AARHUS UNIVERSITY

PORTFOLIO – PROGRAMMING FOR THE HUMANITIES

Bertram Højer, 201808398– BA Cognitive Science

Peter Thramkrongart, 201806892– BA Cognitive Science

Jakob Grøhn Damgaard, 201808996 – BA Cognitive Science

December 15th 2019

Semester: Autumn 2019

Teacher: Pantelis M. Papadopoulos

Indholdsfortegnelse

1. Problem Decomposition

1.1 Problem title: The Perfect Bet

Analyst: Bertram Højer

Problem description:

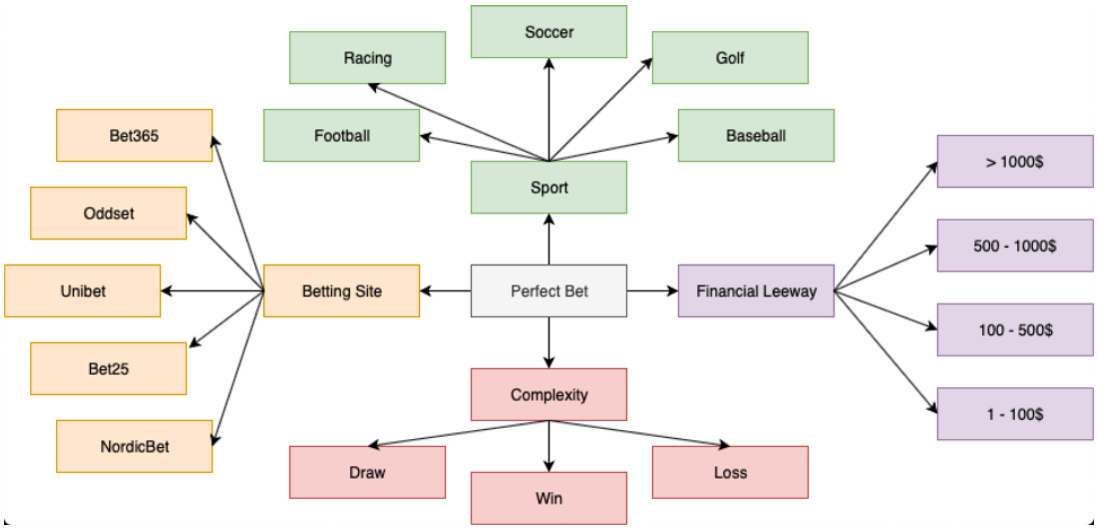
Making the perfect bet can be a very hard thing to do, there is a plethora of variables to account for, and very often one of those variables seem to be pure luck. You can however increase your chances of winning more often than you lose by doing your due diligence.

As mentioned, there are many variables to consider when decomposing the problem, these variables could be e.g. which sport you should be betting on, how complicated you should make your bet, how much money to invest in the bet, potential superstitious beliefs that you have, which betting site has the best odds.

Problem decomposition:

1. Which sport will you bet on:
    a. Soccer b. Football c. Racing d. Golf e. Basketball f. Baseball
2. Complexity of bet:
    a. Single-bet b. Parlay-bet c. Multi-bet
3. Financial leeway:
    a. Over 1000$ b. 500-1000$ c. 100-500$ d. 1-100$
4. Betting-site:
    a. Bet365 b. Oddset c. Unibet d. 888 e. Bet25 f. Nordicbet

Problem mind map:



1.2 Problem title: Designing a Cognitive Experiment

Analyst: Peter Thramkrongart

Problem description:

A cognitive experiment is an experiment that consists of one or more cognitive tasks. A cognitive task is a task designed to measure thought process. Examples include memory tasks, small gambling tasks or pattern recognition tasks. Cognitive experiments strive to simplify human behaviour and isolate cognitive functions. When designing a cognitive experiment, a lot of considerations have to be taken into account. You need to keep the participant interested and engaged. The task should be simple to understand and perform and should be designed to have a fitting level of difficulty.

Problem decomposition:

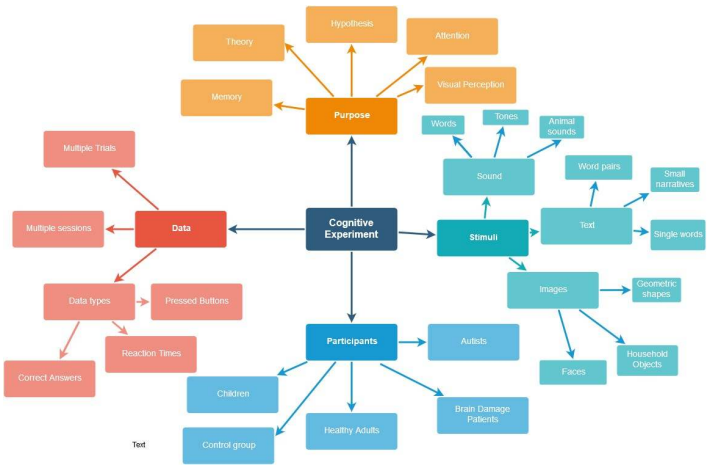
Purpose: The reason to do the experiment. What is the hypothesis? On what cognitive subject is experiment about? This modulates the other subproblems as the design relies on the purpose of the stud

Data: What data points should be recorded?

Stimuli: What type of stimuli should the participant react to?

Participants: Who is the study being conducted on?

Problem mind map[1]:



[1] Larger versions of the mind maps are listed in the appendix

1.3 Problem title: Playing the perfect golf-shot

Analyst: Jakob Grøhn Damgaard

Problem description:

Executing a flawless golf shot is a highly technical and complex task that is deeply dependent on numerous parameters and factors. However, by approaching the problem analytically using Computational Thinking it is possible to decompose the task into sub-problems. These sub-problems can then be assessed individually before being combined in order to complete the grand goal of the shot. Many of these sub-problems must be analysed logically and, hence, Computational Thinking applies perfectly for the assessment of this problem.

Problem decomposition:

Numerous parameters are critical to asses before attempting to carry out the golf shot:

- Distance
    - What is the distance to the target?
    - Which club should I use?
- Motor skills and body control
    - When physically executing the swing, it is critical to:
        - Maintain a steady balance throughout the body
        - Coordinate movements of multiple areas of the body
        - To commit to a certain swing-speed
        - Should I close or open the head of the club to affect shape of the ball flight?
- Weather
    - What is the temperature? Cold air equals more resistance
    - How strong is the wind and which direction is it blowing from?
    - Is it raining? Rain and wet grass equal more resistance
    - What is the air humidity? Higher humidity decreases ball flight length
- Physical geography and landscape
    - Is the target elevated compared to the current ball position?
    - How is the lie of my ball? Fairway, thick rough or perhaps in a bunker?

- Are the curvatures on the green going to affect the direction of my shot upon landing?
- Are there any obstacles (e.g. trees or water hazards) that I should be more cautious to not hit?

Problem mind map:

2. Algorithm Design

2.1 Problem title: The Perfect Bet

Analyst: Bertram Højer

Problem description: Continued from Assignment 1.1 (The Perfect Bet)

Problem algorithm:

1. Specify how much financial leeway you have:
    a. Over 1000$, 500-1000$, 100-500$, 1-100$
2. Specify which sport you wish to bet on:
    a. Soccer, Football, Racing, Golf, Basketball, Baseball
3. Specify which team you wish to bet on:
    a. Team 1, Team 2, Team 3, Team 4
4. Investigate the return of different kinds of bets:
    a. A win to your team, a draw, a loss to your team, a specific result
5. Investigate the returns for the defined team and sport for different bets.
6. Choose the betting-site that has the highest return for any result below 4.00 (otherwise too improbable).
7. Place the bet with specified amount of money.

2.2 Problem title: Writing an algorithm for a university assignment

Analyst: Peter Thramkrongart

Problem Description:

It is time to do your homework for the MOOC-part of your university course on python programming and computational thinking. This week is about algorithms. Algorithms can be thought of as executable recipes or instructions. This week's assignment is to write a simple algorithm for a problem that we each find interesting. Algorithms are executed in steps with optional sub-steps and repeated loops. The assignment itself is made in template and includes, a title, a short description and the actual algorithm.

Algorithm Design:

1. Turn on your computer
2. Read this weeks material on algorithms and make sure you understand it
    a. Read material on watch video on colMOOC about algorithms
    b. Read the papers about search algorithms from colMOOC
3. Open the assignment template from colMOOC
4. Decide on problem for the algorithm to solve the (problem should be simple enough to solve by a 10-20 step algorithm and simple enough to understand in under a 100 words).
5. Decompose the problem into subproblems
    a. Make a mind map if needed
6. Give your algorithm an interesting title
7. Write a description for your problem
8. Write your algorithm in steps and substeps
9. Make sure you have fulfilled the assignment requirements
10. Fix problems if needed
11. Convert to PDF
12. Submit on colMOOC

2.3 Problem title: How to play a game of Tic-Tac-Toe against a computer

Analyst: Jakob Grøhn Damgaard

Problem description:

Beloved for its simplicity and universal accessibility, the good old game of Tic-Tac-Toe is enjoyed around the globe! It is a fun and swift brain-stimulator that can be played across multiple age levels and that demands no more than a piece of scrap paper, a pen and two competitors. Furthermore, algorithmically describing the structure of this banal game can serve as a perfect illustration of Computational Thinking. The game flows in a logic structure and revolves around solving a task by observing and detecting patterns and generalizing game-strategies based on these. For the sake of simplicity, my example will only assess the structure of a game of Tic-Tac-Toe between a self-thinking human being and a computer that plays randomly. By adding an additional abstraction layer – game strategies and tactics – one could further elaborate on how CT can be applied to the game.

Problem algorithm:

Following algorithm description depicts a game-algorithm where a human competitor plays against a computer that places symbols on random free spaces:

1. Draw an empty 3x3 playing board
2. Decide who initiates the games
    a. If it is your turn, decide on an empty space and draw an 'O' in it
    b. Else, the computer will play its turn and draw and 'X' in an empty space
3. Asses the updated playing board
    a. If three of identical symbols are aligned in a row, last player who made a move has won! Proceed to step 5
    b. Else, if there are no more empty spaces, it is a tied game! Proceed to step 5
    c. Else, go back to step 3
4. The game is over! Do you wish to play again?
    a. If yes, go back to step 1,
    b. Else, stop playing

