The Macalester Hangman Game

Brought to you by Aram Petrosyan and Karim Amra

Core Concepts in Computer Science

December 2023, St. Paul

Introduction

Welcome to The Macalester Hangman Game!

We re-created the classic hangman game in the context of Macalester - as our project for the class of Core Concepts in Computer Science. The classic hangman game is a favorite of many people, just like Macalester College! So the combination of those two, we thought, must be amazing!
In this report, we'll go through two main sections. The first is a user manual written for an average computer user, who's not necessarily a computer science student or professional. The second section is a deeper discussion of the program itself - we'll discuss the structure of our program, how we tested it and made sure the game works properly. At the end of the report - we'll also talk about possible future extensions and expansions of the program.
Thank you.

User's manual

You have the "hangman" folder at your disposal. This folder contains 4 other folders - the ".idea", "_pycache_", "draft_code_files" and "paintings". You do not need to interact with these folders at all - they're for the developers' use.

- *Note: If you have the time and the desire for a more personalized experience, you could go into the "paintings" folder, and choose one of the three folders inside it, count how many pictures they have - delete them and add your own pictures with the same quantity, making sure they are all named by numbers starting with 0 and they're all in the JPG format.*

In the same "hangman" parental folder, you also have two python files, called "imageTools.py" and "final_code.py". You do not need to have a Python virtual environment like PyCharm or Visual Studio Code to open them and make changes, but you do need to make sure that your computer has Python installed, so that when you click "final_code.py", your computer will be able to run the program. You can download the latest version of Python from the official website. To play the game, as mentioned, you'll just need to click the "final_code.py" file. You do not need to interact with "imageTools.py" since it's just a python file needed to assist and run the game in the "final_code.py". If you're clicking "final_code" and it launches a text-editor file, you should right-click on it, go to "Open with" and make sure to open it with the Python Launcher and not something else. Figure 1 displays what you'll see once the program is launched in a Windows operating system:

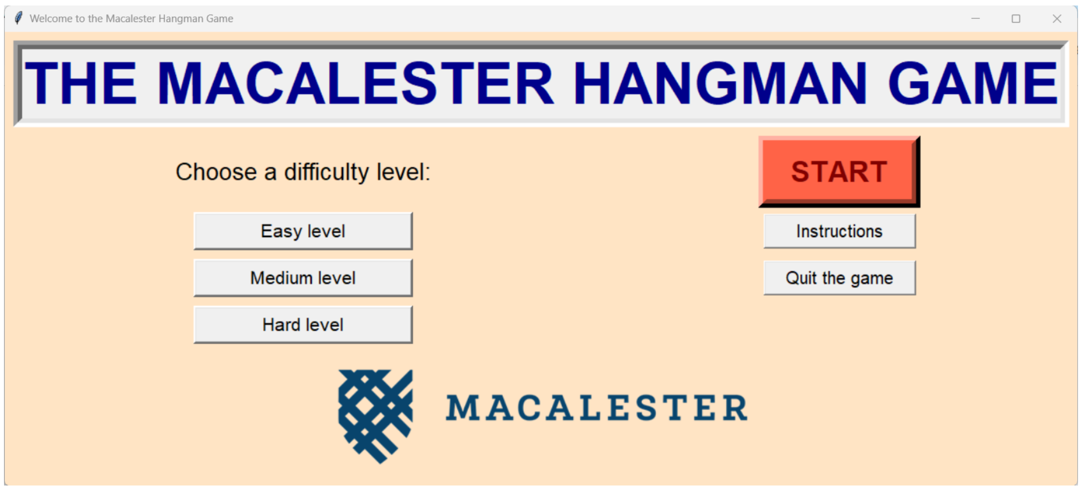
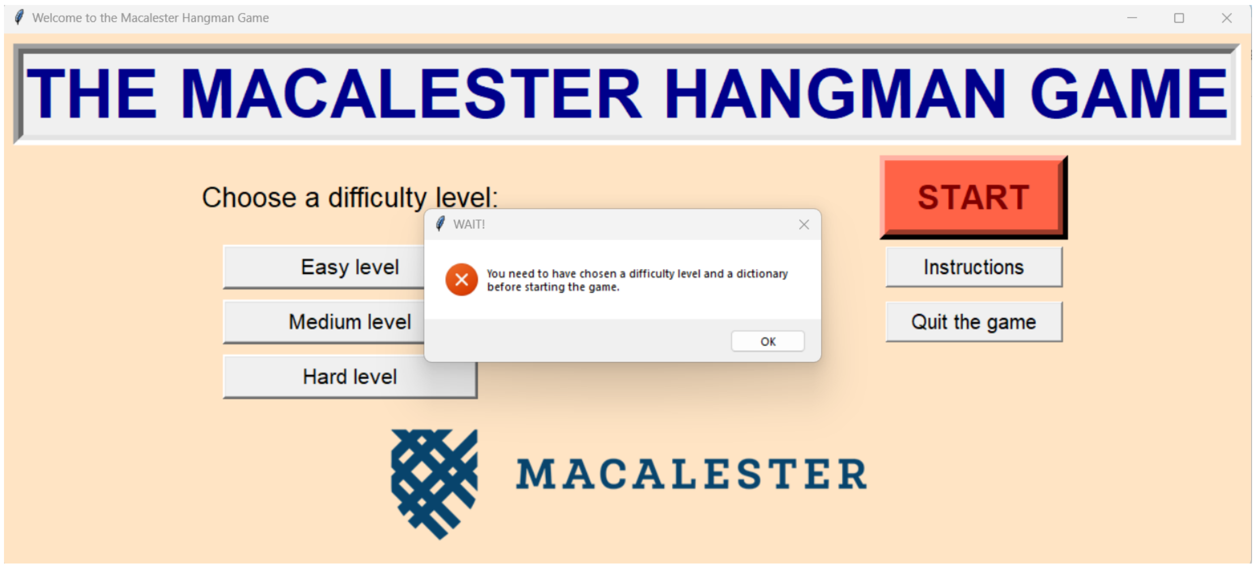

