Full analysis of the 2024 skeleton code.

Try to make edits where you see fit, any help with making this as full of information as possible is greatly appreciated.

Table of Contents

[Imported modules and their functions. 5](#_Toc150327939)

[Random 5](#_Toc150327940)

[OS 5](#_Toc150327941)

[Main(): 5](#_Toc150327942)

[General Function 5](#_Toc150327943)

[Line By Line Analysis 5](#_Toc150327944)

[Puzzle(): 6](#_Toc150327945)

[General Function of the Class 6](#_Toc150327946)

[\_\_init\_\_() 6](#_Toc150327947)

[General Function 6](#_Toc150327948)

[Line By Line Analysis 6](#_Toc150327949)

[\_\_LoadPuzzle() 8](#_Toc150327950)

[General Function: 8](#_Toc150327951)

[Line By Line Analysis: 8](#_Toc150327952)

[AttemptPuzzle() 9](#_Toc150327953)

[General Function: 9](#_Toc150327954)

[Line By Line Analysis: 9](#_Toc150327955)

[\_\_GetCell() 11](#_Toc150327956)

[General Function 11](#_Toc150327957)

[Line By Line 11](#_Toc150327958)

[CheckForMatchWithPattern() 12](#_Toc150327959)

[General Function 12](#_Toc150327960)

[Line By Line 12](#_Toc150327961)

[\_\_GetSymbolFromUser 13](#_Toc150327962)

[General Function 13](#_Toc150327963)

[Line By Line 13](#_Toc150327964)

[\_\_CreateHorizontalLine 14](#_Toc150327965)

[General Function 14](#_Toc150327966)

[Line By Line 14](#_Toc150327967)

[Display Puzzle 14](#_Toc150327968)

[General Function 14](#_Toc150327969)

[Line By Line 14](#_Toc150327970)

[Pattern(): 15](#_Toc150327971)

[General Function of the Class 15](#_Toc150327972)

[\_\_init\_\_ 15](#_Toc150327973)

[General Function 15](#_Toc150327974)

[Line By Line 15](#_Toc150327975)

[MatchesPattern 15](#_Toc150327976)

[General Function 15](#_Toc150327977)

[Line By Line 15](#_Toc150327978)

[GetPatternSequence 16](#_Toc150327979)

[General Function 16](#_Toc150327980)

[Line By Line 16](#_Toc150327981)

[Cell(): 16](#_Toc150327982)

[General Function of the Class 16](#_Toc150327983)

[\_\_init\_\_ 16](#_Toc150327984)

[General Function 16](#_Toc150327985)

[Line By Line 16](#_Toc150327986)

[GetSymbol 16](#_Toc150327987)

[General Function 16](#_Toc150327988)

[Line By Line 16](#_Toc150327989)

[IsEmpty 17](#_Toc150327990)

[General Function 17](#_Toc150327991)

[Line By Line 17](#_Toc150327992)

[ChangeSymbolInCell 17](#_Toc150327993)

[General Function 17](#_Toc150327994)

[Line By Line 17](#_Toc150327995)

[CheckSymbolAllowed 17](#_Toc150327996)

[General Function 17](#_Toc150327997)

[Line By Line 17](#_Toc150327998)

[AddToNotAllowedSymbols 17](#_Toc150327999)

[General Function 17](#_Toc150328000)

[Line By Line 17](#_Toc150328001)

[UpdateCell 18](#_Toc150328002)

[General Function 18](#_Toc150328003)

[Line By Line 18](#_Toc150328004)

[BlockedCell(): 18](#_Toc150328005)

[General Function of the Class 18](#_Toc150328006)

[\_\_init\_\_ 18](#_Toc150328007)

[General Function 18](#_Toc150328008)

[Line By Line 18](#_Toc150328009)

[CheckSymbolAllowed 18](#_Toc150328010)

[General Function 18](#_Toc150328011)

[Line By Line 18](#_Toc150328012)

[Lines 244-245: 18](#_Toc150328013)

[General Function 18](#_Toc150328014)

[Line By Line 19](#_Toc150328015)

[Potential Questions Regarding the Skeleton Code 19](#_Toc150328016)

[UpdateCell 19](#_Toc150328017)

[Rewriting the Pattern Sequence for the Symbols. 19](#_Toc150328018)

# Imported modules and their functions.

## Random

The “random” module is used for random number generation, this is used to create more random puzzles.