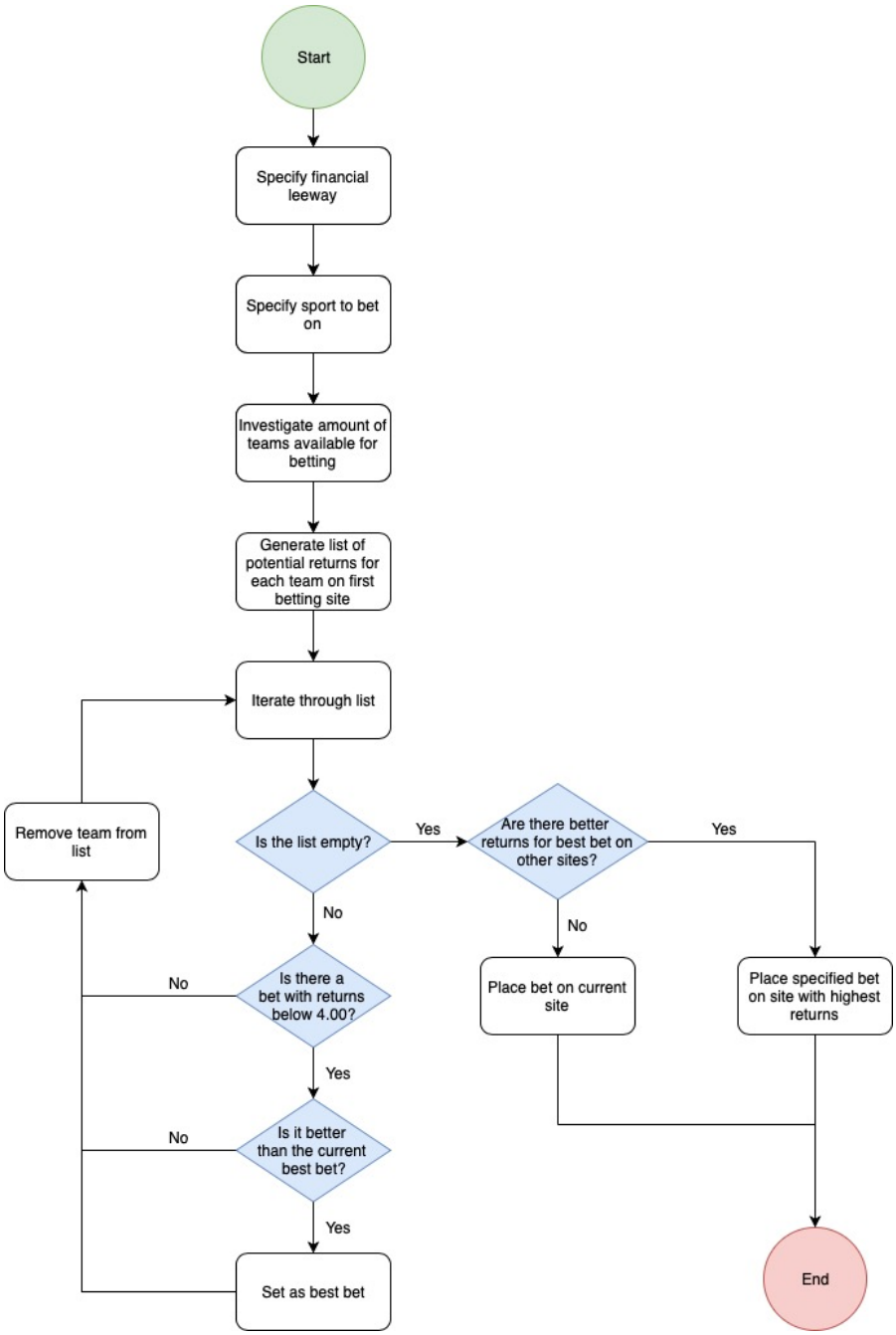
3. Algorithm Flowchart

3.1 Problem title: The Perfect Bet

Analyst: Bertram Højer

Problem algorithm: Continued from assignment 2.1

Algorithm flowchart:



3.2 Problem title: Writing an algorithm for a university assignment

Analyst: Peter Thramkrongart

Problem algorithm: Continued from assignment 2.2

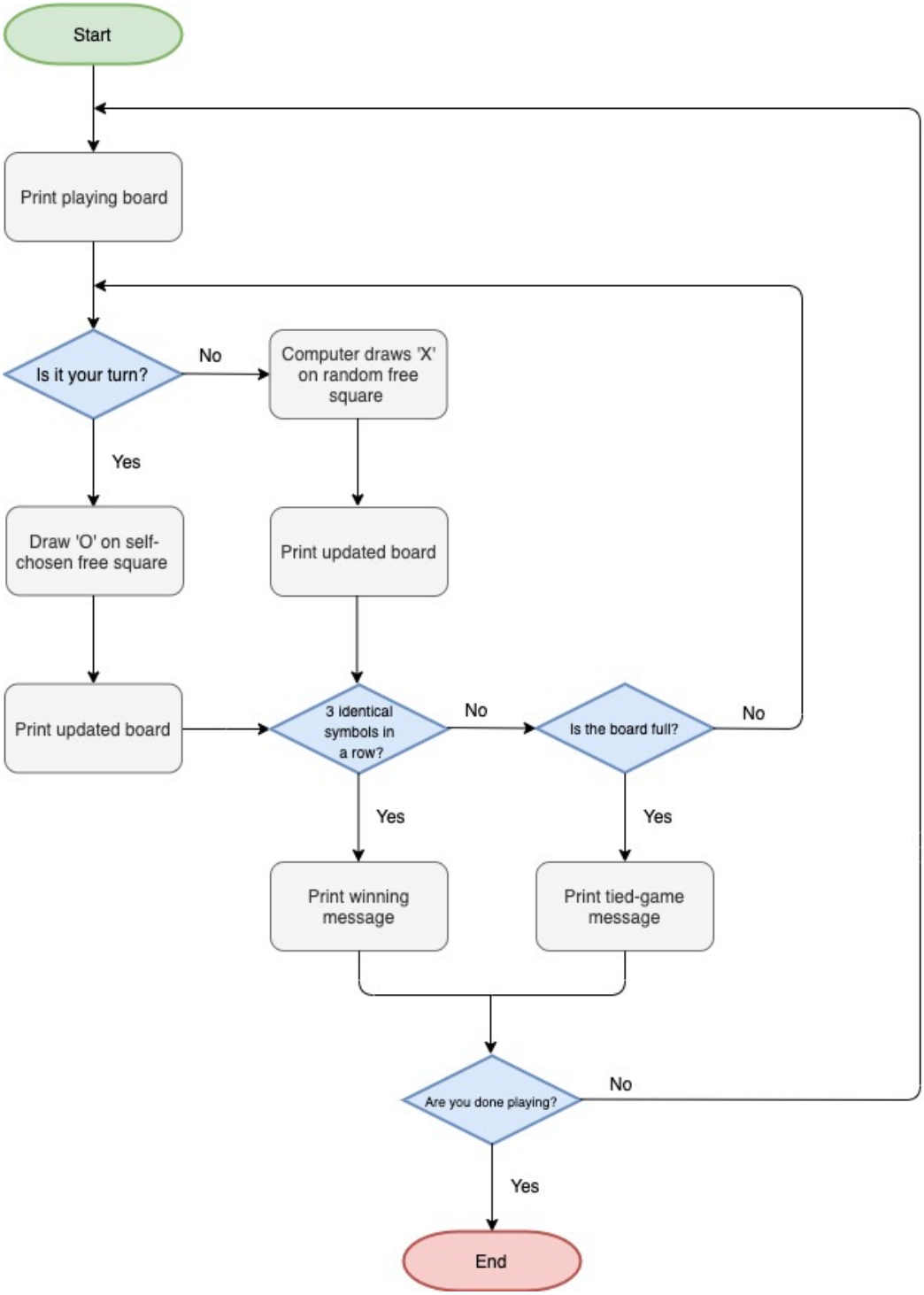
Algorithm flowchart:

3.3 Problem title: How to play a game of Tic-Tac-Toe against a computer

Analyst: Jakob Grøhn Damgaard

Problem algorithm: Continued from assignment 2.3

Algorithm flowchart:



4. Algorithm Pseudocode

4.1 Problem title: The Perfect Bet

Analyst: Bertram Højer

Problem algorithm: Continued from assignment 3.1

Algorithm pseudocode:

```
sports = list of 5 most popular sports
sites = dictionary containing information on bets from different betting-sites
# entails large dataset in dictionary-form potentially gained from webscraping


while condition is True:
    while condition is True:
        bet = user-input (amount of money between 1 – 1000$)
        IF 0 < bet < 1001:
            break

    sport = user-input (which sport to bet on from list[sports])

    bet_list = []

    for i in sites:
        initial_bet = list of return on bets at for specified sport
        bet_list.append()

    for bet in bet_list:
        IF bet > 4.00:
            remove bet from list
```

```
best_bet = max(bet_list)


IF best_best < 2:
        cont = user-input ("there are no good bets, do you wish to try another sport?")
        IF cont == YES:
                print ("Please choose another sport!")
        Else if cont == NO:
                print ("Better luck next time!")
                break
```

4.2 Problem title: Writing an algorithm for a university assignment

Analyst: Peter Thramkrongart

Problem algorithm: Continued from assignment 3.2

Algorithm pseudocode:

```
Set assignment as "Not delivered"

Set readMaterial to either "Read" or "Not read"

WHILE assignment  != "Delivered":
        Turn on your computer
        IF readMaterial != "Read":
            #Read this week's material on algorithms and make sure you understand it
            Read material on watch video on colMOOC about algorithms
            Read the papers about search algorithms from colMOOC
        Open the assignment template from colMOOC
        Decide on problem for the algorithm to solve the #problem should be simple enough to solve by a 10-20 step algorithm and simple enough to understand in under 100 words
        Decompose the problem
        Set problemComplexity to either "Low", "Medium" or "High"
        IF problemComplexity == "High":
                Make a mind map
        Give your algorithm an interesting title
        Write a description for your problem
```

```
        Write your algorithm in steps and sub-steps

        Make sure you have fulfilled the assignment requirements

        Set assignmentRequirements to either "Fulfilled" or "Not fulfilled"

        IF assignmentRequirements != "Fulfilled":

                Fix problems

        Convert to PDF

        Submit on colMOOC

        assignment = "Delivered"

END algorithm
```

4.3 Problem title: How to play a game of Tic-Tac-Toe against a computer

Analyst: Jakob Grøhn Damgaard

Problem algorithm: Continued from assignment 3.3

Algorithm pseudocode:

Following pseudocode depicts a Tic-Tac-Toe game between a user and a random computer. The user starts:

```
set as PLAYING BOARD a 3x3 board with spaces numbered 1-9

function checkForWin(Argument 1, Argument 2, Argument 3, Argument 4):

        IF Argument 2, Argument 3 AND Argument 4 == Argument 1:

        Return TRUE

function winningCombinations(Argument1):

        IF checkForWin(Argument1, any 3-in-a-row combinations on board) = TRUE

        WIN = TRUE

print message: INSTRUCTION MESSAGE

IF button 'enter' is pressed:

        Remove message: INSTRUCTION MESSAGE

print PLAYING BOARD

WHILE nothing interrupts loop

        HUMAN IS PLAYING = TRUE

        WHILE HUMAN IS PLAYING == TRUE:

                Ask user to choose a space on PLAYING BOARD
```

```
        IF chosen space is empty:
                print message 'O' on PLAYING BOARD
                HUMAN IS PLAYING = FALSE
        ELSE
                print message: 'This place is not available!'
winningCombinations('O')
IF WIN == TRUE:
print message: 'You won!'
        END algorithm
IF no more free spaces on board:
print message: 'Game ends in a tie!'
      END algorithm
PC IS PLAYING = TRUE
WHILE PC IS PLAYING == TRUE:
        Choose random number from range(1,8)
        IF chosen space is empty:
                print message 'X' on PLAYING BOARD
                PC IS PLAYING = FALSE
winningCombinations('X')
IF WIN == TRUE:
print message: 'The computer won!'
      END algorithm
```

5. Algorithm Code

5.1 Problem title: The Perfect Bet

Analyst: Bertram Højer

Problem algorithm:

Continued from assignment 4 with adjustments. To be able to run the script multiple random values have to be initialized so the script is slightly different from the pseudo-code. Furthermore, to properly run the pseudocode one would need e.g. the module beautifulsoup4 in order of scraping data from the web.

Algorithm code:

```
# import the 'random' module to get random numbers
import random

# specify list of different sports that the user will be able to bet on
sports = ("Soccer", "Golf", "Racing", "Football", "Baseball")
# specify type of bet
bet_type = ["win", "loss", "draw"]
# Specify betting sites
bet_sites = ("Bet365", "NordicBet", "Unibet", "Oddset", "Bet25")
# define games variable
games = ("Game 1", "Game 2", "Game 3", "Game 4")

while True:
    # specify money variable asking user for how much oney they wish to bet
    while True:
        money = int(input("How much do you wish to bet? \n Please enter a value between 1 - 1000: "))
        if 0 < money < 1001:
            break

    # print amount of money and ask which sport to bet on
    print(f"You chose to bet {money} dollars \n which sport would you like to bet on?")
    # ask user to choose a sport to bet on
    sport = int(input("1: Soccer \n2: Golf \n3: Racing \n4: Football \n5: Baseball \n Enter number: "))

    # if number is entered transform to variable
    if sport == 1:
        sport = sports[0]
    elif sport == 2:
        sport = sports[1]
    elif sport == 3:
        sport = sports[2]
    elif sport == 4:
        sport = sports[3]
    elif sport == 5:
```

```
        sport = sports[4]

    # print which sport was chosen
    print(f"You chose to throw your money at {sport}! Let me investigate potential bets..")

    # specifying randomly initiated values
    rand_value = random.randint(1,10)
    rand_bet_type = random.randint(1, 3)
    # Print which game to bet on /here using random to generate the game number
    print(f"The best bet is on a {bet_type[(rand_bet_type)-1]} in game {rand_value}! \nI will check the
best betting site..")
    site1, site2 = random.sample(range(1, 5), 2)
    bet_choice = int(input(f"""{bet_sites[(site1)-1]} and {bet_sites[(site2)-1]} have great bets,
    choose either one\n1: {bet_sites[(site1)-1]}\n2: {bet_sites[(site2)-1]}\n"""))

    if bet_choice == 1:
        print(f"Your bet has been placed on game {rand_value} at {bet_sites[(site1)-1]}!")
    else:
        print(f"Your bet has been placed on game {rand_value} at {bet_sites[(site2)-1]}!")

    print("It's the best bet, but you're probably going to lose, don't throw away your money!")

    more_bets = input("Would you like to place another bet? \n 'Y' for yes \n 'N' for no \n -->")

    if more_bets.upper() == "N":
        print("Thanks for using 'The Perfect Bet' generator")
        break
    elif more_bets.upper() == "Y":
        continue
    else:
        more_bets = input("Please type either 'Y' or 'N'")
```

5.2 Problem title: Writing an algorithm for a university assignment

Analyst: Peter Thramkrongart

Problem algorithm: Continued from assignment 4 with adjustments to make it run like a script.

