Fundamentals of Artificial Intelligence

Report of

PRACTICAL ASSIGNMENT #1 : Game Development

GAME LINK

Aurélien KITTEL

230ADB016

I. DESCRIPTION OF THE GAME

1. Rules

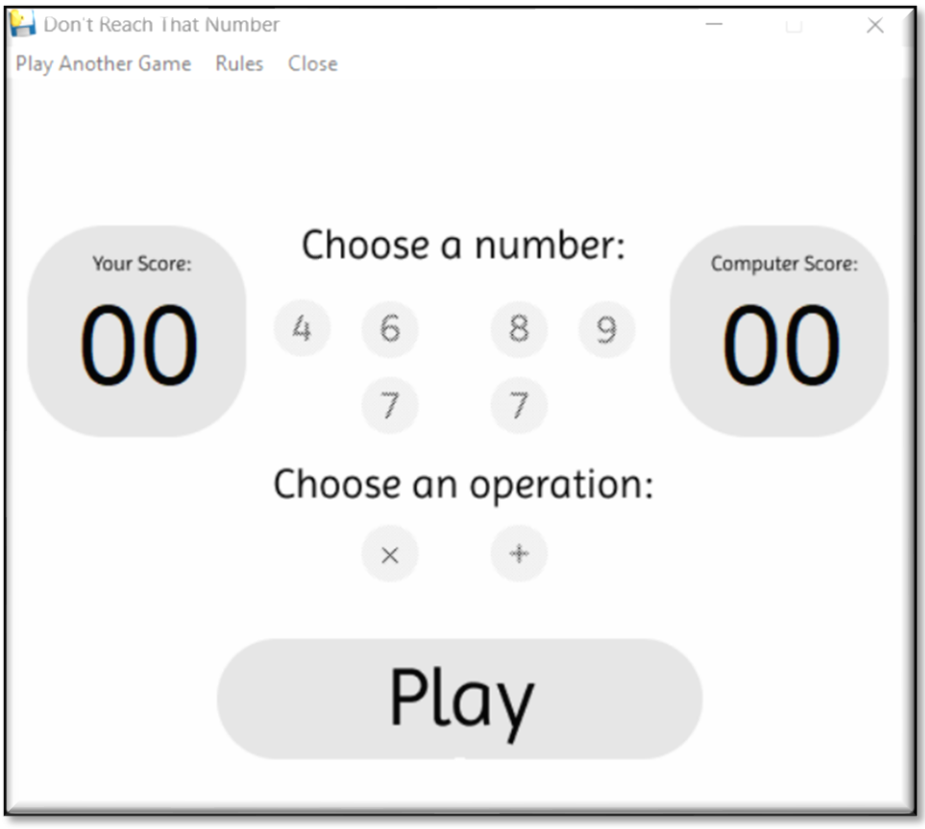
The rules of the game are to use the numbers [4,6,7,7,8,9] to operate operations (sum and multiplication) to increase our score to be as close as possible to a generated number (randomly created between 43 and 67) when it will have no numbers left.

The winner is the player who is the closest to the generated number at the end.

The players must not reach the number because in this case their score would be reset to 0.

Players can exceed the generated number but in this case a modulo will apply.

2. User Interface



This capture represents the most important part of the User Interface and is composed by different elements:

- **Menu:**
  → Play Another Game: This button allows the “human player” to start a game just before the game or even during the game.

  → Rules: This button open a dialog which shows the rules of the game to re-explain it if needed.

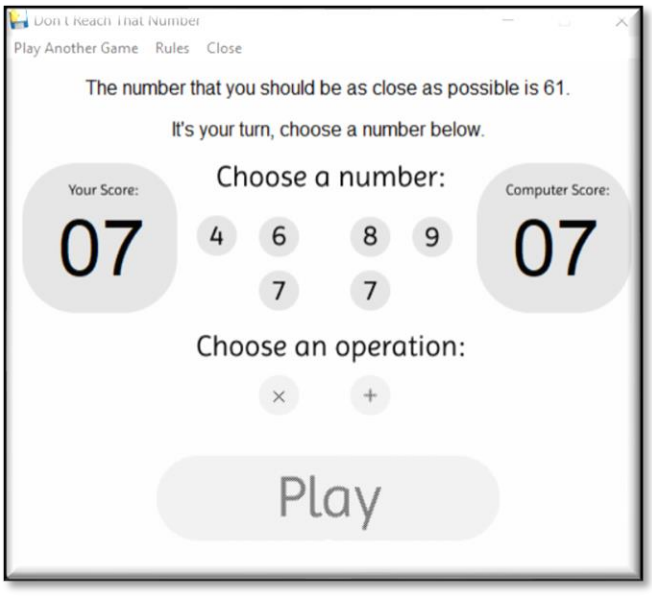
  → Close: This button allows the user to close the game window.

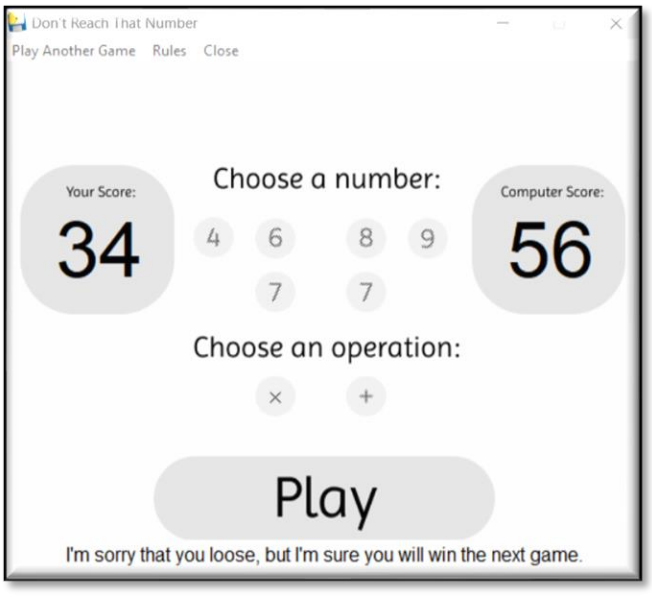
- **Scores:** The scores are updated automatically after a move to always show the current state of the game.