## OS

The “os” module is used for direct interaction with the operating system of the device, making it easier for file handling and editing.

# Main():

## General Function

The general function of this subroutine is to run the game and ensure that the user can access the puzzles. It also includes an indefinite loop system that makes it such that the user can replay the game as many times as they please without having to re-run the program.

## Line By Line Analysis

009| def Main():   
# Declaration and definition of the function.  
010|    Again = "y"   
# This char variable is used as a pseudo-Boolean variable due to the way that it is used in the program.  
011|    Score = 0   
# The player score variable is created so that it may be edited later in the program, this avoids any errors and also makes sure that the contents of this variable are integers as is intended.  
012|    while Again == "y":   
# The indefinite loop in charge of seamless replay ability.  
013|        Filename = input("Press Enter to start a standard puzzle or enter name of file to load: ")  
# This is how the user is able to select a preexisting puzzle to play on if they please.  
014|        if len(Filename) > 0:  
# By checking the length of the user input, the program is able to determine whether or not the user would like to load a preexisting file or not, without having to check the contents of the variable itself.  
015|            MyPuzzle = Puzzle(Filename + ".txt")  
# If the length of the variable is greater than 1, a further check for the file is conducted.  
016|        else:  
017|            MyPuzzle = Puzzle(8, int(8 \* 8 \* 0.6))  
# Otherwise a fresh puzzle is created for the user to play on.  
018|        Score = MyPuzzle.AttemptPuzzle()  
# This is how the user is able to actually access the puzzle whilst also being how the user score is changed.  
019|        print("Puzzle finished. Your score was: " + str(Score))  
# End message containing the user’s achieved score.  
020|        Again = input("Do another puzzle? ").lower()  
# This prompts the user with a choice to play again or not, if the user inputs any value other than “y”, the loop on line “012” is not preformed and the program ends.

# Puzzle():

## General Function of the Class

This class oversees the creation, loading and playing of the puzzle itself. This is done using the different functions that make up the class.

## \_\_init\_\_()

### General Function

Behind the scenes setup of the class whilst also calling any new variables that are going to be useful for the class.

This also contains the code related to the creation of new patterns that will be able to be used in play.

### Line By Line Analysis

023|def \_\_init\_\_(self, \*args):  
# Declaration of the initialiser function, note the usage of “\*agrs” to allow many arguments to be entered when called, this is important when noting how lines 014 – 017 functioned.

024|    if len(args) == 1:  
# The condition that takes into consideration how many arguments were entered when the class was called, this would run if the user had entered a file name as the condition related to that would only input one argument into the class.

025|        self.\_\_Score = 0

# Creating the variable \_\_Score, this is because the Score variable on line 014 was only local to the main function.

026|        self.\_\_SymbolsLeft = 0

# Creating the variable \_\_SymbolsLeft, this oversees counting how many symbols the user is allowed to place which is set at 0 until the puzzle is loaded.

027|      self.\_\_GridSize = 0  
# Creating the variable \_\_GridSize, this oversees how large the grid is and is set at 0 until the puzzle is loaded

028|        self.\_\_Grid = []

# Creating the array \_\_Grid. This is the actual playing field of the game and will be edited accordingly when the puzzle is loaded.

029|        self.\_\_AllowedPatterns = []  
# Creating the array \_\_AllowedPatterns, this would contain what patterns the player is allowed to use whilst playing.

030|        self.\_\_AllowedSymbols = []  
# Creating the array \_\_AllowedSymbols, this would contain what symbols the player is allowed to use whilst playing.