Algorithm code:

```
counter = 0#make counter

#defining main question function
def defaultQuestion(message,question,message1):#three strings
    print(message)#print message
    while counter<1:#define while loop
        answer = input(question )#get input answer
        answer = answer.lower()#make lowercase
        if answer == "yes":# end loop if "yes" is entered
            print(message1)#print message
            break

defaultQuestion("\nTurn on your computer!",#turn on computer
"\nHave you turned on your computer yet?","good")

while counter<1:#New while loop
    answer = input("\nHave you read this week's material?")#question
    answer = answer.lower()#to lowercase
    if answer == "yes":#if yes break loop
        print("\nGood!")#print message
        break
    if answer != "yes":#if not yes
        print("\nWatch the video on colmooc!")#print message
        while counter < 1:#start additional loop
            video = input("\nHave you watched the video now?")#question
            video = video.lower()#lower case
            if video == "yes":#if yes print message and start additional loop
                print("\nGood! Now, read the paper!")
                while counter<1:# if yes max counter= end loop
                    read = input("\nHave you you read it now?")#question
                    read = read.lower()# to lower case
                    if read == "yes":
                        print("\ngood!")
                        counter = 1

counter=0 #reset counter

defaultQuestion("\nOpen the assignment template",#open template
"\nHave you opened it?", "good!")

defaultQuestion('''\nNow it's time to decide on a problem
for the algorithm to solve...
\nThe problem should be simple enough to solve by a 10-20 step algorithm
 and simple enough to under in under a 100 words\n''',#Decide on problem
"Have you found your problem yet? ",
"\ngood!")

while counter<1:#while loop
    answer = input('''\nNow it's time to decompose the problem into solvable parts
    \nDo you think you need to make a mind map? write "yes" if you do and
     "no" if you don't:''')#question
```

```
    answer = answer.lower()# to lowercase
    if answer == "no":#if no print message and break loop
        print("\nOK!")
        break
    while counter < 1:#additional while loop
        mindMap = input("\nHave you made your mind map?")#question
        mindMap = mindMap.lower()#to lower
        if mindMap == "yes":#if yes print message and max counter = break loop
            print("\nGreat!")
            counter = 1
            break

counter = 0  #reset counter

defaultQuestion("\nNow it's time to give your algorithm an interesting title!",
"\nHave you come up with a title yet? ",#give title
"\nCool")

title = input("\nWrite your title here:")#what is your title

print("\nYour title is" +" "+'"'+ title +'"')#print title

defaultQuestion("\nWrite a description for your problem",
"\nHave you written it yet?",#decription
"\nGood!")

defaultQuestion("\nWrite a your algorithm in steps and substeps",
"\nHave you written it yet?",#write algorithm
"\nGreat!")

defaultQuestion('''\nNow it's to check for errors and make sure you
 are fulfilling the assignment requirements''',#error check
"\nAre you done checking?",
"\nGreat!")

while counter < 1:#while loop
    answer = input("\nDid you have any problems?")#question
    answer = answer.lower()#lower
    if answer == "yes":#if yes: print message and start additional loop
        print("\nOkay, fix them, then")
        while counter < 1:#while loop
            fix = input("\nHave you fixed the problems?")#question
            fix = fix.lower()#lower
            if fix == "yes":#if yes: print message, max counter = break loop
                print("\nGreat!")
                counter = 1
    if answer == "no":#if no break loop
        break

counter = 0#reset counter

defaultQuestion("\nNow convert it to a PDF",
"\nHave you converted it yet?",#convert to PDF
"\nGood!")

defaultQuestion('''\nNow you are about done and it's time
 to submit your work to colMOOC''',#submit to MOOC
"\nHave you submitted your work yet?",
"\nGreat! Now you are done!")
```

5.3 Problem title: How to play a game of Tic-Tac-Toe against a computer

Analyst: Jakob Grøhn Damgaard

Problem algorithm: Continued from assignment 4.3

Algorithm code:

Following code initialises a game of Tic-Tac-Toe between the user and computer that places symbols on random free spaces. The script randomly chooses who starts:

```
# Importing necessary libraries
import random
import time

# Defining framework playing board
def showBoard(board):
    print("_____________")
    print('|', board[0], '|', board[1], '|', board[2], '|')
    print("-------------")
    print('|', board[3], '|', board[4], '|', board[5], '|')
    print("-------------")
    print('|', board[6], '|', board[7], '|', board[8], '|')
    print("¯¯¯¯¯¯¯¯¯¯¯¯¯")

# Function that checks whether three arguments are identical
def checkForWin(symbol, one, two, three):
    if playingBoard[one] == symbol and playingBoard[two] == symbol and playingBoard[three] ==
symbol:
        return True

# Setting win to false
win = False

# Functions that runs checkForWin-function on all 3-in-a-row combinations on the board
def winningCombinations(symbol):
    global win
    if checkForWin(symbol, 0, 1, 2): # 1st row
        win = True
    if checkForWin(symbol, 3, 4, 5): # 2nd row
        win = True
    if checkForWin(symbol, 6, 7, 8): # 3rd row
        win = True
```

```
    if checkForWin(symbol, 0, 3, 6): # 1st column
        win = True
    if checkForWin(symbol, 1, 4, 7): # 2nd column
        win = True
    if checkForWin(symbol, 2, 5, 8): # 3rd column
        win = True
    if checkForWin(symbol, 0, 4, 8): # Diagonal L-R
        win = True
    if checkForWin(symbol, 2, 4, 6): # Diagonal R-L
        win = True

# Array of numbers for instruction board
instructionBoard = ["1","2","3","4","5","6","7","8","9"]
# Array of spaces for empty board
playingBoard = [" "," "," "," "," "," "," "," "," "]
# Defining a list of numbers from 0-8
numbers = [0,1,2,3,4,5,6,7,8]

# Define game-loop where human user starts
# While-loop runs until game is won or tied
def userStarts():
    while True:


        humanPlay = True
        # Human is playing
        while humanPlay == True:
            # Ask human to choose a square
            userInput = int(input('Select a space by typing the corresponding number and
             press enter: '))
            # If square is empty, draw an 'O' in it
            if playingBoard[userInput-1] != 'O' and playingBoard[userInput-1] != 'X':
                playingBoard[userInput-1] = 'O'
                #Updatae playing board
                showBoard(playingBoard)
                # Human is finished playing
                humanPlay = False
            # If square is NOT empty, try a again
            else:
                print('This place is not available!')

        # Check it human has won (are there 3 O's in a row?)
        winningCombinations('O')
        if win == True:
            # If yes, print winning message and end algorithm
            print('Congratulations! You won!')
            break

        count = 0
```

```
        # Count how many square are NOT empty
        for i in numbers:
            if playingBoard[i] == 'O' or playingBoard[i] == 'X':
                count += 1

        # If all squares are full, the game is a tie
        if count == 9:
            time.sleep(1)
            print('No more free spaces! The game is a tie!')
            break

        # Computer's turn to play
        print('Computer is playing!')
        # Time delay to give a more genuine feel
        time.sleep(1)

        pcPlay = True

        while pcPlay == True:
            # Computer chooses random square
            pcInput = random.randint(0,8)

            # If square is empty, draw an 'X' in it - if square is occupied it automatically
            chooses a new square
            if playingBoard[pcInput-1] != 'X' and playingBoard[pcInput-1] != 'O':
                playingBoard[pcInput-1] = 'X'
                showBoard(playingBoard)
                pcPlay = False

        # Check if computer has won
        winningCombinations('X')
        if win == True:
            # If yes, end algorithm, if not, it loops through main loop again from the top
            print('Shit! The computer won!')
            break


# Define game-loop where computer user starts
# While-loop runs until game is won or tied
def compStarts():
    while True:

        # Computer's turn to play
        print('Computer is playing!')
        # Time delay to give a more genuine feel
        time.sleep(1)

        pcPlay = True

        while pcPlay == True:
            # Computer chooses random square
```

```
            pcInput = random.randint(0,8)

            # If square is empty, draw an 'X' in it - if square is occupied it automatically
chooses a new square
            if playingBoard[pcInput-1] != 'X' and playingBoard[pcInput-1] != 'O':
                playingBoard[pcInput-1] = 'X'
                showBoard(playingBoard)
                pcPlay = False

        # Check if computer has won
        winningCombinations('X')
        if win == True:
            # If yes, end algorithm, if not, it loops through main loop again from the top
            print('Shit! The computer won!')
            break

        count = 0

        # Count how many square are NOT empty
        for i in numbers:
            if playingBoard[i] == 'O' or playingBoard[i] == 'X':
                count += 1

        # If all squares are full, the game is a tie
        if count == 9:
            time.sleep(1)
            print('No more free spaces! The game is a tie!')
            break


        humanPlay = True
        # Human is playing
        while humanPlay == True:
            # Ask human to choose a square
            userInput = int(input('Select a space by typing the corresponding number and
            press enter: '))
            # If square is empty, draw an 'O' in it
            if playingBoard[userInput-1] != 'O' and playingBoard[userInput-1] != 'X':
                playingBoard[userInput-1] = 'O'
                #Updatae playing board
                showBoard(playingBoard)
                # Human is finished playing
                humanPlay = False
            # If square is NOT empty, try a again
            else:
                print('This place is not available!')

        # Check it human has won (are there 3 O's in a row?)
        winningCombinations('O')
        if win == True:
            # If yes, print winning message and end algorithm
```

```
            print('Congratulations! You won!')
            break


# ACTUAL GAME STARTS HERE #

# Printing instruction board and instruction text so participant knows how to play
print('Welocome to a game of Tic-Tac-Toe!')
showBoard(instructionBoard) # Using show
instruction = input('The spaces on the playing board have the same numbers as those seen in
                     the board above \n' +
                    'Use theese numbers when making your moves \n' +
                    'Press "enter" when you are ready to proceed to the game ')


# Displey emptuy playing board
time.sleep(1)
showBoard(playingBoard)

# Random generate either 0 or 1
whoStarts = random.randint(0,1)

# If 0 is generated, run script where user starts
if whoStarts == 0:
    userStarts()
# If 1 is generated, run script where computer startss
else:
    compStarts()
```

Following screenshots show the terminal-outputs of the code during a game:

```
Welocome to a game of Tic-Tac-Toe!
 _____________
| 1 | 2 | 3 |
 _____________
| 4 | 5 | 6 |
 _____________
| 7 | 8 | 9 |

The spaces on the playing board have the same numbers as those seen in the board above
Use theese numbers when making your moves
Press "enter" when you are ready to proceed to the game
```

```
 _____________
|   |   |   |
 _____________
|   |   |   |
 _____________
|   |   |   |

Select a space by typing the corresponding number and press enter: 9
```

```
 _____
|   |   |   |
 -----------
|   |   |   |
 -----------
|   |   | O |
 -----
Computer is playing!
 _____
| X |   |   |
 -----------
|   |   |   |
 -----------
|   |   | O |
 -----
Select a space by typing the corresponding number and press enter:
```

```
Select a space by typing the corresponding number and press enter: 5
 _____
| X |   | O |
 -----------
|   | O | X |
 -----------
|   |   | O |
 -----
Computer is playing!
 _____
| X |   | O |
 -----------
| X | O | X |
 -----------
|   |   | O |
 -----
Select a space by typing the corresponding number and press enter: 7
 _____
| X |   | O |
 -----------
| X | O | X |
 -----------
| O |   | O |
 -----
Congratulations! You won!
```

6. Group Project: Good shroom or doom shroom?

6.1 Project Description

The DoomShroom-project uses machine learning to tackle the world-spanning problem of classifying edible mushrooms apart from poisonous mushrooms. Danish forests floors are loaded with beautiful mushrooms. Many are delicious, tasty treats, others are mischievous death traps. How does the average mushroom forager confidently tell these apart? With the image recognition app DoomShroom! A simple computer app that can classify the mostly delicious boletus species as either edible or deadly from a single image! Using the TensorFlow framework, DoomShroom utilizes a binary neural network classifier* and a tkinter framework to provide a simple and intuitive GUI to provide the user with a comfortable and satisfying classifier experience. Note, that the following project documentation report only assesses the user interface script. For further insight into pre-processing and model-training scripts, see appendix.