- **Play Button:** This button is clickable only when no game has started or when a game is finished to play again.

- **Numbers Buttons:** The numbers buttons are clickable only for the “human player” during a game at when it should choose the number to increase his score.

- **Operations Buttons:** The operations buttons are clickable only for the “human player” during a game when it should choose the operation to perform to increase his score.





On those two screenshots above, we can see the design of the interface during a game and at the end of it. Three other widgets have been added to the interface:

- **Two Labels on the top of the window** that are shown only during the game:

  → The first label indicates the number that both players should be as close as possible at the end of the game.

  → The second label indicates the next step that the player should perform: choose a number, operator…

- **One Label at the bottom of the window** that is shown only between two games and show the result of the game.

II. USER MANUAL & EXAMPLE OF THE GAME.

Playing the game is simple. First, you got three choices:

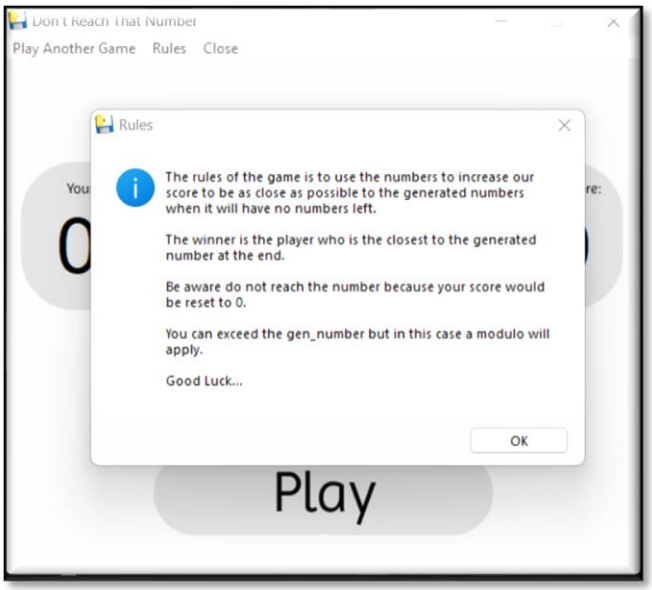
1) Run it through an windows executable file **game_app.exe** that you will find in the folder **EXECUTABLE_APP** in the project folder.

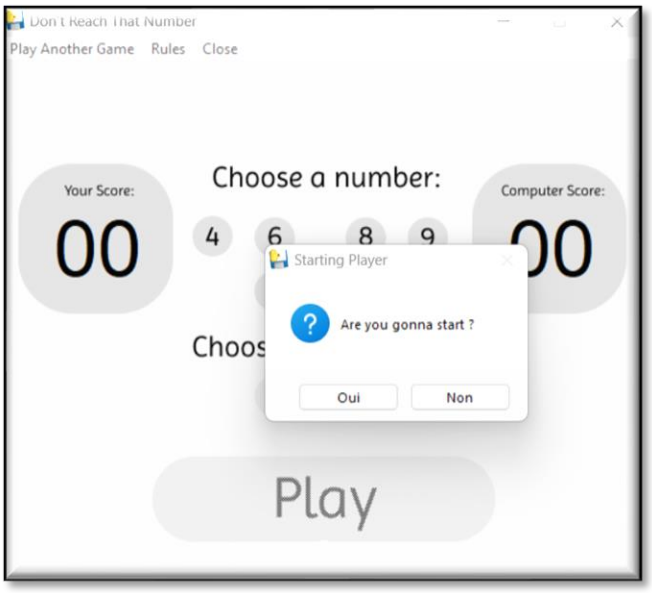
2) Run it through the Python Script **main.py** in the **PYTHON** folder in the project folder. Nevertheless, to run the project through this, you must have the: tkinter, random, PIL and pyglet libraries already installed on your device.

3) Run the Shortcut **Game** in the **PROJECT_FOLDER**.

When you will run the app, through whatever technique you want, it will open a window where you'll see the main UI as presented before.

As said before, you can open a dialog to show the rules if it's the first time that you play the game or even just to remember it.





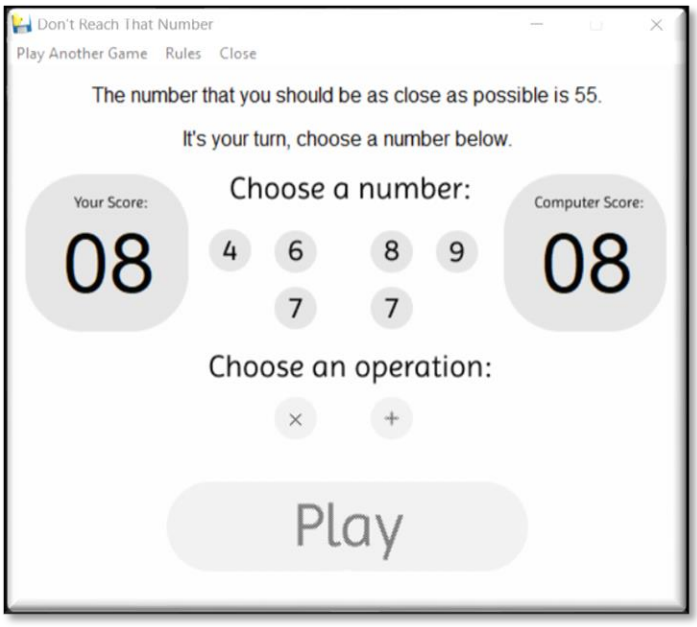
Whatever play button you'll click to start the game, the window will open a yes/no dialog to ask you who will start the game.