031|        self.\_\_LoadPuzzle(args[0])  
# This loads the puzzle using the file name inputted in line 013.

032|    else:  
# If the user did not input anything in line 013, then the condition would have inputted two arguments when calling the class.

033|        self.\_\_Score = 0  
# Score reset

034|        self.\_\_SymbolsLeft = args[1]  
# Creates the variable \_\_SymbolsLeft, this uses the second argument inputted which was “8 \* 8 \* 0.6” which would equal 38, therefore the user would have 38 symbols at their disposal.

035|        self.\_\_GridSize = args[0]  
# Creates the variable \_\_GridSize, this uses the first argument inputted which was “8”.

036|        self.\_\_Grid = []  
# Creates the array \_\_Grid, this is the plaing field of the puzzle.

037|        for Count in range(1, self.\_\_GridSize \* self.\_\_GridSize + 1):

# This definite loop will run the code below 64 times, this is in accordance with the variable \_\_GridSize in which the number of times the code is run is the same as the square of the variable.

038|            if random.randrange(1, 101) < 90:  
# A condition is called with a random number generator where a number is generated between 1 and 101.

039|                C = Cell()  
# If the number generated is lower than 90, then C is equal to the output of the class Cell().

040|            else:

041|                C = BlockedCell()  
# If the number generated is higher than 90, then C is equal to the output of the class BlockedCell().

042|            self.\_\_Grid.append(C)

# After the result of the condition, whatever option was generated would be added to the \_\_Grid array, creating the playing field.

043|        self.\_\_AllowedPatterns = []  
# Creates an empty array \_AllowedPatterns, this will contain the patterns that are allowed to be used in the puzzle.

044|        self.\_\_AllowedSymbols = []  
# Creates an empty array \_\_AllowedSymbols, this will contain the symbols that are allowed to be used in the puzzle.

045|        QPattern = Pattern("Q", "QQ\*\*Q\*\*QQ")

# This creates the first pattern that can be used using the class Pattern().

046|        self.\_\_AllowedPatterns.append(QPattern)  
# The new pattern is then added to the \_\_AllowedPatterns array.

047|        self.\_\_AllowedSymbols.append("Q")  
# The symbol associated with the new pattern is added to the \_\_AllowedSymbols array.

048|        XPattern = Pattern("X", "X\*X\*X\*X\*X")  
# This creates the second pattern that can be used using the class Pattern().

049|        self.\_\_AllowedPatterns.append(XPattern)  
# The new pattern is then added to the \_\_AllowedPatterns array.

050|        self.\_\_AllowedSymbols.append("X")  
# The symbol associated with the new pattern is added to the \_\_AllowedSymbols array.

051|        TPattern = Pattern("T", "TTT\*\*T\*\*T")  
# This creates the third pattern that can be used using the class Pattern().

052|        self.\_\_AllowedPatterns.append(TPattern)  
# The new pattern is then added to the \_\_AllowedPatterns array.

053|        self.\_\_AllowedSymbols.append("T")  
# The symbol associated with the new pattern is added to the \_\_AllowedSymbols array.

## \_\_LoadPuzzle()

### General Function:

This subroutine’s main function is to attempt to load a puzzle if the user decides to pick a preexisting puzzle.

This also has an exit clause in case the inputted puzzle name does not exist.

### Line By Line Analysis:

055|def \_\_LoadPuzzle(self, Filename):

# Creation of the subroutine, this takes the filename inputted at line 013 as an argument.

056| try:

# Since there is a possibility of mistyping or simply inputting an incorrect file name, error handling will be a necessary step to take in such a project as this one.

057|        with open(Filename) as f:

# This opens the file and assigns its contents to a variable “f”.

058|            NoOfSymbols = int(f.readline().rstrip())  
# In the file itself the number of symbols that are usable in this puzzle is shown.