Welcome to the Macalester Hangman Game
THE MACALESTER HANGMAN GAME
Choose a difficulty level:
START
Easy level
Instructions
Medium level
Quit the game
Hard level
MACALESTER

Figure 1. The Main Menu of The Macalester Hangman Game at its initially launched stage.

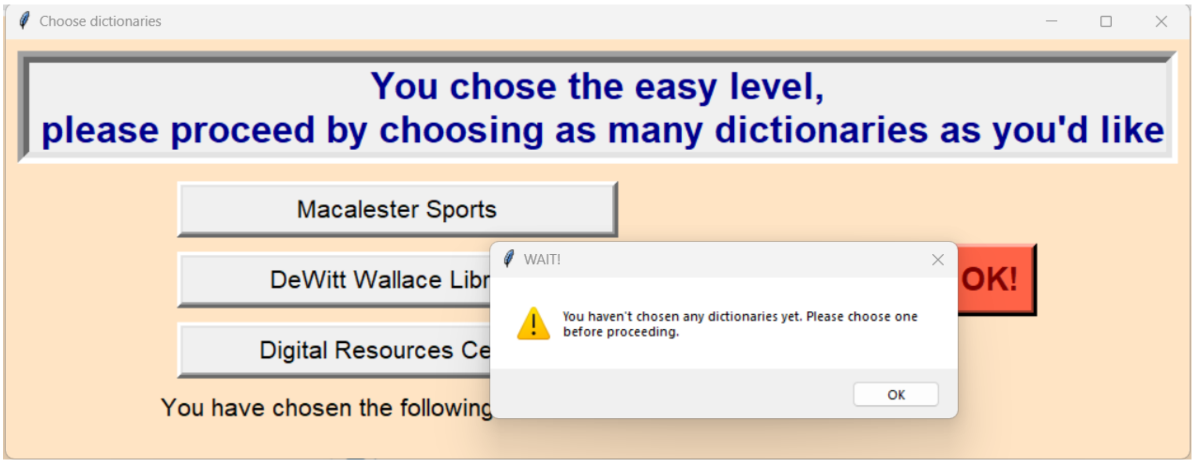
Make sure to click the “Instructions” button if you’re unsure about the rules of the classic hangman game. Before clicking the “START” button you’ll need to choose a difficulty level and a dictionary. In fact, if you try starting the game without doing the previously mentioned - a warning message will pop up asking you to configure the necessary settings of the game, as displayed in Figure 2. This warning message is not an error of the program and you should not be worried - just follow the instructions:


Welcome to the Macalester Hangman Game
THE MACALESTER HANGMAN GAME
Choose a difficulty level:
START
Easy level
Medium level
Hard level
Instructions
Quit the game
WAIT!
OK
MACALESTER

You need to have chosen a difficulty level and a dictionary before starting the game.