* We take no legal responsibility for your possible death from using this app. Do not let your life depend on a single (but awesome) exam project. The classifier is imprecise so do not rely on it.

6.2 Project Requirements

The system consists of a graphical user interface that implements a pretrained convolutional neural network, also coded in Python using the TensorFlow library. The purpose of the system is to enable the user to classify whether a self-uploaded image displays an edible or a poisonous boletus mushroom. The user's interaction with the program is purposefully kept rather simple and the interface only contains 3 different pages. These pages have between 2-3 buttons which are used to navigate through the interface or to choose, upload and classify an image. The below table provides a structured overview of the interface architecture and the built-in functions:

Program Functions

Start Page	
The start page is, as the words indicates, the starting page of the app. When the GUI is initiated the user will land on this page allowing them to navigate to either the information page or the model page through buttons.	
Identify Image	This is a button which navigates to the model page
More Information	This is a button which navigates to the information page
Model Page	
The model page is accessed through the button labelled 'Classify Image'. This page initially contains a frame in the middle of the page and 3 buttons below the frame. These buttons enable the user to go back to the start page or to upload and classify an image. When an image has been classified, a reset button appears which, when pressed, resets the page to its initial state.	
Choose Image	This is a button which opens a folder-window allowing the user to navigate their computer and find an image in JPG-format that they wish to display and analyse using the neural network.
Identify	This is a button which allows the user to identify whether their shroom is edible or poisonous by running the chosen image through neural network model. The classification prediction will display below the chosen image. Using identify will replace 'Choose Image' by a button labelled 'Reset' and remove the button labelled 'Identify'.
Reset	This is a button which allows the user to reset the model page, removing the image and the prediction made by the model. Clicking the reset button will reset the entire page allowing the user to upload a new image for classification.
Start Page	This is the button which allows the user to navigate back to the start page from either the model page or the information page and resets the page to allow the upload of a new image.

Information Page	
The information page is accessed through the button on the start page labelled ‘More information’. The page displays short paragraphs of information on edible as well as poisonous boletaceea. It also shows two sample images which the user can use as references to gain further confidence in the prediction made by the neural network.	
Infobox 1	Frame that displays a short, Danish text describing which kinds of poisonous mushrooms there are in the Danish nature of the kind boletaceae and a picture of the poisonous *rubroboletus satanas.*
Infobox 2	Frame that displays a short, Danish text describing a normal boletaceae as well as a short note on their edibility. Also contains a picture of the exquisite species *boletus edulis*.
Start Page	This is the button which allows the user to navigate back to the start page from either the model-page or the info-page.

6.3 Installation Notes

The program was written in Python 3.7 and all scripts should be performed on this version. The provided folder contains three different scripts as well as a pre-trained model, test images and images needed for the graphical user-interface. The user is only encouraged to interact with the ‘doomShroomGUI.py’ script. The pre-processing script and the modelling script are, however, included in the folder for the sake transparency. These are respectively called ‘DoomShroomPre.py’ and ‘DoomShroomModel.py’. The model-training script is also possible to run, however, it does not engage the user in any way nor produce any understandable/interesting outputs. Beware that the model-training script loops over multiple convolutional neural network models with varying structure to determine the one with optimal classification precision and, thus, the script takes a long time to finish training. The pre-processing script uses the OS package. Therefore, we have excluded the raw training data, so it is not possible to run this script. Do however feel free to examine the code. To run the scripts multiple packages are necessary to have installed:

Packages	
TensorFlow Used in: Model-training and interface scripts	Used for designing, testing and extracting/importing the neural network algorithm
opencv Used in: All scripts	Used for importing and easily modifying images
Tkinter Used in: Infterface script	Used for developing the graphical user-interface
PIL (Used in: Infterface script)	Used for easy manipulation and import of images in consort with tkinter
numpy Used in: Pre-processing and interface scripts	Used for manipulation of arrays to reshape images and flatten data
pathlib Used in: Infterface script	Used for getting current directory and printing it to the terminal when running the script.
matplotlib Used in: Pre-processing script	Used in order to display image example in preprocessing script
tqdm Used in: Pre-processing script	Used to iterate over multiple files in a directory in the preprocessing script
os Used in: Pre-processing script	Used to specify filepath **ONLY IN PREPROCESSING - NOT USED IN THE INTERFACE SCRIPT**
pickle Used in: Pre-processing script	Used for creating and saving feature and label arrays during preprocessing
random Used in: Pre-processing script	Used for shuffling the training data
time Used in: Model-training script	Used for naming the model based on the time it was run.

6.4 User Manual

The DoomShroom Classifier interface aspires to provide smooth and simple user-interaction. You run the doomShroomGUI.py file by using the provided folder as the directory. The simply designed GUI will pop up and the user is free to do as he/she pleases. The user will initially land on the start page:

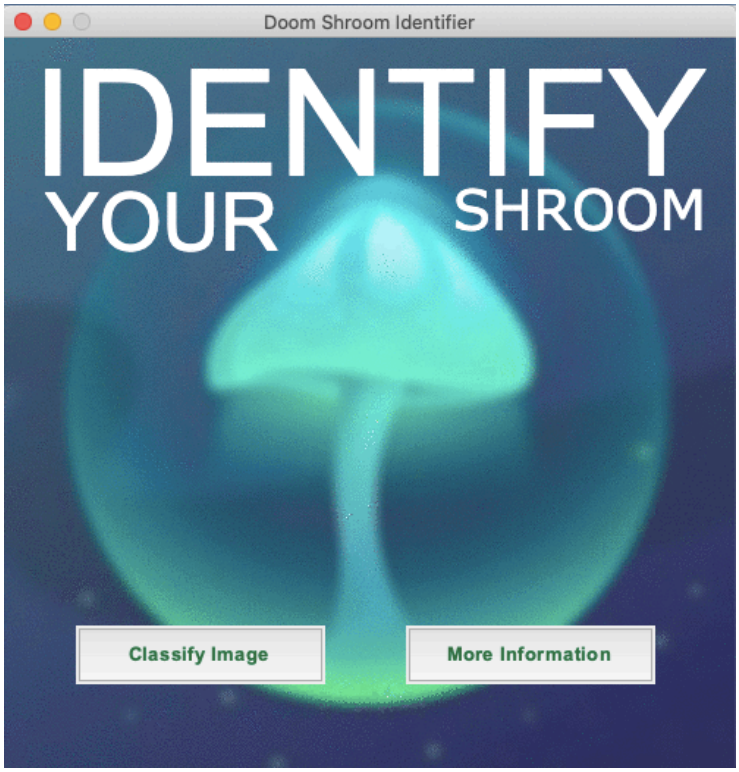


Figure 1 - Front page of GUI

We recommend navigating to the information page by pressing the "More Information" button. Here the user will be presented with some general information about edible and poisonous boletecea in Denmark. The user will also be presented with two example images. One image of the deadly *rubroletus satanas* and one image of the appetizing and common-known *boletus edulis*:

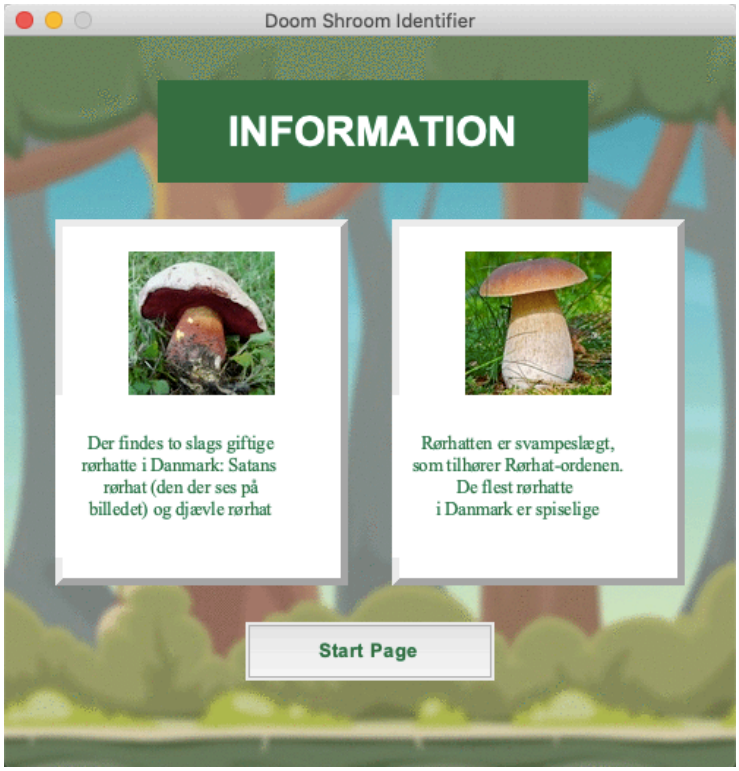


Figure 2 - Information page of GUI

From the 'Information Page' the user is able to return back to the start page by pressing the "Start Page" button. From the start page the user can also access the classification page by pressing the "Classify Image" button:

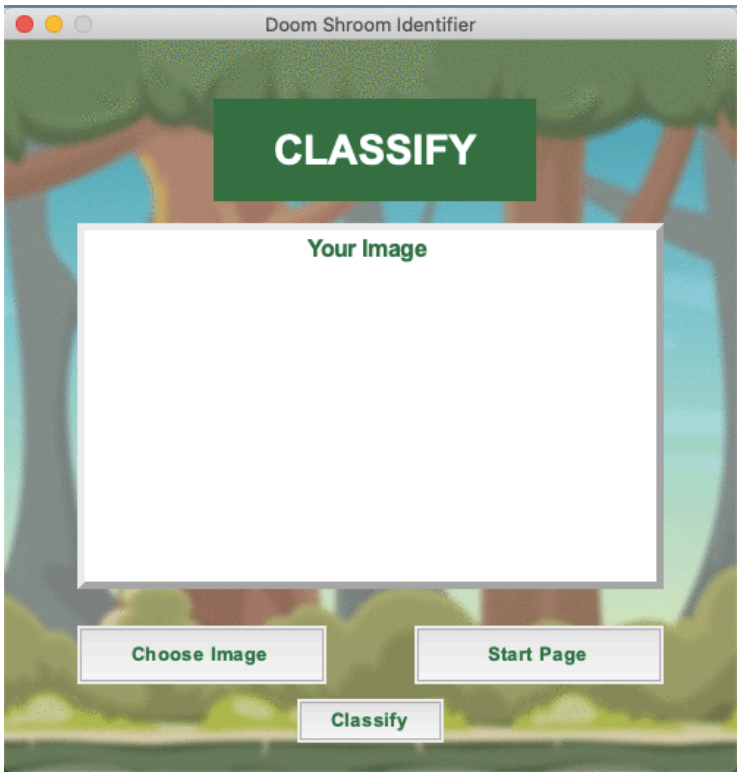


Figure 3 - Model page before uploading image

On the classification page the user can return back to the start page by pressing the “Start Page” button, or the user can choose an image to classify by pressing the “Choose Image” button. By pressing this button, the user is able to upload a JPG image from his/her computer to classify. Any JPG is usable, but we recommend starting off by using one of the two *test images* provided in the directory folder. When an image is selected it will be displayed in a 150x150 pixel format in the centre of the interface. The user can then proceed to classify the chosen image by clicking the grey “Classify” button. This will run the image through the neural network model, which will calculate an output. The user will then be provided with a classification and an attached probability estimate below the image:

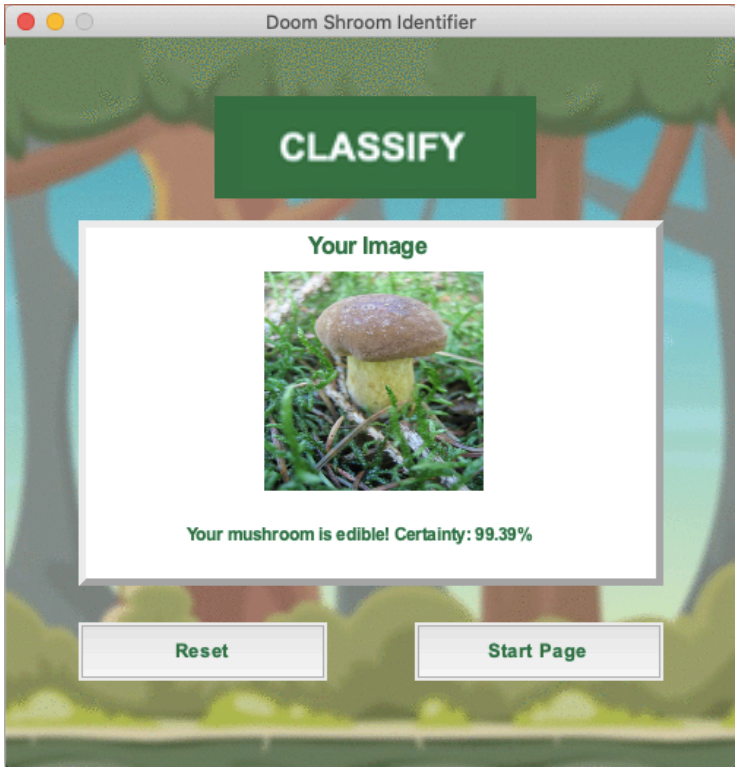


Figure 4 - Model page after uploading and classifying an image

After classifying the user can either return back to the start page by pressing the “Start Page” button or reset the whole page by pressing the “Reset” button, allowing the user to upload a new image for classification. This can be repeated until the user has no more mushrooms he/she wishes to classify. When the user reaches that point, we recommend trying out the algorithm on images of your friends and family to see which mushrooms they are according to the algorithm - this is great fun!