If you click on "Oui" (Yes), you will start the game. Otherwise, the computer will perform its first move.

When you'll have chosen which player starts, the game will automatically start by setting the initial scores (randomly created between 6 and 9 included) as well as the number that the player should be as close as possible which will be shown in a top label as seen previously.

The play button is automatically disabled. Nevertheless, it's still possible for the player to start another game by clicking on "Play Another Game" in the menu.

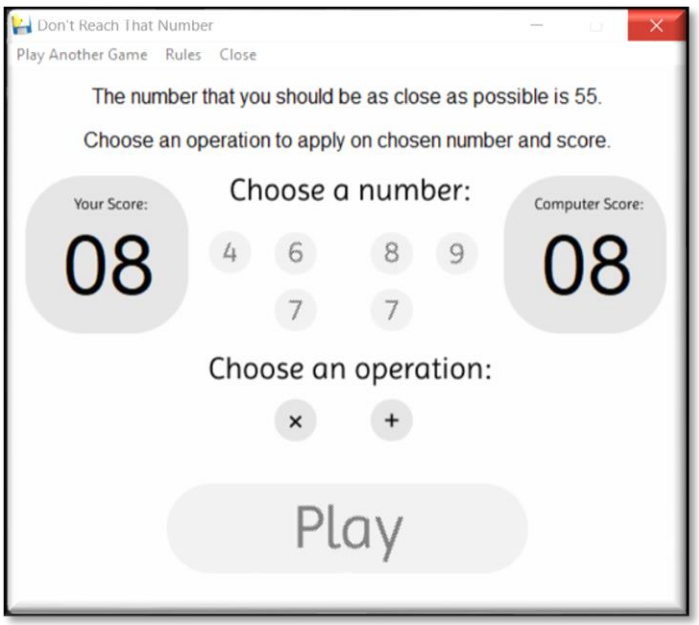
Imagine that we selected "yes" for the following that will detail the game flow.



As it can be seen on the screenshot, the scores have been updated as well as the text labels that now show:

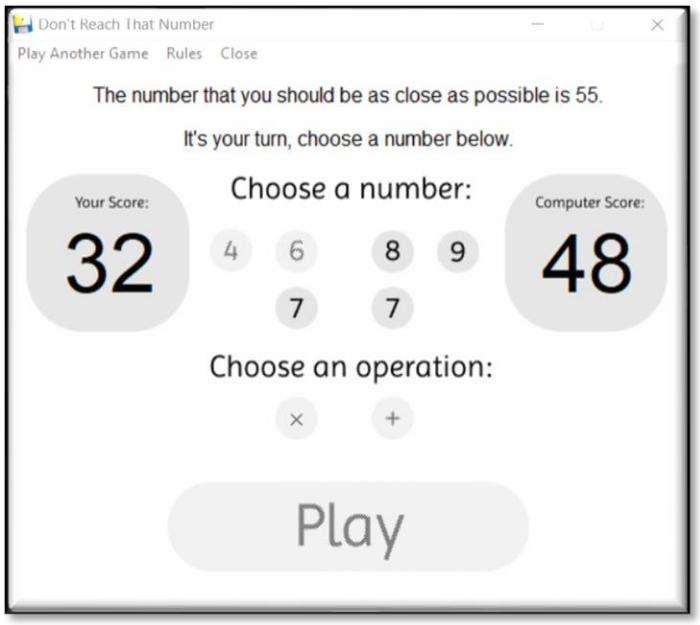
→ The number to be as close as possible.
→ The next action that the player should perform: here, choose a number.

For each player step, it's simple: you follow the instruction written in the second label from the top, for example : "choose a number" or "choose an operator". As it can be seen in the previous screenshot, to avoid errors from the human in the process of the game, only the buttons that permit to go to the next steps are clickable, that's why the operators' buttons are disabled on this screenshot.



This is the next step of the game, now the numbers' buttons are disabled, and the operators' ones are clickable. It can be also seen that the instructions in the second label have changed to tell the player to now choose an operation to perform. After that, the score of the player is update as well as the computer one almost instantly because it performs each of the steps fast.

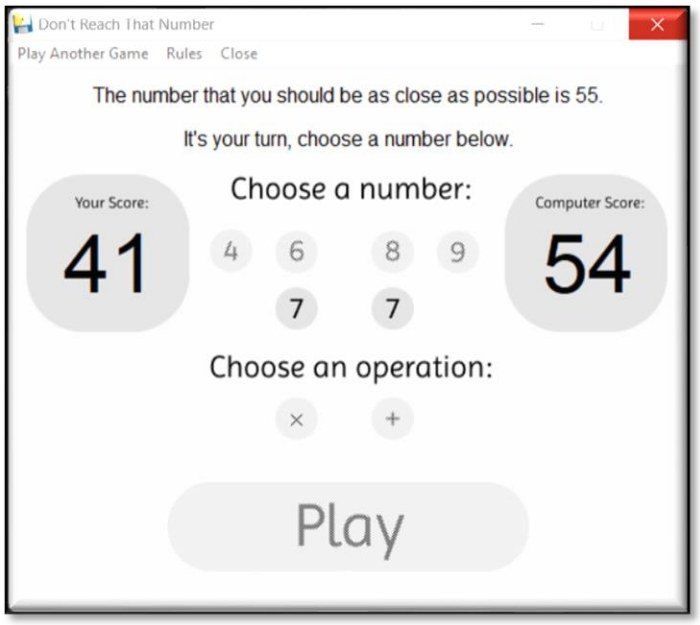
In this example case, I choose 4 and "x" as the first step and the computer 6 and "x".