Figure 2. The warning message after an attempt of starting the game without configuring settings for it.

You can click any of the difficulty level buttons and you’ll get a similar window which will ask you to choose a dictionary or multiple dictionaries by clicking on their names. If you wish to remove one of the dictionaries you clicked - you can simply click on it again. You’ll have to choose at least one dictionary before clicking the “OK!” button - otherwise you’ll get a warning message like the one below. Again, this is not an error message of the program and you shouldn’t worry - just follow the instructions stated on the message.


Choose dictionaries

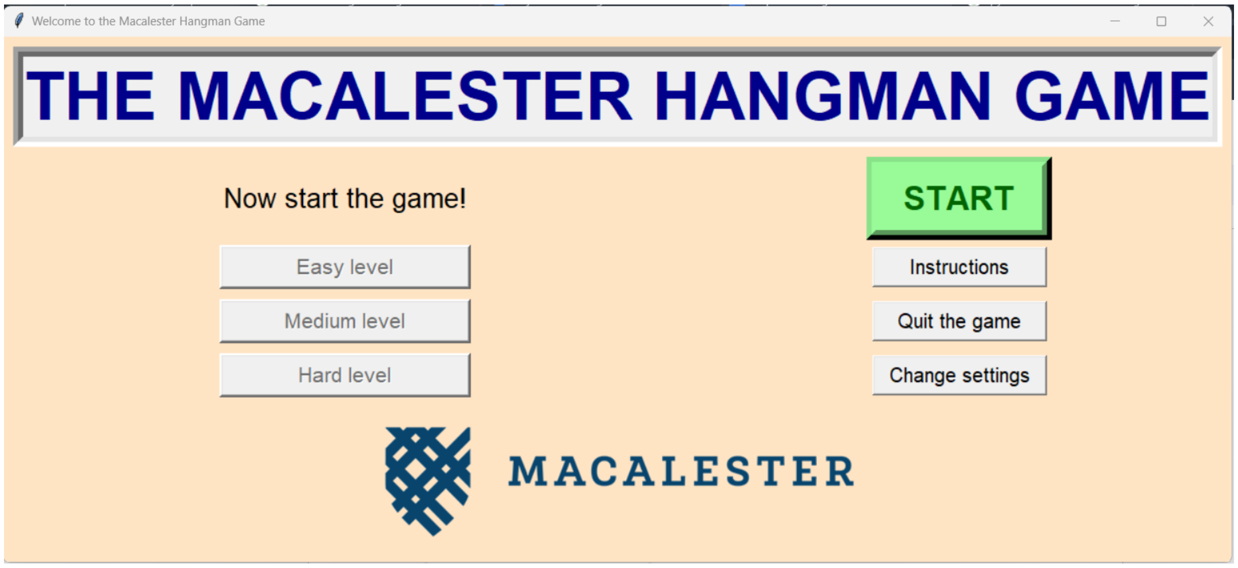
You chose the easy level,
please proceed by choosing as many dictionaries as you'd like

WAIT!

You haven't chosen any dictionaries yet. Please choose one before proceeding.

Figure 3. The warning message after an attempt of clicking "OK!" without choosing a dictionary.

You'll also notice that the "OK!" button turns green whenever it's valid to click it! After successfully configuring the settings of the game (meaning - choosing the difficulty level and the dictionaries), you'll notice that the Main Menu is a little changed than what it was initially. The "START" button is now green, and there's also a new button called "Change Settings" which basically restarts the game and allows you to configure new settings if you think you don't like your already selected choices. You can also notice that the buttons for the difficulty levels are disabled - this is done to avoid players choosing various settings at the same time and messing with the proper flow of the game. Figure 4 displays all the above mentioned changes:


Welcome to the Macalester Hangman Game

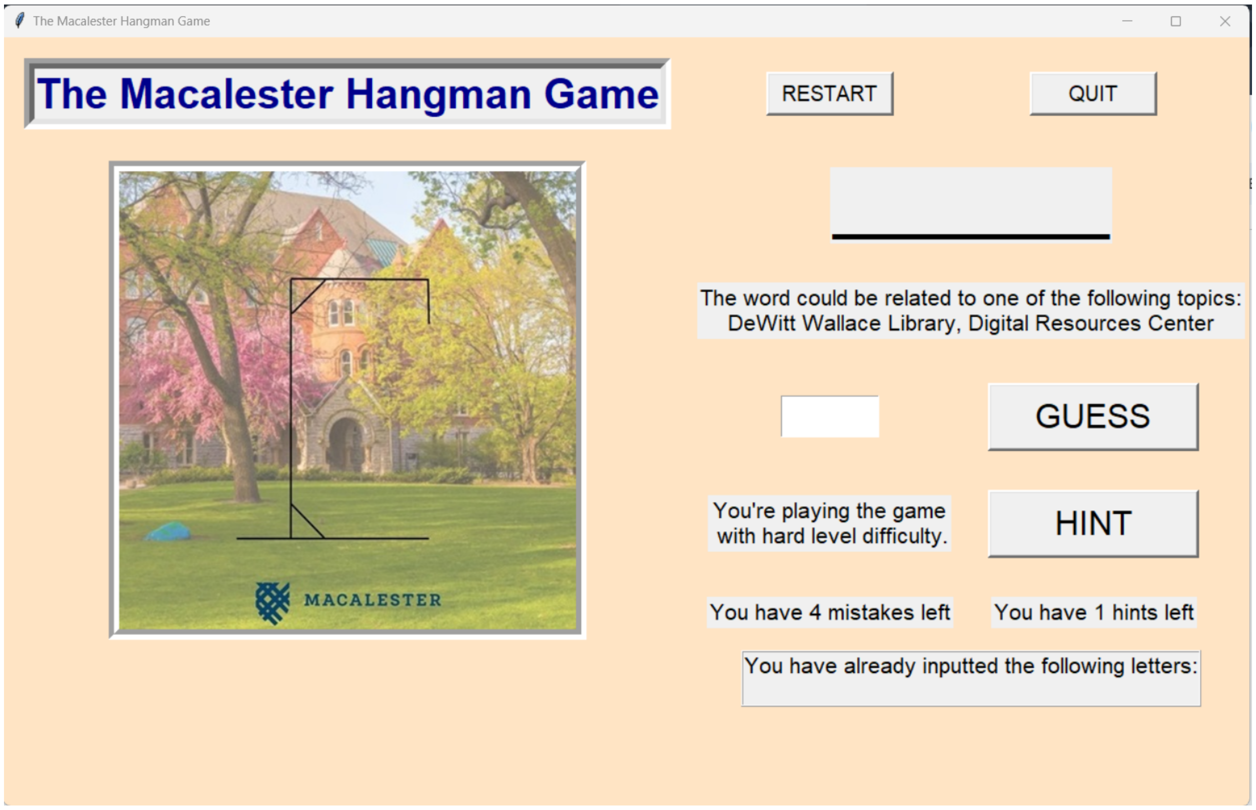
THE MACALESTER HANGMAN GAME

Now start the game!

Figure 4. The Main Menu after having configured settings for the game.

- *Note: Please don't use the Red X button at the right top of the splash windows throughout the game. This can result in semantic errors and prevent you from the proper gaming experience.*

You're set! Figure 5 shows your screen after starting the game, if you, for example, chose the hard level difficulty and the dictionaries about Macalester's Library and Digital Resources Center. On the screen you can see various texts explaining what dictionaries you have chosen, what level of difficulty you're playing with, how many mistakes you have left, how many hints you have left and also what letters you have already inputted. The main "tools" of the game are obviously the entry widget where you can input letters and the "GUESS" button which you can click to send your inputted letter to the program.



The word could be related to one of the following topics: DeWitt Wallace Library, Digital Resources Center

You're playing the game with hard level difficulty.