059|            for Count in range (1, NoOfSymbols + 1):

# The number of symbols that are usable is then used to create a loop

060|                self.\_\_AllowedSymbols.append(f.readline().rstrip())

# This loop adds the allowed symbols that are also listed in the puzzle file itself into the \_\_AllowedSymbols array.

061|            NoOfPatterns = int(f.readline().rstrip())

# The program then reads froms the file the amount of patterns that exist in this puzzle.

062|            for Count in range(1, NoOfPatterns + 1):

# This too is used in a loop

063|                Items = f.readline().rstrip().split(",")

# First it turns each entry into a list by splitting the line where there is a comma. For reference, each pattern entry in the file looks like this ” Q,QQ\*\*Q\*\*QQ”.

064|                P = Pattern(Items[0], Items[1])

# It then creates a pattern using the two parts of the new list.

065|                self.\_\_AllowedPatterns.append(P)

# And then adds the pattern to the ­­\_\_AllowedPatterns array.

066|            self.\_\_GridSize = int(f.readline().rstrip())

# The grid size is also a part of the file and it too is being read.

067|            for Count in range (1, self.\_\_GridSize \* self.\_\_GridSize + 1):

# It is then used to create the playing field.

068|                Items = f.readline().rstrip().split(",")

# A list by the name of Items is made to show what is meant to be in each element of the grid.

069|                if Items[0] == "@":  
# If the element has the “@” symbol in it, that signals that the cell is meant to be blocked.

070|                    C = BlockedCell()  
# Blocking the cell

071|                    self.\_\_Grid.append(C)  
# Adding the blocked cell to the grid.

072|                else:

073|                    C = Cell()  
# If the cell does not have the “@” symbol in it, then it does not need to be blocked and therefore a normal cell in inserted.

074|                    C.ChangeSymbolInCell(Items[0])  
# However if there is another kind of symbol in the cell, then the cell is changed to that other symbol.

075|                    for CurrentSymbol in range(1, len(Items)):

076|                        C.AddToNotAllowedSymbols(Items[CurrentSymbol])  
# This makes it so that the user cannot input the current symbol into that occupied cell.

077|                    self.\_\_Grid.append(C)  
# Inserts the cell into the grid.

078|            self.\_\_Score = int(f.readline().rstrip())  
# This reads the score that was left on the puzzle document. This is the score that the user starts off with.

079|            self.\_\_SymbolsLeft = int(f.readline().rstrip())  
# This reads the amount of symbols left for the puzzle.

080| except:

081|        print("Puzzle not loaded")

# If the user inputs a file name that does not exist, this error message is shown instead.

## AttemptPuzzle()

### General Function:

Allows the user to access the puzzle so that they can begin to play the game.

### Line By Line Analysis:

083|def AttemptPuzzle(self):

# Creation of the function.

084|    Finished = False

# Boolean variable that identifies if the game is still in progress.

085|    while not Finished:

# The following code will continue to run whist the variable Finished remains False.

086|        self.DisplayPuzzle()

# This runs the function DisplayPuzzle() which oversees providing the visuals for the game.

087|        print("Current score: " + str(self.\_\_Score))

# This then prints the current score of the game at hand.

088|        Row = -1

# Creation of an integer variable “Row”, this is set at -1 to ensure that a valid row isn’t edited if a piece of code fails.

089|        Valid = False

# Boolean variable “Valid” is created and is set at False at default.

090|        while not Valid:

# Loop that runs if the “Valid” variable is set at False

091|            try:

092|                Row = int(input("Enter row number: "))

# User is prompted to enter a row number

093|                Valid = True

# If no errors occur (aka if the user inputs an integer that is valid rather than some other symbol) the “Valid” variable is changed to True which allows for an escape from the loop

094|            except:

095|                pass

# However, if an error does occur (aka the user inputs an invalid symbol) the “Valid” variable stays the same resulting in the loop making another pass.