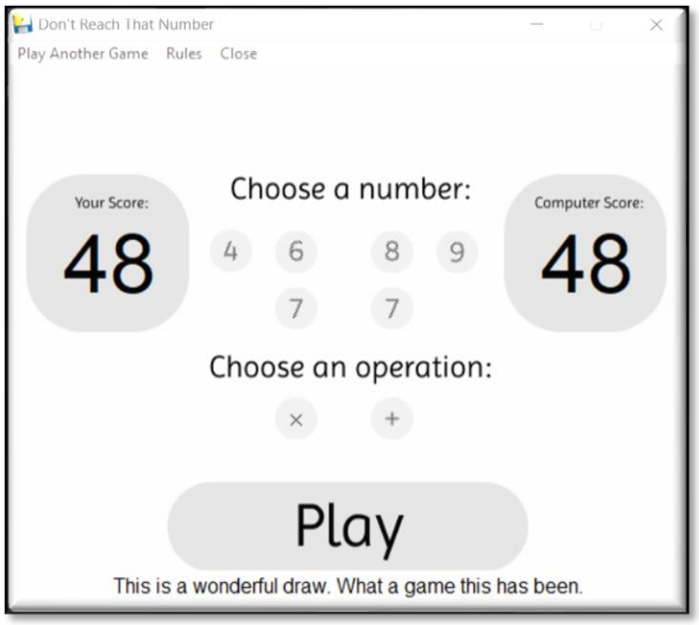
We can see on this screenshot that the operators' buttons are again disabled after performing the action but also that 2 of the numbers' buttons are also disabled, they're the "already-used" buttons that according to the game rules cannot be used anymore. It can also be noticed that as always, the scores have been updated as well as the "step instruction label'.

For the next steps, now that I detailed the basic process, I will only put screenshots along with values that directed to this state of game until the game will be finished.

I choose the value "9" and "+" while computer choose "8" and "x" to arrive to 54 by using the modulo when we exceed the number: (48x8)%55=54.



I choose 7 and "+" while computer choose 7 and "x" using another time the modulo property which conduct us to a draw result.



By analyzing this screenshot, we can see differences between the "during-game" interface and this one:

→ Both top labels have been set to "".

→ All the buttons have been disabled except the play button that have reactivated to allow user to play another game.

→ A bottom label is now shown. We can see now inside it the result of the game.

It's important to notice that if the play button is clicked it will open a dialog asking for which player is starting and then repeat all the states but for the new game with:

→ New initial scores between 6 and 9.

→ New number to reach between 43 and 67.

→ New player starting? It's the choice of the human player…

III. DATA STRUCTURES

The only data structure defined in the Graph Class which represents the nodes and the state space graph of the game.

```
class Graph:
    def __init__(self,val,player,children=[],del_num=-1,op=""):
        self.val=val
        self.children=children
        self.player=player
        self.heuristic_measure=-2
        self.del_num=del_num
        self.op=op
```

This class owns 5 attributes:

1) **Val:** the val attribute is a tuple composed by three values that represent the parameters of the game (player_score, [list of the numbers that can still be used], computer_score).

2) **Children:** List of Graph. This represents the next states that can be reached from current game state (the next possible moves).

3) **Player:** String (“player” or “computer”) that represents the player that will perform the move. It’s an attribute that is notably use in the minimax algorithm to be able to know the current level to be able to perform the right action minimize if it’s “player” and maximize if it’s “computer”.

4) **Heuristic Measure:** Attribute to store the value of the move. Initialized to -2 and modified during the minimax algorithm.

5) **Del Num:** Store the number that have been used to perform the action. It’s used in the game algorithm to detect which num the computer has chosen to disable for the next moves the linked button.

6) **Op:** Store the operation that have been performed. Useful to know what the choice of the computer was.

Other data structures, defined in the Tkinter module, to create the UI.

IV. MAIN ALGORITHMS

1. State Space Graph Generation.

```
def create_graph(list, player, player_score, computer_score,gen_num,del_num=-1,op=""):
    prev=-1
    children=[]
    lenlist=len(list)
    for value in list:
        if value==prev: continue
        prev=value
        if(lenlist>=1):
            l = list.copy()
            l.remove(value)
            if  (player=="player"):
                gmult=create_graph(l,"computer", (player_score * value) % (gen_num),
computer_score,gen_num,value,"*")
                gsum=create_graph(l, "computer", (player_score + value) % (gen_num),
computer_score,gen_num,value,"+")

            else:
                gmult=create_graph(l, "player",player_score, (computer_score * value) %
(gen_num), gen_num,value,"*")
                gsum=create_graph(l, "player" ,player_score, (computer_score + value) %
(gen_num), gen_num,value,"+")
            children.append(gmult)
            children.append(gsum)
    return Graph((player_score,str(list),computer_score),player,children,del_num,op)
```

This algorithm takes 7 attributes:

→ **List:** Remaining numbers that can be used.

→ **Player:** Current player performing the move.

→ **The scores of the player and the computer**

→ **Gen Num:** The generated number to be as close as possible at the end of the game.

→ **Del Num and op:** The parameters of the move performed to arrive to that step.

This function iterates over the list, choosing each value one at a time and creating two new graphs for each chosen value. The two new graphs correspond to two different operations - multiplication and addition - that can be performed on the current player's score with the chosen value. The function then recursively calls itself with the updated list and the new scores and adds the resulting graphs as children of the current node. Finally, the function returns the graph that represent the complete game with all possible states represented.

