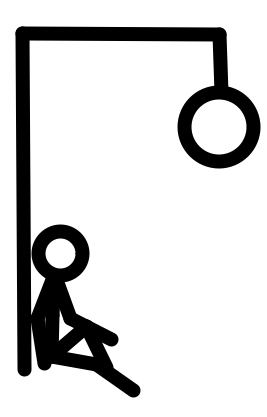
Steven Hernandez

Hangman



# Repository and Videos

This project is located on the GitHub repository. To access it and its files, got to:

[https:/github.com/shernand1273/hangman](https://github.com/shernand1273/hangman)

To access my GitHub repository home page:

<https://github.com/shernand1273>

Video Walk-through (DEMO): <https://bit.ly/2PggEtY> (shortened link)

<https://onedrive.live.com/?cid=95AED8A5FA63E878&id=95AED8A5FA63E878%214228&parId=95AED8A5FA63E878%21140&o=OneUp>

Architecture Walkthrough: <https://bit.ly/2MiJnRb> (shortened link)

<https://onedrive.live.com/?cid=95AED8A5FA63E878&id=95AED8A5FA63E878%214229&parId=95AED8A5FA63E878%21140&o=OneUp>

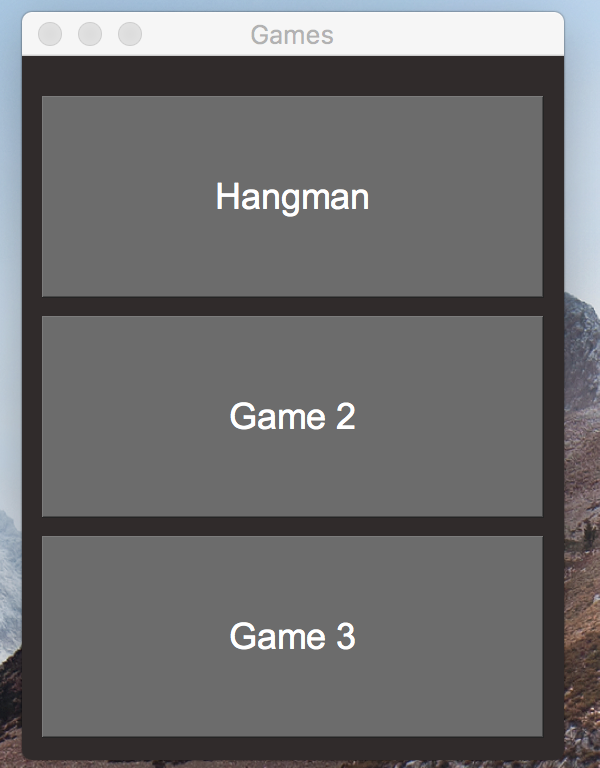
# Executive Summary (To do)

The name of the game is Hangman. The objective is for the user to guess a randomly selected word letter by letter before he/she runs out of tries. The traditional way of playing the game is to draw a stick figure piece by piece as the user fails to correctly guess a letter. Instead of adopting this method, this game will not draw a hanging man piece by piece. A customized animation will show every time the user either guesses wrong or right. A right guess will not trigger an animation, but you will see the guess field in the user interface updated to show the letter. If the user guesses wrong, a .gif animation will be triggered and shown in the main window. As the user continues to guess wrong, the .gif animations will show the character getting closer to the rope. When the user fails all guesses, the system will display the last .gif animation, which shows the character hanging. Should the user win the game (guess all letters), system will display an animated gif “you win” on the user interface window.