096|        Column = -1

# Creation of integer variable “Column”, this is set at -1 to ensure that that a valid column isn’t edited if a piece of code fails

097|        Valid = False

# Boolean variable “Valid” is reassigned to False.

098|        while not Valid:

# Loop that runs when the variable is set at False.

099|            try:

100|                Column = int(input("Enter column number: "))

# User is prompted to enter a column number

101|                Valid = True

# If no errors occur (aka if the user inputs an integer that is valid rather than some other symbol) the “Valid” variable is changed to True which allows for an escape from the loop

102|            except:

103|                pass

# However, if an error does occur (aka the user inputs an invalid symbol) the “Valid” variable stays the same resulting in the loop making another pass.

104|        Symbol = self.\_\_GetSymbolFromUser()

# The variable “Symbol” gets the information from the function \_\_GetSymbolFromUser() and assigns it to itself

105|        self.\_\_SymbolsLeft -= 1  
# Every time the user inputs a valid symbol, the variable \_\_SymbolsLeft decreases by one.

106|        CurrentCell = self.\_\_GetCell(Row, Column)

# This gets the current cell in a form where it is the index of the array \_\_List

107|        if CurrentCell.CheckSymbolAllowed(Symbol):

# This checks if the symbol at the current position is allowed

108|            CurrentCell.ChangeSymbolInCell(Symbol)

# If the symbol is allowed, it will change the symbol in the cell with the new valid symbol.

109|            AmountToAddToScore = self.CheckforMatchWithPattern(Row, Column)

# This adds an amount to the player’s score if a valid pattern is made with that new symbol change.

110|            if AmountToAddToScore > 0:

111|                self.\_\_Score += AmountToAddToScore

# If the amount that needs to be added is higher than 0 then the new score is added to the player’s total score

112|        if self.\_\_SymbolsLeft == 0:

113|            Finished = True

# If the user plays all of their allowed symbols, then the Finished variable is switched to True which means the program no longer has to go through the loop

114|    print()

115|    self.DisplayPuzzle()

116|    print()

# This displays the current state of the puzzle, the two print functions that surround it are there to create space between the different parts of the outputs

117|    return self.\_\_Score

# The player’s total score is then returned to the variable that was used to call this function.

## \_\_GetCell()

### General Function

Self-explanatory, it gets the position of the cell called as an index of the grid.

### Line By Line

119|def \_\_GetCell(self, Row, Column):

# Creation of the function

120|    Index = (self.\_\_GridSize - Row) \* self.\_\_GridSize + Column – 1

# Variable “Index” gets the location of the called space and presents it as an integer

121|    if Index >= 0:

# Wil not run the program if the index is less than 0 (no grid would a -1 space)

122|        return self.\_\_Grid[Index]

# Returns the value in the position called

123|    else:

124|        raise IndexError()

# If the index is negative an error would be called to indicate that the value entered is out of bounds.

## CheckForMatchWithPattern()

### General Function

Checks if there is a viable pattern made with the symbols entered.

### Line By Line

126|    def CheckforMatchWithPattern(self, Row, Column):

# Function creation

127|        for StartRow in range(Row + 2, Row - 1, -1):

# Loop that counts down from the inputted row + 2 down to the inputted row – 1 in increments of -1

128|            for StartColumn in range(Column - 2, Column + 1):

# Loop that counts from the inputted column - 2 up to the inputted column + 1 in increments of 1.

129|                try:

# exception handling is used in case there is no corresponding cell in the altered column numbers (e.g row + 2 and column -2)

130|                    PatternString = ""

# A variable string is created to store the symbols.

131|                    PatternString += self.\_\_GetCell(StartRow, StartColumn).GetSymbol()

132|                    PatternString += self.\_\_GetCell(StartRow, StartColumn + 1).GetSymbol()

133|                    PatternString += self.\_\_GetCell(StartRow, StartColumn + 2).GetSymbol()

134|                    PatternString += self.\_\_GetCell(StartRow - 1, StartColumn + 2).GetSymbol()

135|                    PatternString += self.\_\_GetCell(StartRow - 2, StartColumn + 2).GetSymbol()