2. Minimax Algorithm

```
def minimax(G,gen_num):
    if G.children==[]:
        if(G.val[0]==G.val[2]):
            G.heuristic_measure=0
            return 0
        elif(gen_num-G.val[0]>gen_num-G.val[2]):
            G.heuristic_measure = 1
            return 1
        else:
            G.heuristic_measure = -1
            return -1
    if(G.player=="computer"):
        max=minimax(G.children[0],gen_num)
        for child in G.children[1:]:
            val=minimax(child,gen_num)
            if val > max:
                max=val
        G.heuristic_measure=max
        return max
    else:
        min = minimax(G.children[0],gen_num)
        for child in G.children[1:]:
            val=minimax(child, gen_num)
            if val< min:
                min = val
        G.heuristic_measure = min
```

This algorithm takes 2 attributes:

→ **G:** The state space graph representing the game created by the previous function.

→ **Gen Num:** The number to be as close as possible as the end of the game.

The function implements the minimax algorithm for the graph structure defined previously. The function starts by checking if the graph has any children, which means it is not a leaf node. If the node is a leaf node, it assigns a heuristic measure to the node based on the difference between the player's score and the computer's score. If the score difference is 0, the measure is 0, meaning it's a tie. If the computer has a higher score, the measure is 1, meaning it's a win for the computer, and -1 otherwise.

If the node is not a leaf node, the function recursively calls itself on each child node and computes the heuristic measure for each child. If the current node represents the computer's turn, it selects the maximum heuristic measure from its children, and if it represents the player's turn, it selects the minimum. The final heuristic measure for the current node is then assigned to the maximum or minimum value computed, and the function returns it. This algorithm is used to make decisions on the best move for the computer in a two-player game.

3. Main Functions of the Game

```
def update_score(p,score):
    if(p=="player"):
        if(score<10): player_score_str.set("0"+str(score))
        else: player_score_str.set(str(score))
    else:
        if (score < 10): computer_score_str.set("0" + str(score))
        else: computer_score_str.set(str(score))

def delete_num(num):
    if(num==4):
        states[0]=DISABLED
        four_but.configure(state=DISABLED)
    elif(num==6):
        states[1]=DISABLED
        six_but.configure(state=DISABLED)
    elif(num==7):
        if(states[2]!=DISABLED):
            states[2]=DISABLED
            seven_but.configure(state=DISABLED)
        elif(states[3]!=DISABLED):
            states[3]=DISABLED
            seven_but_bis.configure(state=DISABLED)
    elif(num==8):
        states[4]==DISABLED
        eight_but.configure(state=DISABLED)
    else:
        states[5]=DISABLED
        nine_but.configure(state=DISABLED)

def end_game(player_score,computer_score,gen_num):
    description_label_str.set("")
    subdescription_label_str.set("")

    if (player_score==computer_score):
        result_label_str.set("This is a wonderful draw. What a
game this has been.")
    elif(gen_num-computer_score<gen_num-player_score):
        result_label_str.set("I'm sorry that you loose, but I'm
sure you will win the next game.")
    else:
        result_label_str.set("Congratulations, you win. We will
have to improve our computer intelligence.")
    for i in range(len(states)): states[i]=NORMAL
    btn_clicked(0)
    play.configure(state=NORMAL)
    mult.configure(state=DISABLED)
    sum.configure(state=DISABLED)
```

3.1. Update Score

The update_score function takes two arguments, p and score, where p is a string indicating whether the score to update belongs to the player or the computer, and score is the new score to set.

If p is equal to "player", the function updates the player's score by setting the value of the player_score_str StringVar. If score is less than 10, the score is formatted as a string with a leading 0, otherwise it is simply converted to a string.

If p is equal to "computer", the function updates the computer's score by setting the value of the computer_score_str StringVar. If score is less than 10, the score is formatted as a string with a leading 0, otherwise it is simply converted to a string.

3.2. Delete Num

The delete_num function takes an integer num as input and disables the button corresponding to that number in the GUI. The function also updates the state of the corresponding button in the states list to DISABLED.

The function first checks the value of num, and then uses a series of if and elif statements to determine which button to disable. For example, if num is equal to 4, it disables the four_but button and updates the state of the first element of states to DISABLED. If num is equal to 7, it checks the states of two buttons: seven_but and seven_but_bis. If the first button is not disabled, it disables that button and updates the state of the third element of states to DISABLED. If the first button is already disabled, it disables the second button and updates the state of the fourth element of states to DISABLED.

3.3. End Game

The end_game function takes three arguments, player_score, computer_score, and gen_num.

The function first sets the description_label_str and subdescription_label_str variables to empty strings. Then it checks if the player and computer scores are equal, in which case it sets the result_label_str variable to a message about a draw. If the computer's

score is closer to the generated number than the player's score, it sets result_label_str to a message about the player losing, otherwise it sets it to a message about the player winning.

Next, the function sets all the button states to NORMAL and calls btn_clicked(0) to reset the game. Finally, it disables the mult and sum buttons and enables the play button.

3.4. Launch Game

```
def launch_game():
    global chosen_num,chosen_op

    chosen_num = IntVar()
    chosen_op = StringVar()
    chosen_op.set("")
    initial_state = randint(6, 9)

    player = ("player" if messagebox.askquestion("Starting Player","Are
you gonna start ?") =="yes" else "computer")
    generated_number = randint(43, 67)
    description_label_str.set("The number that you should be as close as
possible is "+str(generated_number)+".")

    list_num = [4, 6, 7, 7, 8, 9]
    player_score = initial_state
    computer_score = initial_state
    g = create_graph([4,6,7,7,8,9], player, player_score, computer_score,
generated_number)
    minimax(g, generated_number)

    update_score("player",player_score)
    update_score("computer",computer_score)

    while len(list_num)!=0:
        if player=="player":
            subdescription_label_str.set("It's your turn, choose a number
below.")
            root.wait_variable(chosen_num)
            subdescription_label_str.set("Choose an operation to apply on
chosen number and score.")
            root.wait_variable(chosen_op)
            if (chosen_op.get() == "*"):
                player_score *= chosen_num.get()
            else:
                player_score += chosen_num.get()
            player_score %= (generated_number)
            update_score("player",player_score)
            list_num.remove(chosen_num.get())
            for item in g.children:
                if (player_score, str(list_num), computer_score) ==
item.val:
                    g = item
                    break
```

```
else:
            subdescription_label_str.set("The computer is thinking about its
next move.")
            root.after(5)
            index = 0
            for i in range(1, len(g.children)):
                if (g.children[i].val[0] == g.val[0] and
                        g.children[i].heuristic_measure >
g.children[index].heuristic_measure):
                    index = i
            g = g.children[index]
            new_list=eval(g.val[1])
            subdescription_label_str.set("It choose "+str(g.del_num)+" for
the number.")
            delete_num(g.del_num)
            subdescription_label_str.set("It choose " + str(g.op) + " for
the op.")
            computer_score = g.val[2]
            update_score("computer",computer_score)
            list_num = new_list
        if (player == "player"):player = "computer"
        else:player = "player"
    end_game(player_score,computer_score,generated_number)
```