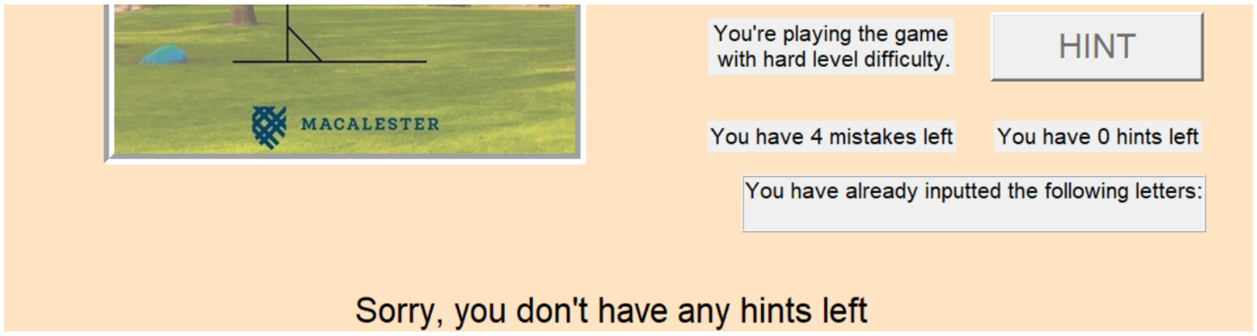
You have 4 mistakes left

You have 1 hints left

You have already inputted the following letters:

Figure 5. The very initial state of the screen, if the user has, for example, chosen the hard level and the dictionaries about Macalester's DRC and Library.

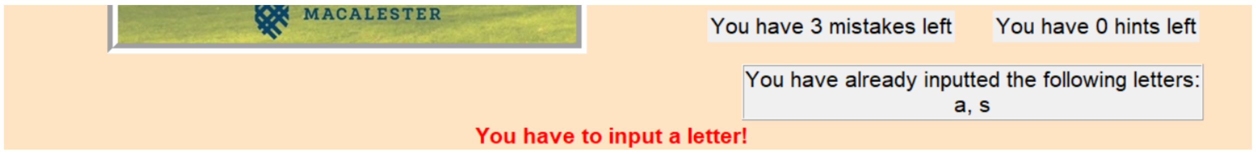
The "HINT" button is also useful for requesting hints, but it'll be disabled if you attempt requesting a hint after having run out of your allowable hints based on the difficulty level you chose. In case that ever happened, you'll also get a message indicating that you don't have any hints left, as displayed in Figure 6.



You're playing the game with hard level difficulty.
You have 4 mistakes left
You have 0 hints left
You have already inputted the following letters:
Sorry, you don't have any hints left

Figure 6. At the bottom of the screen, you'll get a message about not having any hints left if you attempt to request one while having used up all your available hints.

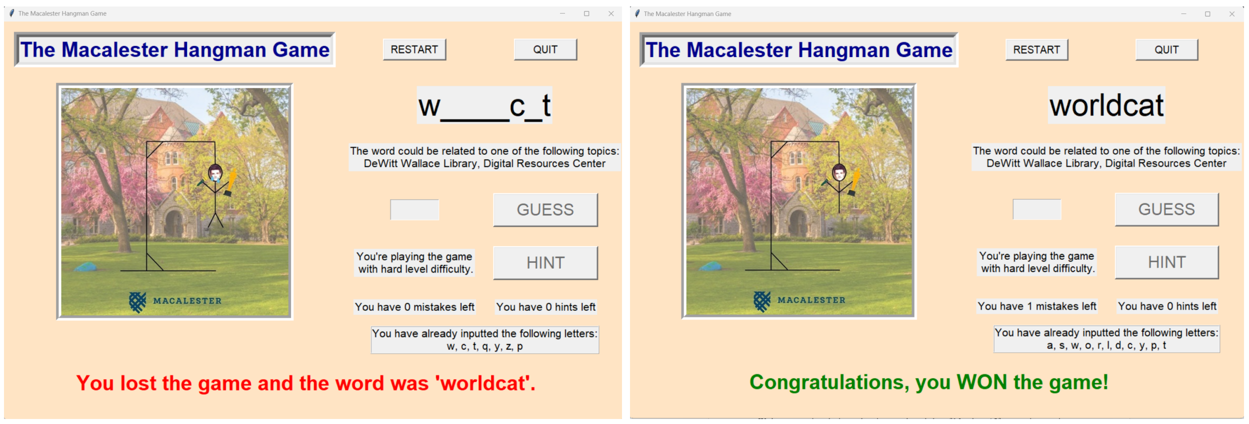
Throughout the game, if your input is invalid (e.g. you tried inputting a letter you've previously tried, or a number, or some sort of punctuation sign, etc.) - you'll get a warning message explaining why you can't input it, as displayed in Figure 7.