136|                    PatternString += self.\_\_GetCell(StartRow - 2, StartColumn + 1).GetSymbol()

137|                    PatternString += self.\_\_GetCell(StartRow - 2, StartColumn).GetSymbol()

138|                    PatternString += self.\_\_GetCell(StartRow - 1, StartColumn).GetSymbol()

139|                    PatternString += self.\_\_GetCell(StartRow - 1, StartColumn + 1).GetSymbol()

# (131 - 139) Getting the symbols present in the 3 by 3 grid marked by the function, this then stored the symbols in the PatternString variable such that it can be checked.

140|                    for P in self.\_\_AllowedPatterns:

# Loops through all the allowed patterns that were assigned.

141|                        CurrentSymbol = self.\_\_GetCell(Row, Column).GetSymbol()

# Gets the symbol used in the starting cell and assigns it to a CurrentSymbol variable.

142|                        if P.MatchesPattern(PatternString, CurrentSymbol):

# This then checks if the pattern exists in the grid using the MatchesPattern() function.

143|                            self.\_\_GetCell(StartRow, StartColumn).AddToNotAllowedSymbols(CurrentSymbol)

144|                            self.\_\_GetCell(StartRow, StartColumn + 1).AddToNotAllowedSymbols(CurrentSymbol)

145|                            self.\_\_GetCell(StartRow, StartColumn + 2).AddToNotAllowedSymbols(CurrentSymbol)

146|                            self.\_\_GetCell(StartRow - 1, StartColumn + 2).AddToNotAllowedSymbols(CurrentSymbol)

147|                            self.\_\_GetCell(StartRow - 2, StartColumn + 2).AddToNotAllowedSymbols(CurrentSymbol)

148|                            self.\_\_GetCell(StartRow - 2, StartColumn + 1).AddToNotAllowedSymbols(CurrentSymbol)

149|                            self.\_\_GetCell(StartRow - 2, StartColumn).AddToNotAllowedSymbols(CurrentSymbol)

150|                            self.\_\_GetCell(StartRow - 1, StartColumn).AddToNotAllowedSymbols(CurrentSymbol)

151|                            self.\_\_GetCell(StartRow - 1, StartColumn + 1).AddToNotAllowedSymbols(CurrentSymbol)

# (143 - 151) Goes through all the positions in the 3 by 3 grid and disallows the symbol used in the grid to be used again in that 3 by 3 section

152|                            return 10

# Then the function is to return the integer 10 to show how many points are to be added to the score

153|                except:

154|                    pass

155|        return 0

# Then the function is to return the integer 0 to show how many points are to be added to the score

## \_\_GetSymbolFromUser

### General Function

This is an input request for the user that will keep looping until a valid symbol is entered (one that is in the \_\_AllowedSymbols variable).

### Line By Line

157|    def \_\_GetSymbolFromUser(self):

# Function creation

158|        Symbol = ""

# Empty string variable initialisation so that 159 has something to check against

159|        while not Symbol in self.\_\_AllowedSymbols:

# Indefinite loop until the value in the variable Symbol matches a value in \_\_AllowedSymbols

160|            Symbol = input("Enter symbol: ")

# User input

161|        return Symbol

# If the input matches a symbol in the \_\_AllowedSymbols variable, the symbol is returned

## \_\_CreateHorizontalLine

### General Function

Self-explanatory, creates a horizontal line.

### Line By Line

163|    def \_\_CreateHorizontalLine(self):

# Function creation

164|        Line = "  "

# Empty string variable initialisation so that 166 has something to add to

165|        for Count in range(1, self.\_\_GridSize \* 2 + 2):

166|            Line = Line + "-"

# Adds a “-” to the Line variable until the loop ends.

167|        return Line

# Returns the completed line.

## Display Puzzle

### General Function

Creates the visual aspect associated with the puzzle such that the user has something to work with.

