adding 5 to BonusPool after adding 5 to the score when completing a challenge **[1 mark]**
* Adding 5 to BonusPool when playing a card to the sequence that is a partial solution to a challenge **[1 mark]**
* Resetting BonusPool to 0 under all circumstances where a card is not played towards a challenge **[1 mark]**
* Creating the new attribute BonusPool and initialising it to 0 **[1 mark]**
* Writing the code for IsPartial such that it returns True if the card just played added to the solution or False if it didn’t add to an existing challenge **[1 mark]**

### Example Solution

Changes to PlayCardToSequence

            print()

*# CHANGE*

            self.\_\_Score += 5 + self.\_\_BonusPool

*# END CHANGE*

*# CODE ADDED*

            self.\_\_BonusPool += 5

        else:

            if self.\_\_CurrentLock.IsPartial(self.\_\_Sequence):

              self.\_\_BonusPool += 5

            else:

              self.\_\_BonusPool = 0

*# END ADDITION*

Changes to PlayGame

                            self.\_\_GetCardFromDeck(CardChoice)

*# CODE ADDED*

                            self.\_\_BonusPool = 0

*# END ADDITION*

                        elif DiscardOrPlay == "P":

Code for IsPartial

**def** IsPartial(self, seq):

      partial = False

      for C in self.\_Challenges:

        condition = C.GetCondition()

        if len(condition) == 3:

          seqLen = seq.GetNumberOfCards() - 1

          if seqLen > 0 and condition[1] == seq.GetCardDescriptionAt(seqLen) and condition[0] == seq.GetCardDescriptionAt(seqLen-1):

            partial = True

          elif seqLen >= 0 and condition[0] == seq.GetCardDescriptionAt(seqLen):

            partial = True

      return partial

Code for new BonusPool attribute

        self.\_\_LoadLocks()