Enjoy your mushroom classification extravaganza!

Once again note, that the folder and project encompass three distinct python scripts, but the manual is only intended for the 'doomShroomGUI.py script'. The other scripts are accessible but merely show how the data was pre-processed and, furthermore, how the TensorFlow neural network model was defined. They are added to increase the transparency of the project. The pre-processing script can and should not be run, as it implements the 'os' package. The appendix contains more information about the neural network algorithm and how neural networks with differently defined parameters perform on the image-database used. Assessing the appendix might be more beneficial than running the actual script.

6.5 Code Index

Following table explains the most important variables in the script and what they store:

Variables	
Tkinter specification variables	
app	tkinter variable specified to show the interface when running the script
chooseImg	Button that takes commands addImage() and openImage()
classifyButton	Button that takes commands preprocces(), runModel(), predCalculation() and reset()
infoButton	Button that takes command showFrame() to navigate to InfoPage
modelButton	Button that takes command showFrame() to navigate ModelPage
panel	Global label that holds the user-chosen image
predictionLabel	Label that displays edibility and probability
resetButton	Button that takes commands removeImage(), removeText() and replace()
startButton	Button that takes command showFrame() to navigate to StartPage when defined in class InfoPage and additionally takes commands

	removeImage(), removeTake() and replace when defined in class ModelPage
Other variables	
chosenImage	Holds pre-processed user-chosen image from computer
edibility	Global variable that is either edible or poisonous depending on value of prediction
file	User-chosen image from computer
model	Pre-trained model as loaded from folder
prediction	Global variable that holds models classification prediction of chosen image
probability	Global variable that holds certainty of prediction in %

Note that the interface script furthermore contains numerous less-significant local variable and local tkinter specification variables that are defined separately in each class e.g. variables named ‘style‘, ‘bg’ and ‘label’. These variables manage the general aesthetics of the interface and place items such as background image, headers, text and frames onto the separate pages.

6.6 Module Analysis

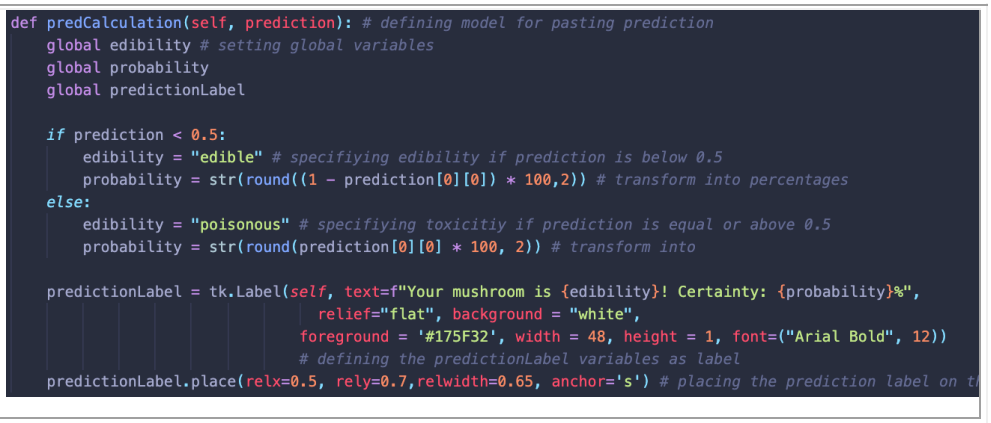
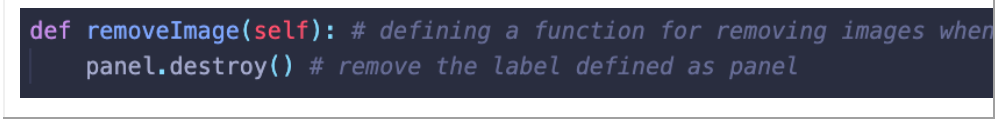
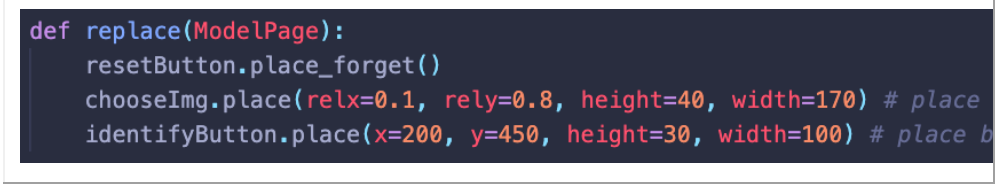
The interface is coded based on the concepts of object-oriented programming paradigm in order to ease the process of navigating through multiple pages in our program and placing different objects on them. This process entails defining a main back-end class that holds all globally needed functions and a container that contains all of the separate pages. Following module analysis table runs through the four defined classes, the self-defined functions and the most important built-in tkinter functions that are implemented in the script. The classes are listed chronologically in accordance with the structure of the script (to maximize understandability) whilst the functions are listed in alphabetical order:

Classes (Pages within the app)	
MainApp	A class essentially functioning as the 'back-end' for the GUI. In here all functions, as stated below, are defined and all other classes inherit all functions from this parent-class. Furthermore, contains a container variable in which all the other pages are stored and instructs the program to show StartPage frame when the script is initiated. This class takes the overarching parameters of the GUI such as the title, the window-size ('geometry') and the limitations on window-reshaping.
StartPage	This class inherits functions and key-parameters from the parent-class and tkinter and specifies a frame with a specific background image and locally specified buttons and labels. This functions mainly as a starting page used for navigation to the modelling page and the information page and is the page displayed when the script is initiated.
ModelPage	This class inherits functions and key-parameters from the parent-class and tkinter and specifies a frame with a specific background image and locally specified buttons and labels. This functions as the page through which the user can upload their own images as well as classify them by running them through the pre-trained neural network model.
InfoPage	This class inherits functions and key-parameters from the parent-class and tkinter and specifies a frame with a specific background image and locally specified buttons and labels. This function merely as an extra layer of interface complexity and provides a small amount of information about the mushrooms identifiable by the TensorFlow classifier.
Functions (function defined to allow user-interaction and run GUI)	
addImage()	Sets a variable 'file' to global to be used when calculating a prediction on the image.

	Sets the variable ‘file’ as an image chosen using the filedialog module which allows the user to choose any image of .jpg-format.
	`def addImage(self): # defining a function for choosing image`
	`  global file # Making 'file'-variable global so it can be accessed in rest of script`
	`  file = filedialog.askopenfilename(initialdir="/", title="Select File", ######### NEED TO MAKE`
	`  filetypes=(("JPG-images", "*.jpg"), ("all files", "*")))`

openImage()	Sets a variable ‘panel’ to global (so it can be accessed by other functions) to be used when displaying the user-chosen image.
	Sets a variable ‘img’ to the previously defined variable ‘file’ resizing it as well as changing the format into a tk.PhotoImage for the image to be displayable in the GUI.
	Creates a label named panel and places the panel on the model-page of the GUI.
	`def openImage(self): # defining a function for pasting chosen image to frame`
	`  global panel # Making 'panel'-variable global so it can be accessed in res`
	`  img = Image.open(file) # Opening image`
	`  img = img.resize((150, 150)) # Uploaded image will be warped to a square l`
	`  img = ImageTk.PhotoImage(img) #  Image into PhotoImage format`
	`  panel = tk.Label(self, image=img) # Turn image into label`
	`  panel.image = img`
	`  panel.place(x=175, y=158) # Place label on frame`

predCalucla tion()	Sets the variables ‘edibility’, ‘probability’ and ‘predictionLabel’ to global for display on model-page. Sets the edibility status to either ‘edible’ or ‘poisonous’ dependent on the outcome of the sigmoid function in the final layer of the neural network model. If the function outcome is above 0.5 the prediction is set to ‘edible’ and the certainty is calculated as a probability and saved in the ‘probability’ variable. If function outcome is below 0.5 the prediction it is set to ‘poisonous’ and the certainty is calculated as a probability by subtracting the function outcome from 1 and saved in the ‘probability’ variable. A tk.Label is then created by name of ‘predictionLabel’ adding the values of the earlier defined variables to a string of text which is placed on the frame of the model-page below the image.

	
preprocess()	Sets the user-chosen image as a global variable named ‘chosenImage’. Sets the variable ‘imgSize’ to 100 changing the dimensions of the chosen image to 100x100. The pixel colour values ae then standardised by dividing with 255 (maximum colour intensity), resized using the variable ‘imgSize’ and reshaped to have the proper dimensions to work in consort with the TensorFlow model. 
removeImage()	Destroys the variable ‘panel’ removing it from the environment. 
removeText()	Destroys the variable ‘predictionLabel’ removing it from the environment. 
replace()	Removes the ‘reset’ button from the frame. Places the ‘Choose Image (chooseImg)’ button on the frame. Places the ‘Identify (identifyButton)’ button on the frame. 
reset()	Removes the ‘Identify (identifyButton)’ button from the frame.

	Removes the 'Choose Image (chooseImg)' button from the frame.
	Places the 'reset' button on the frame as defined under the class ModelPage.
	`def reset(ModelPage):`
	`  identifyButton.place_forget()`
	`  chooseImg.place_forget()`
	`  resetButton.place(relx=0.1, rely=0.8, height=40, width=170)`

runModel()	Sets the variable 'prediction' as global.
	Specifies the variable 'model' as the pre-trained model saved in the folder along with the scripts and images.
	Sets variable 'prediction' to a prediction based on running the specified model on the user-chosen image.
	`def runModel(self): # defining function analysing preprocessed models`
	`  global prediction # making the variable global`
	`  model = tf.keras.models.load_model("0-128-1-CNN.model") # loading the model from`
	`  prediction = model.predict([chosenImage]) # categorizing an image saving and sav`

showFrame()	Sets the variable 'frame' equal to a variable from a list as defined by which button is pressed in the GUI.
	Raises the specific frame to the top layer to display it on the GUI.
	`def showFrame(self, name): # defining a function for sh`
	`  frame = self.frames[name] # specify which frame to`
	`  frame.tkraise() # raise the frame to top`

Important built-in tkinter functions	
ttk.Button	Specifies a variable as a button with a certain style and commands attached to them. Example of the specification of the variable 'startButton' in InfoPage class: • Button text defined as "Start Page" • Style set to "Custom.TButton" ○ 'Custom.TButton' configured class-locally by variable 'style'

	• Lambda command set to 'controller.showFrame(StartPage) which calls function showFrame() raising the StartPage frame to the top when button is pressed
	`startButton = ttk.Button(self, text="Start Page", style = 'Custom.TButton', command=lambda : controller.showFrame(StartPage)) #`
	`startButton.place(x=165, y=400, height=40, width=170) # place the button on`

tk.Frame	Initiates a frame for each page for labels, buttons and images to be placed upon.
	