You have 3 mistakes left
You have 0 hints left
You have already inputted the following letters:
a, s
You have to input a letter!

Figure 7: You could get a red bold warning message telling you that you have to input a letter, if, for example, you attempt to input a number, a punctuation mark, or a special character.

Finally, you'll get a red message announcing your loss, as well as the word you were trying to guess, or a green message announcing your victory, as displayed in, respectively, the left and right sides of Figure 8.



w_____c_t

The word could be related to one of the following topics:
DeWitt Wallace Library, Digital Resources Center

You're playing the game with hard level difficulty.

You have 0 mistakes left

You have 0 hints left

You have already inputted the following letters:
w, c, t, q, y, z, p

You lost the game and the word was 'worldcat'.

worldcat

The word could be related to one of the following topics:
DeWitt Wallace Library, Digital Resources Center

You're playing the game with hard level difficulty.

You have 1 mistakes left

You have 0 hints left

You have already inputted the following letters:
a, s, w, o, r, l, d, c, y, p, t

Congratulations, you WON the game!

Figure 8: Left-side screen demonstrates when you lose the game, and the right-side screen demonstrates the announcement of your victory!

After the end of the game, and, in fact, during the game you have the opportunity to click either the "RESTART" or "QUIT" buttons, and, respectively restart the whole game again - having to configure settings too, and quit the whole game with its program.

We hope you enjoy playing and testing your knowledge of Macalester! If you run into any issues, semantic errors then please contact us through our emails, which can be found in the Instructions in the Main Menu.

Discussion of the program

- Summary of the program

To be fair, we're not completely happy with our program code. It spans across more than a thousand lines, and if we were more efficient, we could have had less use of lines. To be more specific, certain very similar functions were repeated and we didn't manage to find a way of iterating them through a loop, especially since we realized our code's inefficient length at the end of the project and didn't have much time to explore the iterating options. Since the game was working completely fine as well , we weren't incentivized to put effort into incorporating efficient loops and iterations in the code. Still, we do realize that if we decide to develop the game further in the future, we'll have to come up with an effective loop system and the reasons will be explained after we discuss the structure of the program.

The code starts with import statements that are all needed to execute the program. These include the importing of tkinter, random, PhotoImage (from tkinter), PIL, PIL.Image, PIL.ImageTk, messagebox (from tkinter), and string. Some of these were provided by the professor during office hours to help us display images in the Graphical User Interface we created. All these images were manually designed by using "Canva" browser software.

After the import statements, we have the "hangmanGUI" class definition. In this class we have a constructor function and 28 other functions that all are needed to run the constructor.

The constructor function starts with the database of dictionaries - three dictionaries for Macalester Sports, Library and Digital Resources Center. Each dictionary has words as keys and hints for those words as values for the keys. We also define lists of the keys for the purpose of making the random choice of a word more straightforward.

Following the dictionaries, we have the portion of the code where we initialize all needed data and variables. This portion isn't, in fact, showing all the "initializers", because many tkinter label texts are defined later. A complete portion of all initializers needed for the program can later be found in the "restartResponse" function.

- *Note: You may notice that "self.all_dictionaries" is initialized with ["hm"]. "hm" is just a random word we put (it was actually a result of loudly thinking "hmm" and spontaneously assigning a string with whatever came to our mind). If we initialize this by just "self.all_dictionaries=[]" then we'll run into errors later in the program indicating that the list is empty and taking something from it isn't possible. We understand that there probably was a better solution than just assigning a "hm" for the sake of the list not* *initially being empty (and then getting rid of "hm" when appending with necessary elements). In the future, if we want to make the game better - we need to make sure we have a solution that's more efficient and less error-prone.*

We then start the construction of the main window. This is going to be the actual game window, and not the Main Menu (this will be a splash screen instead). There's no particular reason for this that we can give, but we simply went that way because in the very beginning of writing the code, we didn't have a Main Menu and created the simple version of the game in tkinter's main window. Right after defining the "mainWin" and giving it a title, we then define the hangman images which will be used in all levels . The defined images are put in 3 lists - one for hard, medium and easy difficulty levels. Later in the game, if, say, the number of mistakes is 2, then the 2nd element of the list (or, basically, the third image contained in the list) will be displayed on the screen. And the third image in the folder is just a continuation of the evolving hangman figure from its preceding image. You can see the images in the "paintings" folder.