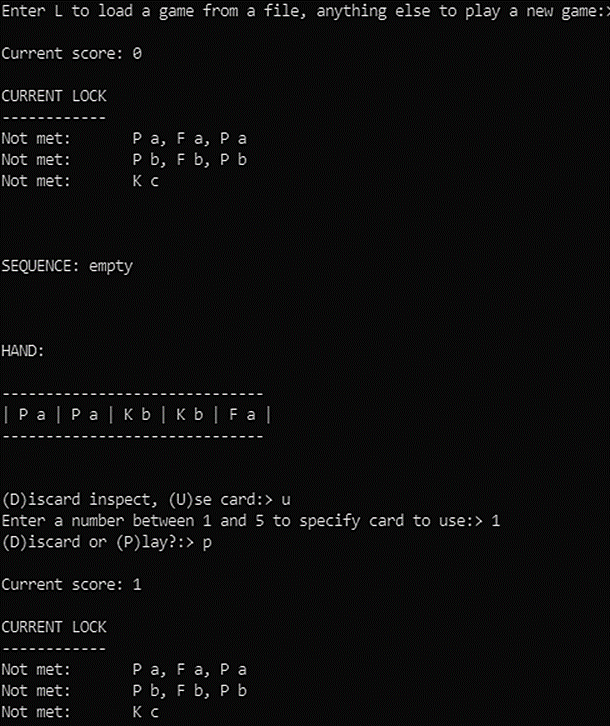
*# CODE ADDED*

        self.\_\_BonusPool = 0

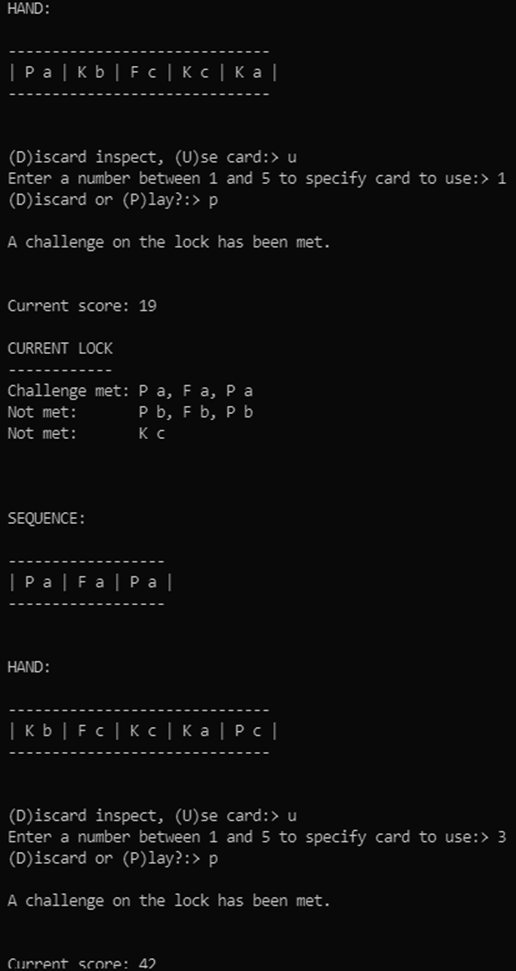
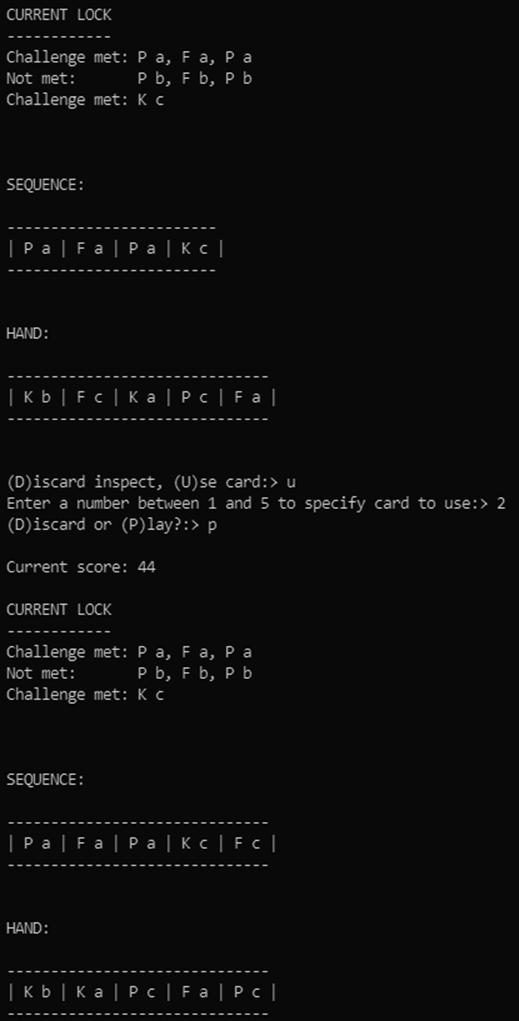
*# END ADDITION*

## Testing:

* Playing three cards to solve a challenge then solve it one card after another **[1 mark]** 🡳







# Task 15 (10 marks)

## Coding:

* Adding FinalLock as a private attribute and initialising it to 0 **[1 mark]**
* Adding the selection conditions for FinalLock == 1 and FinalLock == 2 to CheckIfPlayerHasLost (even if they don’t have the correct contents) **[1 mark]**
* Changing the condition of the selection statement in GetCardFromDeck correctly **[1 mark]**
* Changing ProcessLockSolved to have 10 attempts to find a soluble lock **[1 mark]**
* Changing ProcessLockSolved to call GenerateSolubleLock once 10 attempts have failed **[1 mark]**
* Changing ProcessLockSolved to set FinalLock to 1 and skip the main body of itself if FinalLock is 2 **[1 mark]**
* Writing GenerateSolubleLock such that it always generates a soluble lock (rule of not being able to play two tools of the same type excluded) **[1 mark]**
* Writing GenerateChallenge such that it generates a possible challenge from the cards left (meaning that it may not have two tools of the same type consecutively) **[1 mark]**
* Returning True and False correctly from IsSoluble **[1 mark]**