	Example of the specification of one of the empty white info-boxes on the information page. Aesthetics and and size of frame defined separately:
	`lowerFrameRight = tk.Frame(self, bg='white', bd=10, relief='raised', borderwidth = 5) #`
	`lowerFrameRight.place(relx=0.73, rely=0.25, relwidth=0.4, relheight=0.5, anchor='n') # p`

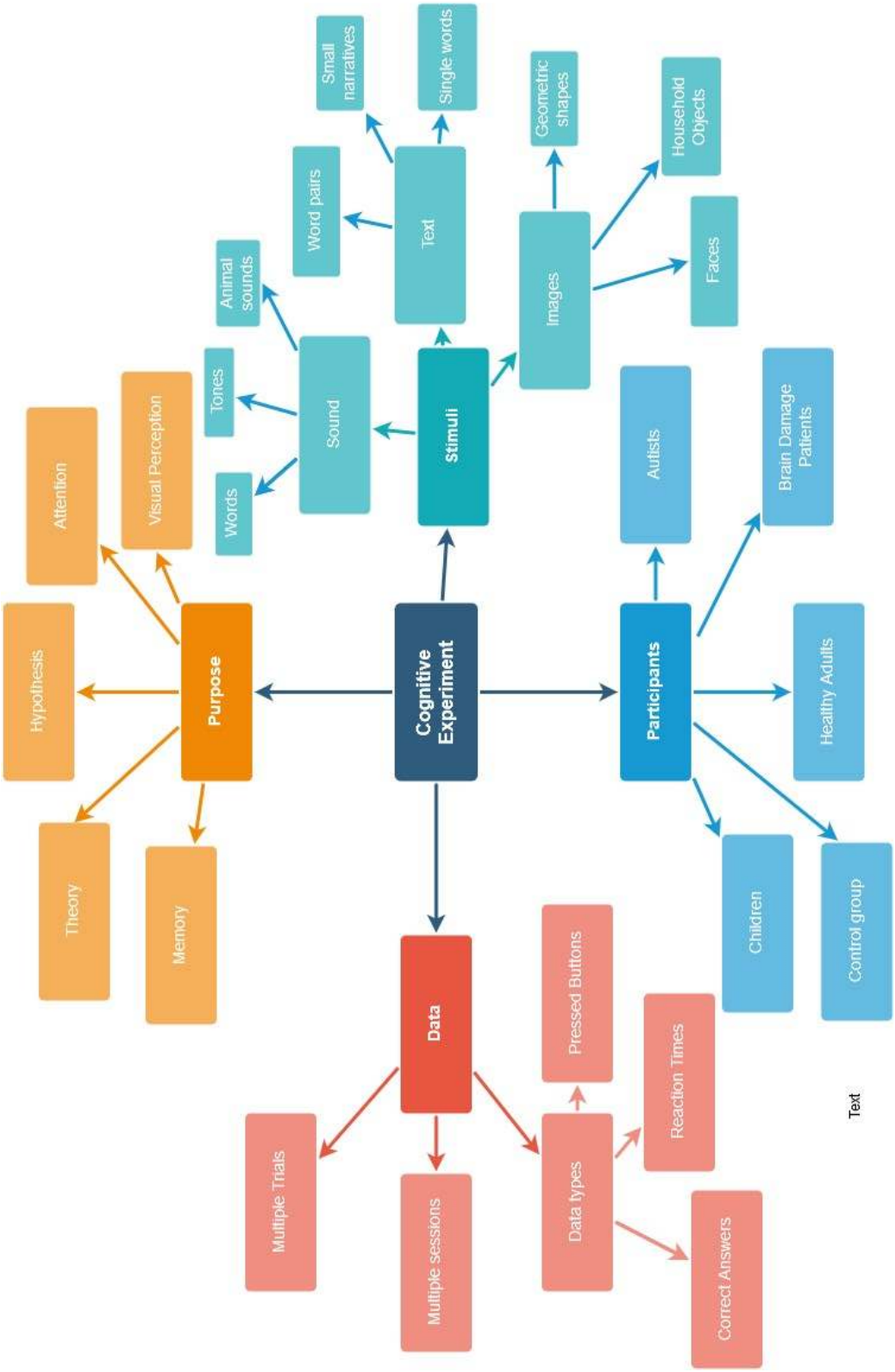
tk.Label	Specifies a variable as a label with certain attributes such as background colour, image or text.
tk.PhotoImage	Takes an image and manipulating the format to be used as a label as e.g. a background image to a page.

x.place	Used with any object (e.g. button, label or image) to place it at specified coordinates within the defined frame.
	
	Example of using both tk.Label, tk.Photoimge and tk.place in order to read, process, define it as an object and finally place an image as the background image:
	`forest = tk.PhotoImage(file="bgForest.gif") # change background to`
	`bg = tk.Label(self, image=forest) # defining bg as a label using d`
	`bg.place(x=0, y=0, relwidth=1, relheight=1) # placing the image as`

Further understanding of the code and the structure/syntax of the program may be obtained from reading the code comments in the scripts.

Appendix

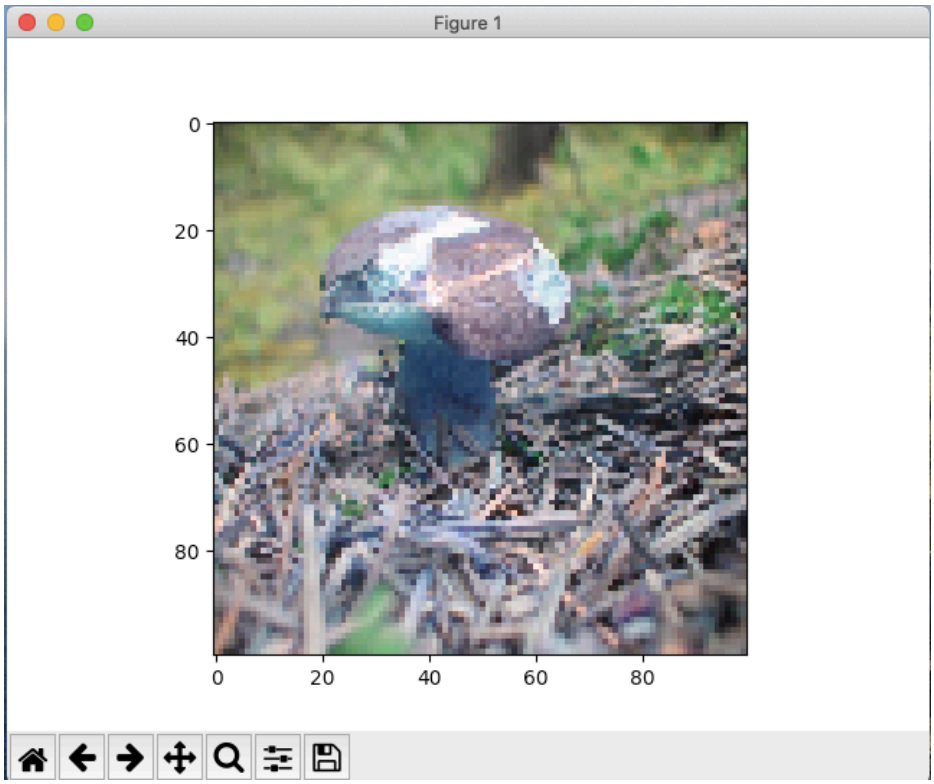
Mind map from 1.2



Pre-processing the data

The following section provides details on how the training images were pre-processed in order to be analyzable for the neural network model. Training data was gathered by scraping google and using a picture database provided by Naturstyrelsen.

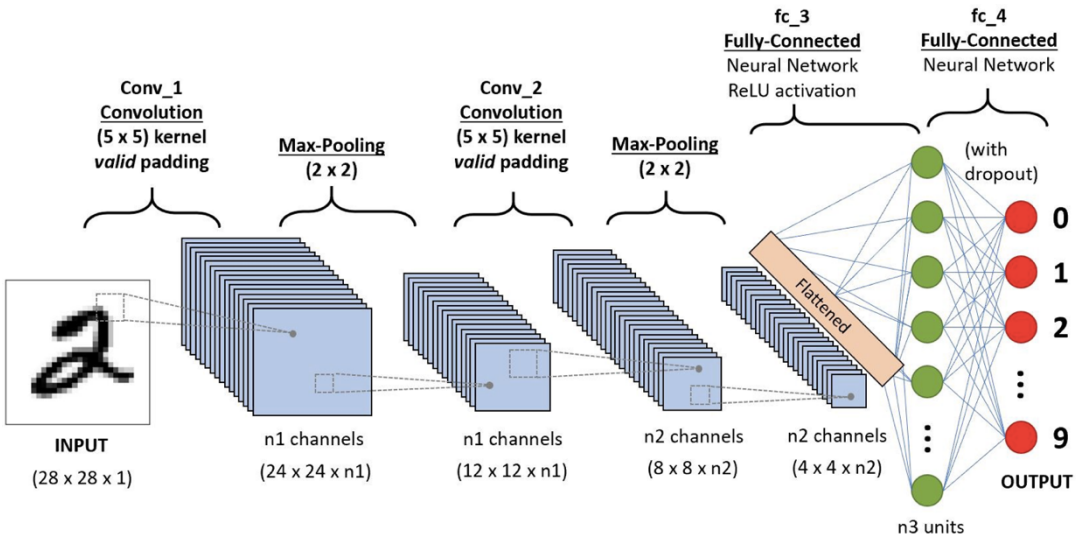
Firstly, all training images were loaded into the script separately. Each image was subsequently assigned a label based on whether it contained an edible or a poisonous mushroom. To reduce computational strain and to ease the process of detecting patterns in the pictures, each image was resized to 100x100 pixels:



For each pixel, colour values on the scale 0-255 were extracted for both green, red and blue. This process converted the image into three dimensional 100x100x3 matrices containing three colour values for each pixel. Finally, all image-matrices and the corresponding category-labels were exported in two separate pickle files.

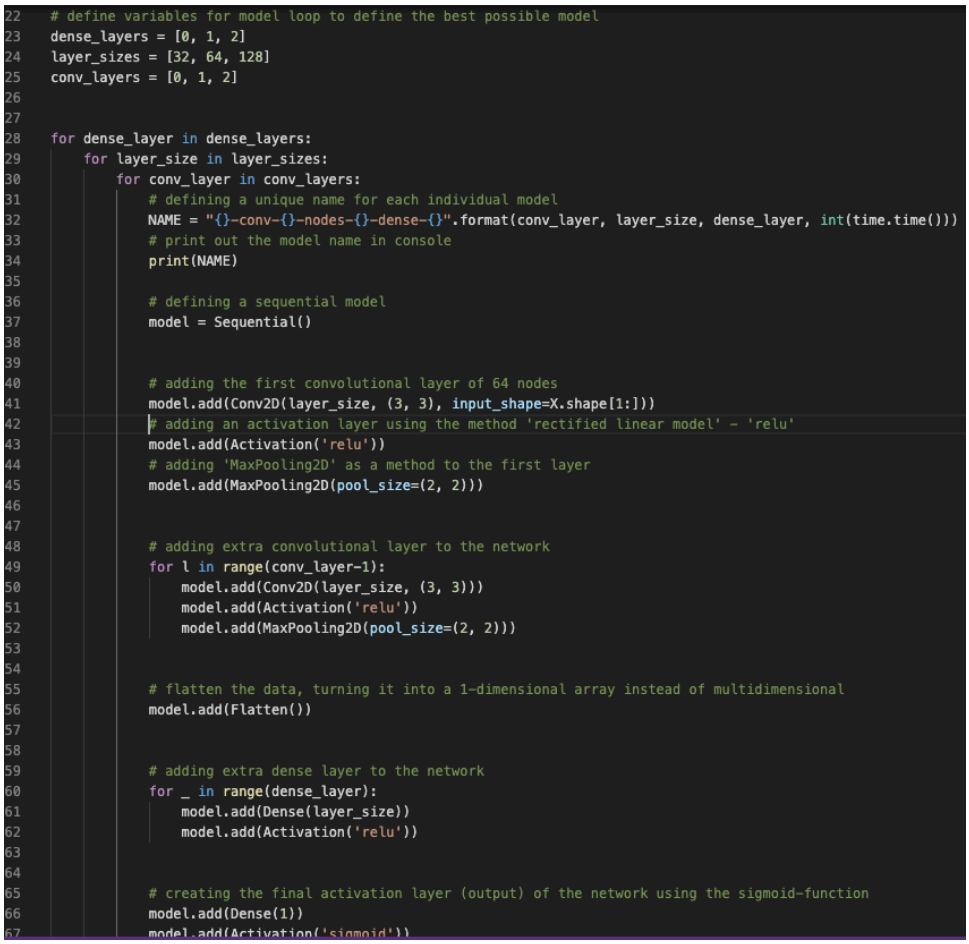
Structuring and training neural network

A convolutional neural network (CNN) is a supervised machine learning algorithm that is particularly specialised for image recognition. In a CNN, the image matrices are first fed through a series of convolutional layers that analyses and reshapes that data using a set of learnable parameters called kernels or filters. Each kernel has a specified receptive field which the convolutional layer uses to produce small 2D activation maps of small sub-parts of the matrices and the algorithm learns patterns in the data through these activation maps. Apart from the data being transformed by the convolutional layers it is also possible to specify so called ReLU and max pooling layers. These respectively remove negative values and collapse values in a small sub-part of the matrix into one point (the highest value). Finally, the three-dimensional matrix is flattened into a column vector which can be passed through one or more dense layers which finally (in our case) output one value. These dense layers consist of a number of fully connected nodes or neurons that are connected via weights and which passes the sum of all received values through an activation function (we used the ReLU function) and uses this value as the output for the next layer. Thus, each input from the column vector influence the rest of the network. The final layer in our network consists of just one neuron. The output of this neuron (we used the sigmoid activation function for this neuron) is the prediction made by the algorithm. In our case, if the value is above 0.5 it indicates that the picture shows a poisonous mushroom and it the value is below 0.5 it is an edible mushroom. The following picture visualizes the concept of a CNN:



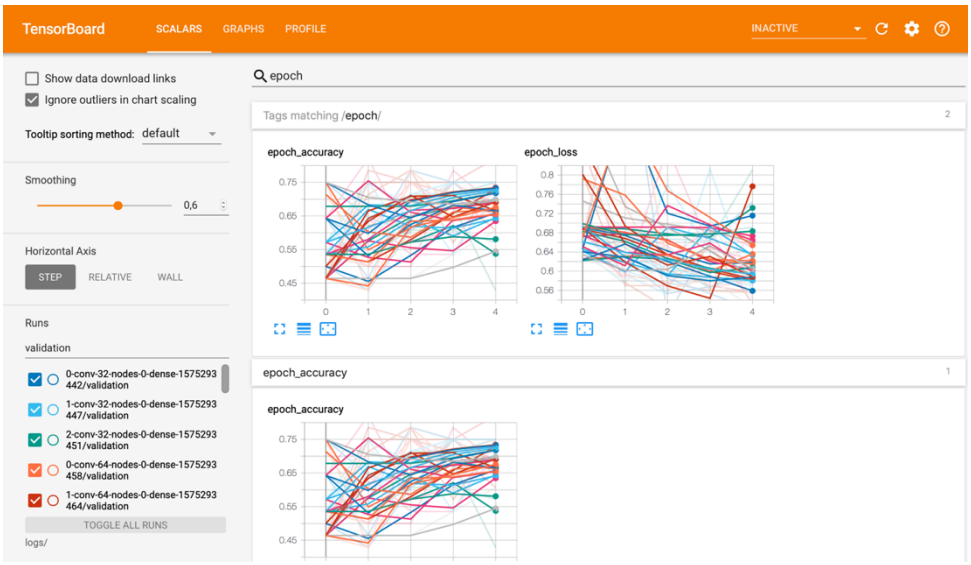
We used the TensorFlow package to define, train and export our neural network. This package contains numerous helpful functions.

In order to optimize the precision our classification algorithm, we decided to loop through and test multiple different models with varying number of convolutional layers, dense layers and number of neurons in each layer:



```
# define variables for model loop to define the best possible model
dense_layers = [0, 1, 2]
layer_sizes = [32, 64, 128]
conv_layers = [0, 1, 2]


for dense_layer in dense_layers:
    for layer_size in layer_sizes:
        for conv_layer in conv_layers:
            # defining a unique name for each individual model
            NAME = "{}-conv-{}-nodes-{}-dense-{}".format(conv_layer, layer_size, dense_layer, int(time.time()))
            # print out the model name in console
            print(NAME)

            # defining a sequential model
            model = Sequential()


            # adding the first convolutional layer of 64 nodes
            model.add(Conv2D(layer_size, (3, 3), input_shape=X.shape[1:]))
            # adding an activation layer using the method 'rectified linear model' - 'relu'
            model.add(Activation('relu'))
            # adding 'MaxPooling2D' as a method to the first layer
            model.add(MaxPooling2D(pool_size=(2, 2)))


            # adding extra convolutional layer to the network
            for l in range(conv_layer-1):
                model.add(Conv2D(layer_size, (3, 3)))
                model.add(Activation('relu'))
                model.add(MaxPooling2D(pool_size=(2, 2)))


            # flatten the data, turning it into a 1-dimensional array instead of multidimensional
            model.add(Flatten())


            # adding extra dense layer to the network
            for _ in range(dense_layer):
                model.add(Dense(layer_size))
                model.add(Activation('relu'))


            # creating the final activation layer (output) of the network using the sigmoid-function
            model.add(Dense(1))
            model.add(Activation('sigmoid'))
```

Using the TensorBoard function from TensorFlow it was possible to track the progress of each of the different models to investigate which performed with the highest precision and produced the minimal loss:



The model with the highest precision consisted of one convolutional layer, one ReLu layer, one max pooling layer and a dense layer consisting of 128 nodes (the model name starts with '0-conv' as one convolutional, ReLu and max pooling layer was defined outside the for-loop). The model yielded a precision of 85% after five-fold cross-validation:

Precision:

Name	Smoothed	Value	Step	Time	Relative
0-conv-128-nodes-1-dense-1576071746/validation	0.8499	0.8378	4	Wed Dec 11, 14:43:20	38s

Loss:

Name	Smoothed	Value	Step	Time	Relative
0-conv-128-nodes-1-dense-1576071746/validation	0.4417	0.4358	4	Wed Dec 11, 14:43:20	38s

Code Screenshots

Pre-processing:

```python
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
from tqdm import tqdm

import random
import pickle

CATEGORIES = ["edible", "deadly"]


for category in CATEGORIES:  # do edible and deadly shrooms
    path = os.path.join(DATADIR,category)  # create path to different kinds
    for img in os.listdir(path):  # iterate over each image in each folder
        img_array = cv2.imread(os.path.join(path,img) ,cv2.IMREAD_COLOR)  # convert the grey-scaled image to array
        plt.imshow(img_array, cmap='color')  # graph it
        plt.show()  # display!

        break  # we just want one for now so break
    break  #...and one more!


# setting variable image-size
imgSize = 100

new_array = cv2.resize(img_array, (imgSize, imgSize)) # reshaping the array to new dimensions
plt.imshow(new_array, cmap='color') # show the image as array
plt.show() # show the plot

```

```python

training_data = [] # define variable 'training_data' as an empty list

def create_training_data():
    for category in CATEGORIES:  # loop through folders containing 'edible' and 'poisonous' images.

        path = os.path.join(DATADIR,category)  # create path do edible & deadly
        class_num = CATEGORIES.index(category)  # get the classification  (0 or a 1). 0=edibly, 1=deadly

        for img in tqdm(os.listdir(path)):  # iterate over each image per category
            try:
                img_array = cv2.imread(os.path.join(path,img), cv2.IMREAD_COLOR)  # convert to array
                new_array = cv2.resize(img_array, (imgSize, imgSize))  # resize to normalize (100x100)
                training_data.append([new_array, class_num])  # add this to our training_data
            except Exception as e:  # in the interest in keeping the output clean...
                pass

create_training_data() # run the above defined functions to create categories


random.shuffle(training_data) # shuffle the training data to overcome potential bias

feats = [] # define variable 'feats' as empty list
labels = [] # defined variable 'labels' as empty list

for features, label in training_data: # start loop iterating through images
    feats.append(features) # append the features for each image
    labels.append(label) # append the label for each image

feats = np.array(feats).reshape(-3, imgSize, imgSize, 3) # reshape the array that is each image
```

```
61    labels = np.array(labels) # turn the labels vector into an array using numpy
62
63    pickle_out = open("feats.pickle","wb") # create a 'pickle' called feats
64    pickle.dump(feats, pickle_out) # add the features to the defined pickle
65    pickle_out.close() # close the pickle
66
67    pickle_out = open("labels.pickle","wb") # create a 'pickle' called labels
68    pickle.dump(labels, pickle_out) # add the labels to the defined pickle
69    pickle_out.close() # close the pickle
```

Model-training:

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.callbacks import TensorBoard
import pickle
import time
import cv2


pickle_in = open("feats.pickle","rb") # reload the pre-processed data
X = pickle.load(pickle_in) # define features as X

pickle_in = open("labels.pickle","rb") # reload the pre-processed labels
y = pickle.load(pickle_in) # define labels as y


X = X / 255.0 # Scaling our X-variable to be on the spectrum of 0-1


# define variables for model loop to define the best possible model
layer_sizes: list
dense_layers = [0, 1, 2] # define amount of dense layers to test
layer_sizes = [32, 64, 128] # define amount of nodes to test
conv_layers = [0, 1, 2] # define amount of convolutional layers to test


for dense_layer in dense_layers: # iterate through kinds of dense-layers
    for layer_size in layer_sizes: # iterate through kinds of layer-sizes
        for conv_layer in conv_layers: # iterate through amount of convolutional layers
            # defining a unique name for each individual model
            NAME = "{}-conv-{}-nodes-{}-dense-{}".format(conv_layer, layer_size, dense_layer, int(time.time()))
            # print out the model name in console
            print(NAME)
```

```python
            model = Sequential() # defining a sequential model


            model.add(Conv2D(layer_size, (3, 3), input_shape=X.shape[1:])) # adding the first convolutional layer of 64 nodes
            model.add(Activation('relu')) # adding an activation layer using the method 'rectified linear model' - 'relu'
            model.add(MaxPooling2D(pool_size=(2, 2))) # adding 'MaxPooling2D' as a method to the first layer



            for l in range(conv_layer-1): # add an extra convolutional layer to the network
                model.add(Conv2D(layer_size, (3, 3))) # add a conv-layer pooling as 3x3
                model.add(Activation('relu')) # add activation function
                model.add(MaxPooling2D(pool_size=(2, 2))) # set the maximum pooling of features


            model.add(Flatten()) # flatten the data, turning it into a 1-dimensional array instead of multidimensional


            for _ in range(dense_layer): # adding extra dense layer to the network
                model.add(Dense(layer_size)) # add the dense layer with 'layer_size'
                model.add(Activation('relu')) # add activation function


            model.add(Dense(1)) # creating the final activation layer (output) of the network using the sigmoid-function
            model.add(Activation('sigmoid')) # add activation function


            dsBoard = TensorBoard(log_dir="logs/{}".format(NAME)) # defining dsBoard to investigate loss-function and accuracy using


            model.compile(loss='binary_crossentropy',
                          optimizer='adam',
                          metrics=['accuracy'],) # compile the model using binary classifcation


            model.fit(X, y,
                      batch_size=5,
                      epochs=5,
                      validation_split=0.2,
                      callbacks=[dsBoard]) # fit the model to the specified data

# unhash if you wish to save a model. This script is however mainly for inspecting how changing the parameters
# of the models changed the accuracy and the loss-function.