After this - everything is quite straightforward - components of widgets for both the main window and splash window screens are constructed. We made sure to write comments so that it's clear which code is for which component. The last part of the constructor function consists of splash screens for choosing dictionaries after having clicked one of the "Easy", "Medium" or "Hard" buttons. This part is quite repetitive and is one of the portions of the code that we wish we managed to somehow iterate through a loop system.

Finally we get to the part of the "hangmanGUI" class where the functions are defined. 20 out of the 28 functions are all binded to some buttons - many of them are repetitive, e.g. "hard_clicked_ok", "easy_clicked_ok", "medium_clicked_ok". All these button response functions are put before the 8 other functions that run the game. One of these functions, the "restartResponse" function is notable due its size but also its simplicity. We were trying to figure out a way to restart the game with a button by destroying everything and "undestroying" the initial constructed things back, but there was no "undestroy" method. Luckily, the solution was simple enough, we just took everything in the game, lists, numbers, buttons, labels , and put them in their initial states again line by line. This function also has a use of the "grid_remove()" method which we hadn't learned in class, we found out about this method in the NMTech Tkinter reference (1) we were provided in class.The purpose of this is that after restarting the game, the "Change Settings" button won't be visible. Its grid will be forgotten and reconstructed again whenever an "OK!" button is pressed while configuring settings.

The 8 last functions are where the actual process for the hangman game goes on.
First, we have a "choosing_word" function which makes use of the random module, chooses a random word from the dictionaries the user has chosen, and returns the word but its letters converted into indices.
Second, the "updating_hangman" function consists of several conditional statements that check which difficulty level is selected and thus use its image list to update the image displayed on the screen based on the number of mistakes made.
Third, the "validInput" function returns a Boolean value which is only True if the preceding tests are passed - those tests make sure the user's input isn't longer than one character, isn't something that's not a letter, isn't empty and hasn't been attempted before.
The fourth, "checking" function starts off by asserting that the returned Boolean value from "validInput" is True before moving onto the rest of its operations, which make sure certain

changes to the hangman image, the other labels and initialized variables happen for both a mistaken attempt and a correct attempt.

The fifth function is the “ending” function, which is called in the previous function as well. Here some buttons get disabled, and based on whether the game was won or lost - certain messages appear on the screen.

The sixth function, in our opinion, has the largest inefficiency in it, because it practically limits us from adding more dictionaries to the game without having to add dozens of lines of code for each dictionary in this function called “hinting”. We had multiple “if, else, elif” statements that manipulated the hinting label and the other associated accumulator variables based on which difficulty level and which dictionary were selected by the user. Everything in this function is quite repetitive so we believe it’d be possible to come up with a helper iterator function, which would give us the opportunity to add more dictionaries to our initial database. We wouldn’t be worried about the “new dozens of lines of code” as a result of new dictionaries - because we can always put those in another file and import the file as a database in our current file.

The seventh function - “cleanString” - is a recursive function, which we’re very happy we managed to incorporate in our program, especially since we had made the plan for the code earlier than we learnt about recursive functions in class. This function wasn’t necessary, in essence, but it’s called for lots of labels which display lists as text - it cleans punctuation in such a way that commas are still left in the string. This makes many labels on various screens look cleaner and more organized to the viewer’s eye.

Finally, the last, eighth function, “go”, is the simplest of all - just runs the whole code using “mainloop()”.

It’s important to note that all these functions described above are actually methods outside the class of hangmanGUI. At the end of all code - we simply create an instance of the hangmanGUI and call its “go” method to run the game.

- Implementation and testing of the program

We did not have a separate testing file, instead we chose the unit-testing strategy, meaning we would check small portions of code as we went on building the program - fix all bugs, and move to the next stage only if everything is working. This type of testing, though, did not guarantee the absence of semantic errors. We decided that we’d “hunt” for semantic errors through testing the game with several friends of ours. Asking friends to play our game is a great strategy for finding semantic errors, because people who are unfamiliar with the code and the program may take actions in the game that we - the developers - would never take, because we have a certain mental image of the game and its process which could unconsciously stop us from taking unique actions. This was, in fact, successful, for example, one of our friends tried to restart the game after having played a game already, and the hangman image wasn't updated to its initial stage - we hadn’t noticed because we were focussed on the labels more.