### Line By Line

169|def DisplayPuzzle(self):

# Function Creation

170|    print()

# Prints an empty space

171|    if self.\_\_GridSize < 10:

172|        print("  ", end='')

# Prints a space without creating a new line for the next output (end=’’ changes the way that the string ends from a new line [\n] to an empty string [] this way the next thing that is printed will be on the same level as the previous outputted text.)

173|        for Count in range(1, self.\_\_GridSize + 1):

174|            print(" " + str(Count), end='')

# This is what prints the incrementing numbers that are visible on the top side of the grid.

175|    print()

# Prints empty space (mostly to override the end=’’ from the previous line)

176|    print(self.\_\_CreateHorizontalLine())

# Creates a horizontal line to separate the top numbers from the grid

177|    for Count in range(0, len(self.\_\_Grid)):

178|        if Count % self.\_\_GridSize == 0 and self.\_\_GridSize < 10:

# Checks to see if the counter is divisible by the grid size and if the grid size is less than 10.

179|            print(str(self.\_\_GridSize - ((Count + 1) // self.\_\_GridSize)) + " ", end='')

# Prints the number decrementing numbers visible of the side of the grid.

180|        print("|" + self.\_\_Grid[Count].GetSymbol(), end='')

# Prints a vertical line followed by whatever symbol was present in that grid position

181|        if (Count + 1) % self.\_\_GridSize == 0:

182|            print("|")

# If the value after the count is divisible by the grid size, then print a vertical line, this marks the end of the column

183|            print(self.\_\_CreateHorizontalLine())

# Then print a horizontal line below

# Pattern():

## General Function of the Class

Everything related to the management of the patterns that the symbols could take.

## \_\_init\_\_

### General Function

Initialises the pattern class.

### Line By Line

186|def \_\_init\_\_(self, SymbolToUse, PatternString):

# Function creation

187|    self.\_\_Symbol = SymbolToUse

# Stores the SymbolToUse argument into the \_\_Symbol variable.

188|    self.\_\_PatternSequence = PatternString

# Stores the PatternString argument into the \_\_PatternSequence variable.

## MatchesPattern

### General Function

Goes through a pattern string and checks to see if it matches the pattern assigned to the symbol used.

### Line By Line

190|def MatchesPattern(self, PatternString, SymbolPlaced):

# Function creation

191|    if SymbolPlaced != self.\_\_Symbol:

192|        return False

# If the SymbolPlaced argument does not match the Symbol variable, then the variable will instantly return False

193|    for Count in range(0, len(self.\_\_PatternSequence)):

# Otherwise it will create a definite loop for each symbol in the \_\_PatternSequence (This is the one that is matched to the Symbol)

194|        try:

# Exception handling in case there is an error regarding the length of each pattern

195|            if self.\_\_PatternSequence[Count] == self.\_\_Symbol and PatternString[Count] != self.\_\_Symbol:

196|                return False

# If the patterns do not match each other at any point the function returns False

197|        except Exception as ex:

198|            print(f"EXCEPTION in MatchesPattern: {ex}")

199|    return True

# If the function does not return anything at this point that means the patterns must be matching.

## GetPatternSequence

### General Function

Getter function for the \_\_PatternSequence private variable.

### Line By Line

201|def GetPatternSequence(self):

# Function creation

202| return self.\_\_PatternSequence

# Returns the value in the \_\_PatternSequence variable

# Cell():

## General Function of the Class

Everything related to the management of the cells of the grid.

## \_\_init\_\_

### General Function

Initialises the Cell class.

### Line By Line

205|def \_\_init\_\_(self):

# Function creation

206|    self.\_Symbol = ""

# Empty string variable named \_\_Symbol

207|    self.\_\_SymbolsNotAllowed = []

# Empty list named \_\_SymbolsNotAllowed.

## GetSymbol

### General Function

Returns the symbol if it exists otherwise returns a ‘-’.