### Example Solution

Addition of FinalLock attribute

*# CODE ADDED*

        self.\_\_FinalLock = 0

*# END ADDITION*

Changes to CheckIfPlayerHasLost

**def** \_\_CheckIfPlayerHasLost(self):

*# CODE ADDED*

        if self.\_\_FinalLock == 1:

            self.\_\_FinalLock = 2

        elif self.\_\_FinalLock == 2:

            print("You have solved the final lock.  Your final score is:", self.\_\_Score)

            return True

*# END ADDITION*

        elif self.\_\_Deck.GetNumberOfCards() == 0:

Changes to GetCardFromDeck

                self.\_\_MoveCard(self.\_\_Deck, self.\_\_Hand, self.\_\_Deck.GetCardNumberAt(0))

*# CHANGE*

        if (self.\_\_Deck.GetNumberOfCards() == 0 and self.\_\_Hand.GetNumberOfCards() < 5 and self.\_\_FinalLock < 1) or self.\_\_Hand.GetNumberOfCards() == 0:

*# END CHANGE*

            self.\_\_GameOver = True

Changes to ProcessLockSolved

        print("Lock has been solved.  Your score is now:", self.\_\_Score)

*# CHANGE*

        if self.\_\_FinalLock < 2:

          while self.\_\_Discard.GetNumberOfCards() > 0:

              self.\_\_MoveCard(self.\_\_Discard, self.\_\_Deck, self.\_\_Discard.GetCardNumberAt(0))

          self.\_\_Deck.Shuffle()

          attempts = 0

          while attempts < 10:

            self.\_\_CurrentLock = self.\_\_GetRandomLock()

            if self.\_\_CurrentLock.IsSoluble(self.\_\_Deck, self.\_\_Hand):

              break

            else:

              attempts += 1

          if attempts == 10:

            print("Final Lock")

            self.\_\_CurrentLock = self.\_\_GenerateSolubleLock()

            self.\_\_FinalLock = 1

            self.\_\_GameOver = True

*# END CHANGE*

Code for GenerateSolubleLock

**def** \_\_GenerateSolubleLock(self):

      cardsLeft = []

      for i in range(self.\_\_Deck.GetNumberOfCards()):

        cardsLeft.append(self.\_\_Deck.GetCardDescriptionAt(i))

      for i in range(self.\_\_Hand.GetNumberOfCards()):

        cardsLeft.append(self.\_\_Hand.GetCardDescriptionAt(i))

      newLock = Lock()

      newLock.AddChallenge(self.\_\_GenerateChallenge(cardsLeft))

      newLock.AddChallenge(self.\_\_GenerateChallenge(cardsLeft))

      return newLock

Code for GenerateChallenge

**def** \_\_GenerateChallenge(self, cards):

      challenge = [random.choice(cards)]

      cards.remove(challenge[-1])

      try:

        for i in range(random.randint(0,2)):

          card = random.choice(cards)

          while card[0] == challenge[-1][0]:

            card = random.choice(cards)

          challenge.append(card)

          cards.remove(card)

      except:

        pass

*# ran out of cards so return what we have*

      return challenge

Code for IsSoluble

*# CODE ADDED*

**def** IsSoluble(self, deck, hand):

      cardsLeft = []

      for i in range(deck.GetNumberOfCards()):

        cardsLeft.append(deck.GetCardDescriptionAt(i))

      for i in range(hand.GetNumberOfCards()):

        cardsLeft.append(hand.GetCardDescriptionAt(i))

      challengesLeft = []

      for i in range(self.GetNumberOfChallenges()):

        challengesLeft.extend(self.\_Challenges[i].GetCondition())

      try:

        for card in challengesLeft:

          cardsLeft.remove(card)

        return True

      except:

        return False

*# END ADDITION*

## Testing:

* Printing out the final lock **[1 mark]** 🡳