Unit testing was, in our opinion, quite efficient against non-semantic errors, such as typos, ValueErrors, SyntaxErrors, NameErrors, etc. The program itself had a function “validInput” which would in essence, test the user’s inputs and make sure that it was compatible with the game. It would return “False” every time the user’s input was invalid (for example, a number, instead of a letter, or a repeated letter), and then another function would make use of this “validInput” tester function, by asserting that the return isn’t False before proceeding.

The print() was a major player in our testing process. We used it in various functions, especially those dealing with dictionaries, lists and randomly choosing words - to figure out what exactly is happening to the lists in the Python console. This helped the debugging process a lot, since print() could visually showcase what exactly function outputs' problems were.
Every time we finished a major part of the game, we'd save the file, copy the whole working code - put it into a new file and start working for the next stage in the new file. This was important to avoid losing track of successful codes, losing large amounts of time rewriting code and sustaining good energy. Finally unit testing seems to have an advantage even when it comes to our own mental health - little wins and successful debugs over a long time helped us stay excited and energized, while if we decided to test everything at the end - we could've lost a lot of energy and mood for staying engaged in the project.

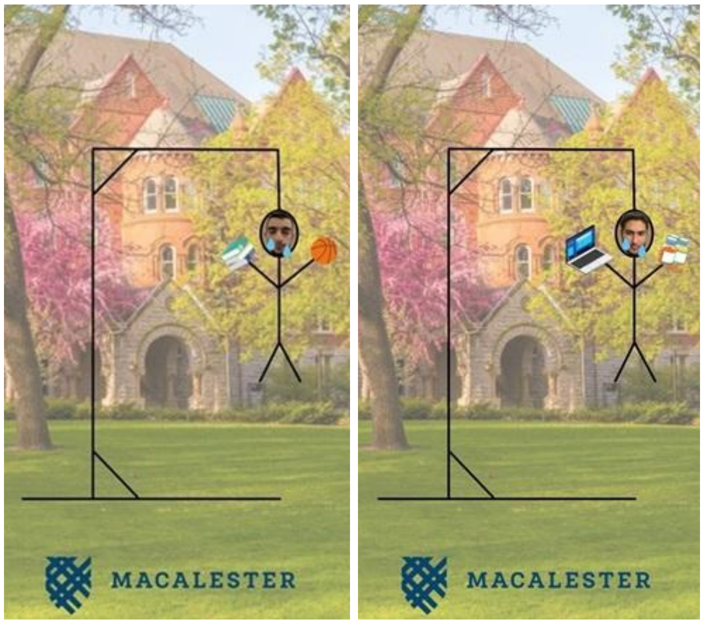
Conclusion

In summary, we created a Python game with tkinter which works well and without errors. The only difficulties that we couldn't overcome were - we couldn't find a way to bind the "entry" key on the keyboard to the "GUESS" button, we'd run into an error everytime we tried binding them. The other difficulty was that if the user closes the splash screens with the red X button in their top right corner - there won't be the same processes in the program as in the case of clicking a "QUIT" or "OK!" button and this could result in some errors. In the future, we would work on these difficulties, and extend the vocabulary base of the game by also increasing the number of dictionaries. For that to be possible, we'll have to find ways of making the code more iterative and efficient, so we won't need to add dozens of lines of repetitive code for each additional dictionary.

Finally, talking about the future, if this game is well developed - we think it could be a great addition to the New Student Orientation at Macalester. Incoming students could play it competitively either to learn about many new Macalester related topics and words, or to check their knowledge after the end of the orientation!

We learned a lot from this project - not only we practiced a great deal of coding with tkinter, functions and various types of data, but we also learned how to write code collaboratively. This is a skill that can't be easily achieved if one tries to learn programming at home - with a book or an online programming platform. The planning phase of the project was also extremely helpful - we realized the importance of designing a project before starting it, writing out functions in English-based "pseudo-code" and thus setting a detailed path to a goal. Though writing the actual code, it did differ from its original plan - we'd never be able to collaborate effectively if we didn't have a plan.

Thank you.



Aram and Karim!

References

(1) Roseman, M. (n.d.). *The grid geometry manager*. TkDocs Tutorial - The Grid Geometry Manager. https://tkdocs.com/tutorial/grid.html