### Line By Line

209|def GetSymbol(self):

# Function creation

210|    if self.IsEmpty():

# Calls the IsEmpty function and applies it to a selection.

211|        return "-"

# If IsEmpty returns True, a ‘-’ is returned

212|    else:

213|      return self.\_Symbol

# If IsEmpty returns False, the variable \_\_Symbol is returned.

## IsEmpty

### General Function

Checks to see if the contents of \_\_Symbol are empty or not.

### Line By Line

215|def IsEmpty(self):

# Function creation

216|    if len(self.\_Symbol) == 0:

217|        return True

# If the length of the \_\_Symbol variable is equal to 0, then return True

218|    else:

219|    return False

# Otherwise return False

## ChangeSymbolInCell

### General Function

Self-explanatory, changes the symbol for that cell.

### Line By Line

221|def ChangeSymbolInCell(self, NewSymbol):

# Function creation

222|    self.\_Symbol = NewSymbol

# Changes the \_\_Symbol variable into the value of the NewSymbol argument.

## CheckSymbolAllowed

### General Function

Checks if the symbol belongs to the \_\_SymbolsNotAllowed list.

### Line By Line

224|def CheckSymbolAllowed(self, SymbolToCheck):

# Function creation

225|    for Item in self.\_\_SymbolsNotAllowed:

# Iterates through every value in the list \_\_SymbolsNotAllowed

226|        if Item == SymbolToCheck:

227|            return False

# If the value in the argument Item matches that of the value in \_\_SymbolsNotAllowed, then the function returns False

228|    return True

# If the iteration is completed without any matches, then the function returns True

## AddToNotAllowedSymbols

### General Function

Self-explanatory, adds a value to the \_\_SymbolsNotAllowed list.

### Line By Line

230|def AddToNotAllowedSymbols(self, SymbolToAdd):

# function creation

231|    self.\_\_SymbolsNotAllowed.append(SymbolToAdd)

# Appends the value in the argument SymbolToAdd into the list \_\_SymbolsNotAllowed

## UpdateCell

### General Function

This function literally does nothing.

### Line By Line

233|def UpdateCell(self):

# Function creation

234|    pass

# Do nothing

# BlockedCell():

## General Function of the Class

Management of blocked cells on the grid. Has Cell() as a parent class.

## \_\_init\_\_

### General Function

Initialise the class.

### Line By Line

237|def \_\_init\_\_(self):

# Function creation

238|    super(BlockedCell, self).\_\_init\_\_()

# Calls its parent class “Cell()” and its initialisation

239|    self.\_Symbol = "@"

# Using the initialisation from the “Cell()” class, assigned “@” to the variable \_\_Symbol.

## CheckSymbolAllowed

### General Function

Returns False, this is because no symbol would be allowed the cell.

### Line By Line

241|def CheckSymbolAllowed(self, SymbolToCheck):

# Function creation

242|    return False

# Returns False

# Lines 244-245:

## General Function

Runs the code when the program is run.

## Line By Line

244|if \_\_name\_\_ == "\_\_main\_\_":

# This condition is normally used to ensure that the indented code is only run if the program is used as a script not a module (so run from the file itself and not borrowing the code), however in the context of this program it does not make much sense to do this as the only code indented is code that can be called as a module.

245|    Main()

# Calls the “Main()” function.

# Potential Questions Regarding the Skeleton Code

## UpdateCell

The [UpdateCell](#_UpdateCell) function does not do anything in the current state of the code and therefore a task may be to create code that works for the function, however there is no clear way of understanding what “updating a cell” could mean given the context of the program. However, it is most likely going to be a way to override a cell that already exists in order to make room for other patterns.

## A grid of arrows with a white background Description automatically generatedRewriting the Pattern Sequence for the Symbols.

Currently the pattern sequence assigned to the symbols takes a very unusual route that is very confusing to understand straight away as an external developer (pictured). Therefore, a potential question could be to rewrite the code in a way that follows a route proposed by the exam paper (one that hopefully makes a lot more sense than the current one).