The launch_game function is the function that is the core of the game.

Firstly, the function initializes some global variables named chosen_num and chosen_op, which are used later to store the chosen number and operation by the player. Then, the function generates an initial state for the game by choosing a random number between 6 and 9, inclusive.

Next, the player is prompted to decide who will start the game, either the human player or the computer. The game then generates a random number between 43 and 67, inclusive, and displays it as the target number that the players should try to reach.

The game then creates a list of numbers [4,6,7,7,8,9], which are the numbers that the players can choose from to perform operations. It also initializes the scores of both the player and the computer to be the initial state generated earlier.

The game then creates a state space graph from the game using the create_graph function The latter is used to calculate the best possible move for the computer player using the minimax algorithm.

After the initialization, the game enters a loop where each iteration represents a turn for either the player or the computer.

→ If it's the player's turn, they are prompted to choose a number from the list of numbers using the GUI interface. Then, they are prompted to choose an operation to apply to the chosen number and their current score. Depending on their choice of operation, the player's score is updated accordingly. The player's score is then reduced modulo the generated number to ensure that it doesn't exceed the target number.
The update_score function is then called to update the score on the GUI. The chosen number is removed from the list of numbers, and the current node of the graph is updated to reflect the player's move and thus the new current state of the game thanks to that code:

```
for item in g.children:
    if (player_score, str(list_num), computer_score) == item.val:
        g = item
        break
g = g.children[index]
```

→ If it's the computer's turn, the function uses the generated graph to choose the best possible move for the computer with the help of the following code:

```
for i in range(1, len(g.children)):
    if (g.children[i].val[0] == g.val[0] and
    g.children[i].heuristic_measure > g.children[index].heuristic_measure):
        index = i
    g = g.children[index]
```

Then, the computer updates the node (and thus the current state of the gale), its score and the GUI score thanks to the update_score function. The chosen number is then removed from the list of numbers.

After each turn, the function checks if it's the end of the game. If the game is over, the end_game function is called to determine the winner and display the result on the GUI.

3.5. Button Clicked

```
def btn_clicked(button=9):
    global chosen_num,chosen_op
    if button<7:
        mult.configure(state=NORMAL)
        sum.configure(state=NORMAL)
        four_but.configure(state=DISABLED)
        six_but.configure(state=DISABLED)
        seven_but.configure(state=DISABLED)
        seven_but_bis.configure(state=DISABLED)
        eight_but.configure(state=DISABLED)
        nine_but.configure(state=DISABLED)
        if button == 1:
            states[0] = DISABLED
            chosen_num.set(4)
        elif button == 2:
            states[1] = DISABLED
            chosen_num.set(6)
        elif button == 3:
            states[2] = DISABLED
            chosen_num.set(7)
        elif button == 4:
            states[3] = DISABLED
            chosen_num.set(7)
        elif button == 5:
            states[4] = DISABLED
            chosen_num.set(8)
        elif button == 6:
            states[5] = DISABLED
            chosen_num.set(9)
    else:
        mult.configure(state=DISABLED)
        sum.configure(state=DISABLED)
        four_but.configure(state=states[0])
        six_but.configure(state=states[1])
        seven_but.configure(state=states[2])
        seven_but_bis.configure(state=states[3])
        eight_but.configure(state=states[4])
        nine_but.configure(state=states[5])
        if button==7: chosen_op.set("*")
        elif button==8: chosen_op.set("+")
        elif button==9:
            play.configure(state=DISABLED)
            result_label_str.set("")
            launch_game()
```

The btn_clicked() function manages the clicks on the different buttons in the graphical user interface. When a button is clicked, it updates the state of the buttons and sets the value of chosen_num or chosen_op variables based on the button clicked. If the button clicked is a number button, it sets the chosen_num variable to the corresponding value and set the state of this button to disabled until the end of the game. Moreover, it disables all the number buttons and activate the operators' ones. If the button clicked is an operator button, it sets the chosen_op variable to the corresponding value and disables the operator buttons. If the btn_clicked() function is called with no arguments, it resets the GUI to its initial state and disables the play button, and then calls the launch_game() function to start a new game.