#model.save("0-128-1-CNN.model") # saving the model when it's been run with the right data
```

Graphical user-interface:

```python
# Importing module 'tkinter' for graphical user interface
import tkinter as tk
from tkinter import ttk
from tkinter import filedialog
from tkinter import PhotoImage
from PIL import Image, ImageTk

# import numpy for easier data-transformations
import numpy as np

# Importing module 'cv2' for importing and modifying images
import cv2

# Importing module 'tensorflow' for neural network processing
import tensorflow as tf

import pathlib

current_dir = pathlib.Path.cwd()
home_dir = pathlib.Path.home()
print(current_dir)
print(home_dir)


class MainApp(tk.Tk): # Defining the main-window as a class for back-end control of pages in the GUI
    def __init__(self, *args, **kwargs): # Have the class self-initiate all the following when the app is run
        tk.Tk.__init__(self, *args, **kwargs) # Initiate tkinter functions
        self.title("Doom Shroom Identifier") # Set the title of the main-app
        self.geometry("500x500") # Set the size of the main-app window
        self.pack_propagate(0) # Set pack_propagate to zero to not allow widgets to control window-size
        self.resizable(0,0) # Don't allow resizing of windows

        box = tk.Frame(self) # create a box for all pages
        box.pack(side="top", fill="both", expand=True) # calling the container
        box.grid_rowconfigure(0, weight=1) # make the cell for rows and columns cover the entire defined window
        box.grid_columnconfigure(0, weight=1)

        self.frames = {} # create a dictionary of frames we want to navigate

        for F in (StartPage, ModelPage, InfoPage): # for each page
            frame = F(box, self) # create the page
            self.frames[F] = frame  # store into frames
            frame.grid(row=0, column=0, sticky="nsew") # grid it to container

        self.showFrame(StartPage) # set StartPage to be the front page
```

```python


    def showFrame(self, name): # defining a function for showing frames
        frame = self.frames[name] # specify which frame to show
        frame.tkraise() # raise the frame to top


    def addImage(self): # defining a function for choosing image
        global file # Making 'file'-variable global so it can be accessed in rest of script
        file = filedialog.askopenfilename(initialdir="/", title="Select File",
                                          filetypes=(("JPG-images", "*.jpg"), ("all files", "*")))


    def openImage(self): # defining a function for pasting chosen image to frame
        global panel # Making 'panel'-variable global so it can be accessed in rest of script
        img = Image.open(file) # Opening image
        img = img.resize((150, 150)) # Uploaded image will be warped to a square like in the model
        img = ImageTk.PhotoImage(img) #  Image into PhotoImage format
        panel = tk.Label(self, image=img) # Turn image into label
        panel.image = img
        panel.place(x=175, y=158) # Place label on frame
```

```
67
68      def removeText(self): # defining a function for removing images when leaving a page
69          predictionLabel.destroy() # remove the label defined as predictionLabel
70
71      def removeImage(self): # defining a function for removing images when leaving a page
72          panel.destroy() # remove the label defined as panel
73
74      def reset(ModelPage):
75          classifyButton.place_forget() # forget the variable defined as 'classifyButton'
76          chooseImg.place_forget() # forget the variable defined as 'chooseImg'
77          resetButton.place(relx=0.1, rely=0.8, height=40, width=170) # place the button-variable defined as 'resetBu
78
79      def replace(ModelPage):
80          resetButton.place_forget() # forget the variable defined as 'resetButton'
81          chooseImg.place(relx=0.1, rely=0.8, height=40, width=170) # place button on frame
82          classifyButton.place(x=200, y=450, height=30, width=100) # place button on frame
83
84
85
86      def preprocess(self,file): # defining a function for preprocessing new images
87          global chosenImage # making the variable global
88          imgSize = 100 # setting standard pixel width/height for all images
89          chosenImage = cv2.imread(file, cv2.IMREAD_COLOR) / 255.0 # reading image using cv2 and standardizing
90          chosenImage = cv2.resize(chosenImage, (imgSize, imgSize)) # resizing the image to standard size
91          chosenImage = chosenImage.reshape(-3, imgSize, imgSize, 3) # reshaping image to allow model-prediction
92
93
94      def runModel(self): # defining function analysing preprocessed models
95          global prediction # making the variable global
96          model = tf.keras.models.load_model("0-128-1-CNN.model") # loading the model from current directory
97          prediction = model.predict([chosenImage]) # categorizing an image saving and saving the prediction
98          # print(prediction)
99
100
101     def predCalculation(self, prediction): # defining model for pasting prediction
102         global edibility # setting global variables
103         global probability
104         global predictionLabel
105
106         if prediction < 0.5:
107             edibility = "edible" # specifiying edibility if prediction is below 0.5
108             probability = str(round((1 - prediction[0][0]) * 100,2)) # transform into percentages
109         else:
110             edibility = "poisonous" # specifiying toxicitiy if prediction is equal or above 0.5
111             probability = str(round(prediction[0][0] * 100, 2)) # transform into
```

```
112
113         predictionLabel = tk.Label(self, text=f"Your mushroom is {edibility}! Certainty: {probability}%",
114                                    relief="flat", background = "white",
115                                    foreground = '#175F32', width = 48, height = 1, font=("Arial Bold", 12))
116                                    # defining the predictionLabel variables as label
117         predictionLabel.place(relx=0.5, rely=0.7,relwidth=0.65, anchor='s') # placing the prediction label on the f
118
119
120
121
122 class StartPage(tk.Frame): # Defining a class for the start page inheriting functions from tk.Frame
123     def __init__(self, parent, controller): # have the class self-initiate functions from itself and parent-class
124         tk.Frame.__init__(self, parent)# initiate a frame for the page
125
126         photo = tk.PhotoImage(file="bgImage.gif") # add a background image
127         bg = tk.Label(self, image=photo) # create a label containing the specified photo
128         bg.place(x=0, y=0, relwidth=1, relheight=1) # placing the bg filling the entire frame
129         bg.image = photo # using the .image function of tkiner to call the background as the photo
130
```

```
        style = ttk.Style() # styling buttons using ttk module
        style.configure('Custom.TButton', font=("Arial Bold", 13),
                        foreground='#175F32') # defining specifications for buttons

        modelButton = ttk.Button(self, text="Classify Image", style = 'Custom.TButton',
                                 command=lambda : controller.showFrame(ModelPage))
                                 # adding button to access the ModelPage
        modelButton.place(relx=0.1, rely=0.8, height=40, width=170) # placing the button on the page

        infoButton = ttk.Button(self, text="More Information", style = 'Custom.TButton',
                                 command=lambda : controller.showFrame(InfoPage))
                                 # adding button to access the InfoPage
        infoButton.place(relx=0.55, rely=0.8, height=40, width=170) # placing the button on the page



class ModelPage(tk.Frame): # define a class for running the model on personally chosen picture
    def __init__(self, parent, controller): # have the class self-initiate functions from itself and parent-class as well as cont

        global classifyButton
        global resetButton
        global chooseImg

        tk.Frame.__init__(self, parent) # initiate a frame for the page

        forest = tk.PhotoImage(file="bgForest.gif") # change background to foresty image
        bg = tk.Label(self, image=forest) # defining bg as a label using defined image
        bg.place(x=0, y=0, relwidth=1, relheight=1) # placing the image as background
        bg.image = forest # using the .image function of tkiner to call the background as the photo

        label = tk.Label(self, text="CLASSIFY", relief="flat", background = "#175F32",
                         foreground = 'white', width = 200, height = 2, font=("Arial Bold", 28), highlightcolor="white") # add a l
        label.place(relx=0.285, rely=0.08, width = 220) # place label and align it at the center of main-app

        style = ttk.Style() # styling buttons using ttk module
        style.configure('Custom.TButton', font=("Arial Bold", 13),
                        foreground='#175F32') # defining specifications for buttons

        chooseImg = ttk.Button(self, text="Choose Image", style = 'Custom.TButton',
                                 command=lambda : [controller.addImage(), controller.openImage()])
                                 # add button for adding and opening images
        chooseImg.place(relx=0.1, rely=0.8, height=40, width=170) # place button on frame

        startButton = ttk.Button(self, text="Start Page", style = 'Custom.TButton',
                                 command=lambda : [controller.showFrame(StartPage), controller.removeImage(),
                                                   controller.removeText(), controller.replace()])
                                 # add button for accessing StartPage and removing image
        startButton.place(relx=0.56, rely=0.8, height=40, width=170) # place button on frame

        classifyButton = ttk.Button(self, text="Classify", style = 'Custom.TButton',
                                 command=lambda : [controller.preprocess(file), controller.runModel(),
                                                   controller.predCalculation(prediction), controller.reset()])
                                 # add button for running model and producing prediction
        classifyButton.place(x=200, y=450, height=30, width=100) # place button on frame

        resetButton = ttk.Button(self, text="Reset", style = 'Custom.TButton',
                                 command=lambda : [controller.removeImage(), controller.removeText(), controller.replace()])
                                 # add button for running model and producing prediction


        lowerFrame = tk.Frame(self, bg='white', bd=10, relief='raised', borderwidth = 5) #  adding white frame in middle of frame
        lowerFrame.place(relx=0.5, rely=0.25, relwidth=0.8, relheight=0.5, anchor='n') # placing the frame

        yourImgLabel = tk.Label(self, text="Your Image", relief="flat", background = "white",
                        foreground = '#175F32', width = 10, height = 1, font=("Arial Bold", 16))
                        # creating a label for header for white frame
        yourImgLabel.place(relx=0.41, y=130) # align it at the center of main-app


class InfoPage(tk.Frame): # Define a class for the information page inheriting functions from tk.Frame
    def __init__(self, parent, controller): # have the class self-initiate functions from itself and parent-class as well as controll
        tk.Frame.__init__(self, parent) # initiate a frame for the page
        forest = tk.PhotoImage(file="bgForest.gif") # change background image
        bg = tk.Label(self, image=forest) # create label with defined photo
        bg.place(x=0, y=0, relwidth=1, relheight=1) # place the image on the frame
        bg.image = forest # using the .image function of tkiner to call the background as the photo
```

```
208
209         label = tk.Label(self, text="INFORMATION", relief="flat", background = "#175F32",
210                          foreground = 'white', width = 18, height = 2, font=("Arial Bold", 28), highlightcolor="white") # add a
211         label.place(relx=0.5, rely=0.1, anchor="center") # place and align it at the center of main-app
212
213         style = ttk.Style() # styling buttons using ttk module
214         style.configure('Custom.TButton', font=("Arial Bold", 13),
215                         foreground='#175F32') # defining specifications for buttons
216
217         startButton = ttk.Button(self, text="Start Page", style = 'Custom.TButton',
218                                  command=lambda : controller.showFrame(StartPage)) # add button for accessing SP
219         startButton.place(x=165, y=400, height=40, width=170) # place the button on the frame
220
221         lowerFrameRight = tk.Frame(self, bg='white', bd=10, relief='raised', borderwidth = 5) # add white frame
222         lowerFrameRight.place(relx=0.73, rely=0.25, relwidth=0.4, relheight=0.5, anchor='n') # place frame
223
224         lowerFrameLeft = tk.Frame(self, bg='white', bd=10, relief='raised', borderwidth = 5) # add white frame
225         lowerFrameLeft.place(relx=0.27, rely=0.25, relwidth=0.4, relheight=0.5, anchor='n') # place frame
226
227         karl = tk.PhotoImage(file="karl.gif") # load image of mushroom
228         karlImg = tk.Label(lowerFrameRight, image=karl) # define label containing defined photo
229         karlImg.place(x=43, y=15) # place label
230         karlImg.image = karl # using the .image function of tkiner to call the background as the photo
231
232         # Adding image
233         satan = tk.PhotoImage(file="satan.gif") # load image of mushroom
234         satanImg = tk.Label(lowerFrameLeft, image=satan) # define label containing defined photo
235         satanImg.place(x=43, y=15) # place label
236         satanImg.image = satan # using the .image function of tkiner to call the background as the photo
237
238         # Defining info text
239         rørText = '''
240         Rørhatten er en svampeslægt,
241         som tilhører Rørhat-ordenen.
242         De flest rørhatte
243         i Danmark er spiselige
244         '''
245
246         # Defining info text
247         gifText = '''
248         Der findes to slags giftige
249         rørhatte i Danmark: Satans
250         rørhat (den der ses på
251         billedet) og djævle rørhat
252         '''
```

```
253
254         rorLabel = tk.Label(lowerFrameRight, text=rørText, relief="flat", background = "white",
255                             foreground = '#175F32', width = 23, height = 7, font=("Times New Roman", 10)) # add info
256         rorLabel.place(relx=0.45, rely=0.99, anchor="s") # place the info-text
257
258         gifLabel = tk.Label(lowerFrameLeft, text=gifText, relief="flat", background = "white",
259                             foreground = '#175F32', width = 23, height = 7, font=("Times New Roman", 10)) # add info
260         gifLabel.place(relx=0.45, rely=0.99, anchor="s") # place the info-text
261
262 app = MainApp() # define variable 'app' as the MainApp() for running tkinter window
263 app.mainloop() # call the mainloop() function on app to activate all defined frames
264
265
266
267
```