V. APPENDIX (CODE)

```
############################## IMPORT ###################################

from tkinter import *
from tkinter import messagebox
from random import randint
from PIL import ImageTk, Image
import pyglet

#########################################################################

############################## GRAPH CLASS ##############################

class Graph:
    def __init__(self,val,player,children=[],del_num=-1,op=""):
        self.val=val
        self.children=children
        self.player=player
        self.heuristic_measure=-2
        self.del_num=del_num
        self.op=op

#########################################################################

############################## GRAPH FUNCTIONS ##########################

def create_graph(list, player, player_score,
computer_score,gen_num,del_num=-1,op=""):
    prev=-1
    children=[]
    lenlist=len(list)
    for value in list:
        if value==prev: continue
        prev=value
        if(lenlist>=1):
            l = list.copy()
            l.remove(value)
            if  (player=="player"):
                gmult=create_graph(l,"computer", (player_score * value) %
(gen_num), computer_score,gen_num,value,"*")
                gsum=create_graph(l, "computer", (player_score + value) %
(gen_num), computer_score,gen_num,value,"+")

            else:
                gmult=create_graph(l, "player",player_score,
(computer_score * value) % (gen_num), gen_num,value,"*")
                gsum=create_graph(l, "player" ,player_score,
(computer_score + value) % (gen_num), gen_num,value,"+")
            children.append(gmult)
            children.append(gsum)
    return
Graph((player_score,str(list),computer_score),player,children,del_num,op)
```

```
def minimax(G,gen_num):
    if G.children==[]:
        if(G.val[0]==G.val[2]):
            G.heuristic_measure=0
            return 0
        elif(gen_num-G.val[0]>gen_num-G.val[2]):
            G.heuristic_measure = 1
            return 1
        else:
            G.heuristic_measure = -1
            return -1
    if(G.player=="computer"):
        max=minimax(G.children[0],gen_num)
        for child in G.children[1:]:
            val=minimax(child,gen_num)
            if val > max:
                max=val
        G.heuristic_measure=max
        return max
    else:
        min = minimax(G.children[0],gen_num)
        for child in G.children[1:]:
            val=minimax(child, gen_num)
            if val< min:
                min = val
        G.heuristic_measure = min
        return min

##############################################################################

############################ MENU FUNCTIONS ################################

def restart():
    end_game(-1,0,1)
    btn_clicked()

def helping():
    messagebox.showinfo("Rules",
                        """The rules of the game is to use the numbers to
increase our score to be as close as possible to the generated numbers when
it will have no numbers left. \n
The winner is the player who is the closest to the generated number at the
end. \n
Be aware do not reach the number because your score would be reset to 0. \n
You can exceed the gen_number but in this case a modulo will apply. \n
Good Luck...""")

##############################################################################

############################## GAME FUNCTIONS ################################

def update_score(p,score):
    if(p=="player"):
        if(score<10): player_score_str.set("0"+str(score))
        else: player_score_str.set(str(score))
    else:
        if (score < 10): computer_score_str.set("0" + str(score))
        else: computer_score_str.set(str(score))
```

```
def delete_num(num):
    if(num==4):
        states[0]=DISABLED
        four_but.configure(state=DISABLED)
    elif(num==6):
        states[1]=DISABLED
        six_but.configure(state=DISABLED)
    elif(num==7):
        if(states[2]!=DISABLED):
            states[2]=DISABLED
            seven_but.configure(state=DISABLED)
        elif(states[3]!=DISABLED):
            states[3]=DISABLED
            seven_but_bis.configure(state=DISABLED)
    elif(num==8):
        states[4]==DISABLED
        eight_but.configure(state=DISABLED)
    else:
        states[5]=DISABLED
        nine_but.configure(state=DISABLED)

def end_game(player_score,computer_score,gen_num):
    description_label_str.set("")
    subdescription_label_str.set("")

    if (player_score==computer_score):
        result_label_str.set("This is a wonderful draw. What a game this
has been.")
    elif(gen_num-computer_score<gen_num-player_score):
        result_label_str.set("I'm sorry that you loose, but I'm sure you
will win the next game.")
    else:
        result_label_str.set("Congratulations, you win. We will have to
improve our computer intelligence.")
    for i in range(len(states)): states[i]=NORMAL
    btn_clicked(0)
    play.configure(state=NORMAL)
    mult.configure(state=DISABLED)
    sum.configure(state=DISABLED)

def launch_game():
    global chosen_num,chosen_op

    chosen_num = IntVar()
    chosen_op = StringVar()
    chosen_op.set("")
    initial_state = randint(6, 9)

    player = ("player" if messagebox.askquestion("Starting Player","Are you
gonna start ?") =="yes" else "computer")
    generated_number = randint(43, 67)
    description_label_str.set("The number that you should be as close as
possible is "+str(generated_number)+".")

    list_num = [4, 6, 7, 7, 8, 9]
    player_score = initial_state
    computer_score = initial_state
    g = create_graph([4,6,7,7,8,9], player, player_score, computer_score,
generated_number)
    minimax(g, generated_number)
```

```
    update_score("player",player_score)
    update_score("computer",computer_score)

    while len(list_num)!=0:
        if player=="player":
            subdescription_label_str.set("It's your turn, choose a number
below.")
            root.wait_variable(chosen_num)
            subdescription_label_str.set("Choose an operation to apply on
chosen number and score.")
            root.wait_variable(chosen_op)
            if (chosen_op.get() == "*"):
                player_score *= chosen_num.get()
            else:
                player_score += chosen_num.get()
            player_score %= (generated_number)
            update_score("player",player_score)
            list_num.remove(chosen_num.get())
            for item in g.children:
                if (player_score, str(list_num), computer_score) ==
item.val:
                    g = item
                    break
        else:
            subdescription_label_str.set("The computer is thinking about
its next move.")
            root.after(5)
            index = 0
            for i in range(1, len(g.children)):
                if (g.children[i].val[0] == g.val[0] and
                        g.children[i].heuristic_measure >
g.children[index].heuristic_measure):
                    index = i
            g = g.children[index]
            new_list=eval(g.val[1])
            subdescription_label_str.set("It choose "+str(g.del_num)+" for
the number.")
            delete_num(g.del_num)
            subdescription_label_str.set("It choose " + str(g.op) + " for
the op.")
            computer_score = g.val[2]
            update_score("computer",computer_score)
            list_num = new_list
        if (player == "player"):player = "computer"
        else:player = "player"
    end_game(player_score,computer_score,generated_number)

##############################################################################

########################## INTERACTION GAME/GUI ##########################

def btn_clicked(button=9):
    global chosen_num,chosen_op
    if button<7:
        mult.configure(state=NORMAL)
        sum.configure(state=NORMAL)
        four_but.configure(state=DISABLED)
        six_but.configure(state=DISABLED)
        seven_but.configure(state=DISABLED)
        seven_but_bis.configure(state=DISABLED)
        eight_but.configure(state=DISABLED)
```

```
        nine_but.configure(state=DISABLED)
        if button == 1:
            states[0] = DISABLED
            chosen_num.set(4)
        elif button == 2:
            states[1] = DISABLED
            chosen_num.set(6)
        elif button == 3:
            states[2] = DISABLED
            chosen_num.set(7)
        elif button == 4:
            states[3] = DISABLED
            chosen_num.set(7)
        elif button == 5:
            states[4] = DISABLED
            chosen_num.set(8)
        elif button == 6:
            states[5] = DISABLED
            chosen_num.set(9)
    else:
        mult.configure(state=DISABLED)
        sum.configure(state=DISABLED)
        four_but.configure(state=states[0])
        six_but.configure(state=states[1])
        seven_but.configure(state=states[2])
        seven_but_bis.configure(state=states[3])
        eight_but.configure(state=states[4])
        nine_but.configure(state=states[5])
        if button==7:
            chosen_op.set("*")
        elif button==8:
            chosen_op.set("+")
        elif button==9:
            play.configure(state=DISABLED)
            result_label_str.set("")
            launch_game()

##########################################################################


####################### DEFINING GUI INTERFACE ###########################

pyglet.font.add_file("Imprima-Regular.ttf")

root = Tk()
root.geometry("531x447")
root.configure(bg = "#f0f0f0")
root.resizable(width=False,height=False)
root.title("Don't Reach That Number")
root.iconbitmap("icon.ico")

#Menu creation
menubar=Menu(root)
menubar.add_cascade(label="Play Another Game",command=restart)
menubar.add_command(label="Rules", command=helping)
menubar.add_cascade(label="Close",command=root.destroy)
root.configure(menu=menubar)

openbg=Image.open("background.png")
bg=ImageTk.PhotoImage(openbg)
labelbg=Label(root,image=bg)
```

```
labelbg.place(x=12,y=79)

states=[NORMAL]*7
files=["4.png","6.png","7.png","8.png",
       "9.png","mult.png","sum.png","play.png"]
images = []
for file in files:
    img = Image.open(file)
    photo = ImageTk.PhotoImage(img)
    images.append(photo)

player_score_str=StringVar()
player_score_str.set("00")
computer_score_str=StringVar()
computer_score_str.set("00")
description_label_str=StringVar()
subdescription_label_str=StringVar()
result_label_str=StringVar()


four_but = Button(image = images[0], state=DISABLED, borderwidth = 0,
highlightthickness = 0, command = lambda:btn_clicked(1), relief = "flat")
four_but.place(x = 157, y = 127, width = 33, height = 33)

six_but = Button(image = images[1], state=DISABLED, borderwidth = 0,
highlightthickness = 0, command = lambda:btn_clicked(2), relief = "flat")
six_but.place(x = 208, y = 128, width = 33, height = 33)

seven_but = Button(image = images[2], state=DISABLED, borderwidth = 0,
highlightthickness = 0, command = lambda:btn_clicked(3), relief = "flat")
seven_but.place(x = 208, y = 172, width = 33, height = 33)

seven_but_bis= Button(image = images[2], state=DISABLED, borderwidth = 0,
highlightthickness = 0, command = lambda:btn_clicked(4),  relief = "flat")
seven_but_bis.place(x = 283, y = 172, width = 33, height = 33)

eight_but = Button(image = images[3], state=DISABLED, borderwidth = 0,
highlightthickness = 0, command = lambda:btn_clicked(5), relief = "flat")

eight_but.place(x = 283, y = 128, width = 33, height = 33)

nine_but = Button(image = images[4], state=DISABLED,borderwidth = 0,
highlightthickness = 0, command = lambda:btn_clicked(6), relief = "flat")
nine_but.place(x = 334, y = 128, width = 33, height = 33)

mult = Button(image = images[5], state=DISABLED,borderwidth = 0,
highlightthickness = 0, command = lambda:btn_clicked(7), relief = "flat")
mult.place(x = 208, y = 258, width = 33, height = 33)

sum = Button(image = images[6], state=DISABLED,borderwidth = 0,
highlightthickness = 0, command = lambda:btn_clicked(8), relief = "flat")
sum.place(x = 283, y = 258, width = 33, height = 33)

play = Button(image = images[7], borderwidth = 0, highlightthickness = 0,
command = lambda:btn_clicked(),relief = "flat")
play.place(x = 124, y = 324, width = 282, height = 70)

# Create and pack the first label
description_label = Label(root,
textvariable=description_label_str,font=("Imprima-Regular",12))
description_label.pack(side=TOP, pady=10)
```

```
# Create and pack the second label
subdescription_label = Label(root, textvariable=subdescription_label_str,
font=("Imprima-Regular",12))
subdescription_label.pack(side=TOP)

#Create and pack the player_score label
labelplayer = Label(root,
bg="#d9d9d9",textvariable=player_score_str,font=("Imprima-Regular",48))
labelplayer.place(x=40,y=115)

# Create and pack the computer_score label
labelcomputer = Label(root, bg="#d9d9d9",textvariable=computer_score_str,
font=("Imprima-Regular",48))
labelcomputer.place(x=413,y=115)

# Create and pack the first label
description_label = Label(root,
textvariable=result_label_str,font=("Imprima-Regular",12))
description_label.pack(side=BOTTOM, pady=10)

###############################################################################

#Launching the GUI
root.mainloop()
```