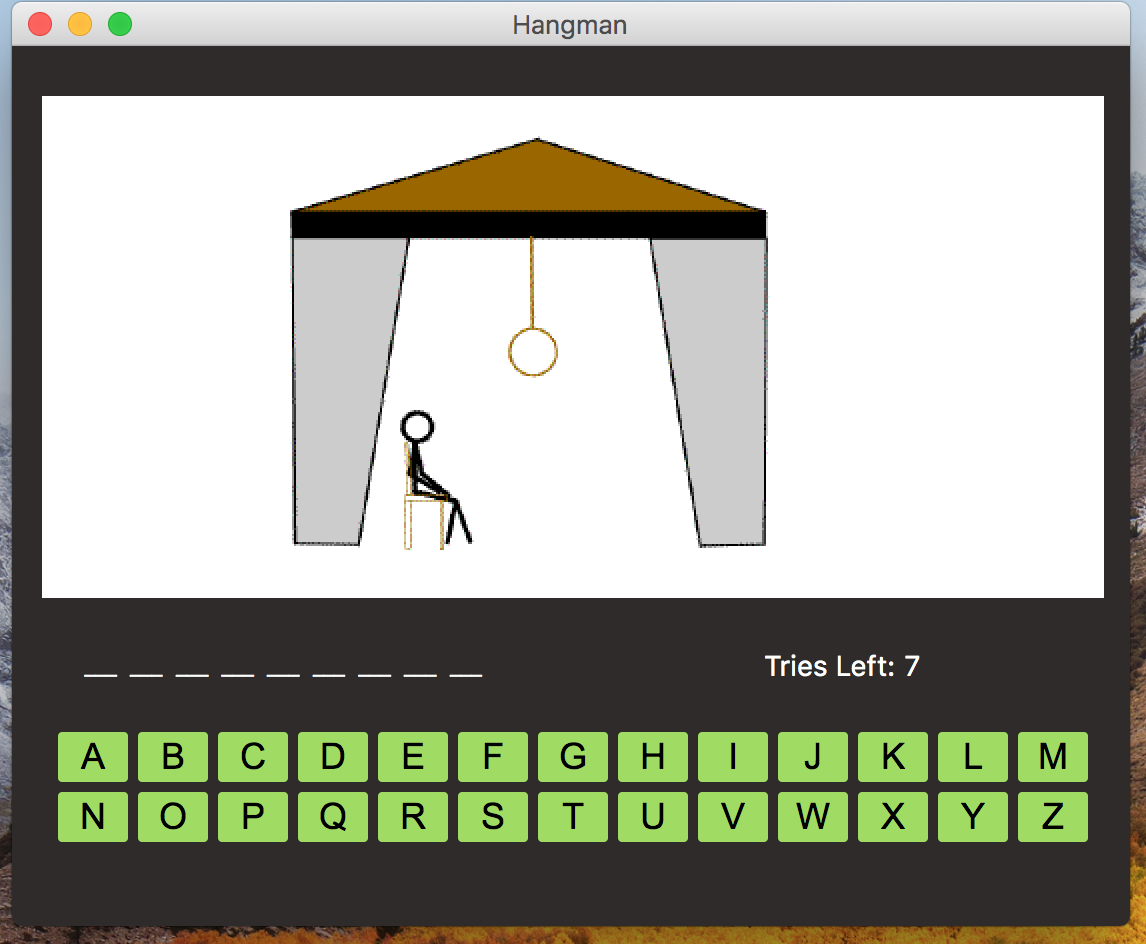
This game was designed to limit user input error. The design eliminated the chance of the user crashing the system by entering the wrong input (i.e. number, symbol). This game does not give the chance to enter a value, instead it presents the user with a series of preprogrammed buttons to handle the input. Each button triggers a test case. Based on this test case, the game is updated accordingly. Because the game was designed with kids ranging from 5-10 in mind, the use of only buttons makes it easy for them to interact with the system. Finally, the design of this game doesn’t require the user to touch his/her keyboard. Therefore, the user will have to rely on using a mouse or trackpad to interact with the system.

# Scenario | Starting and selecting the Game

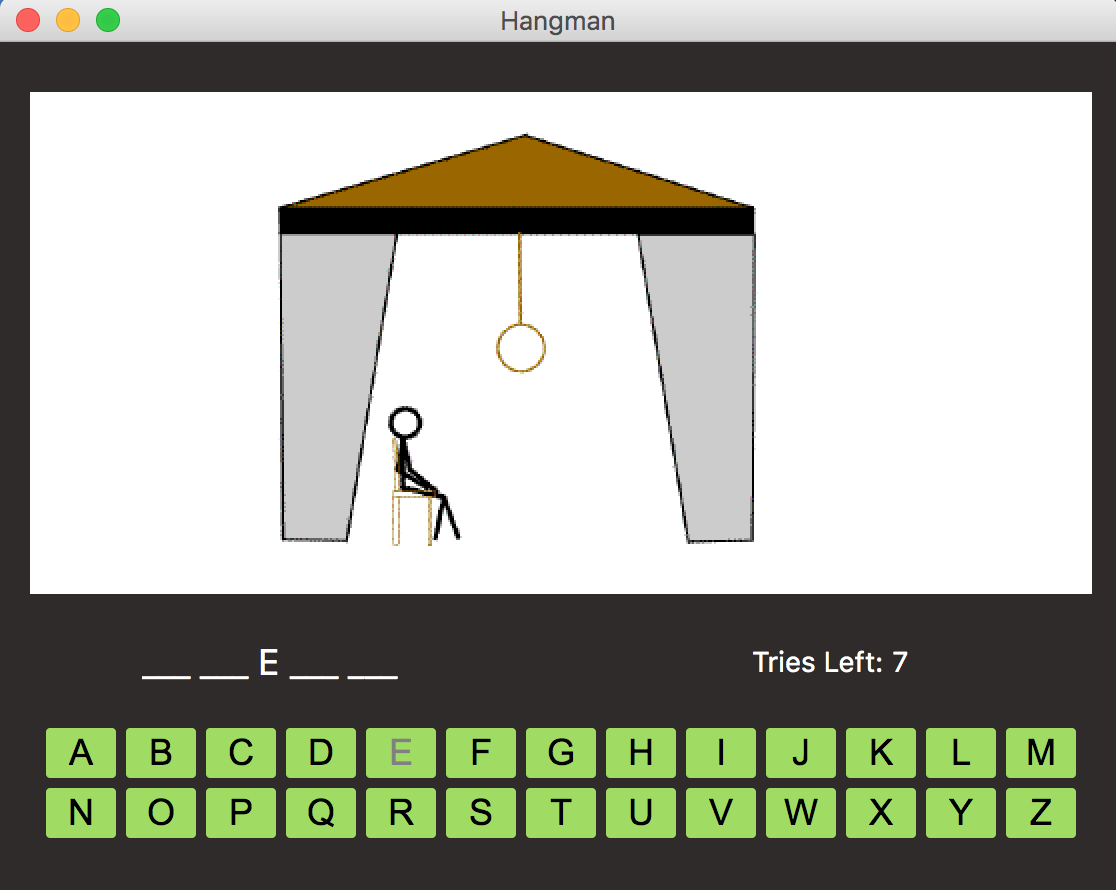
1. The user runs the game launcher.

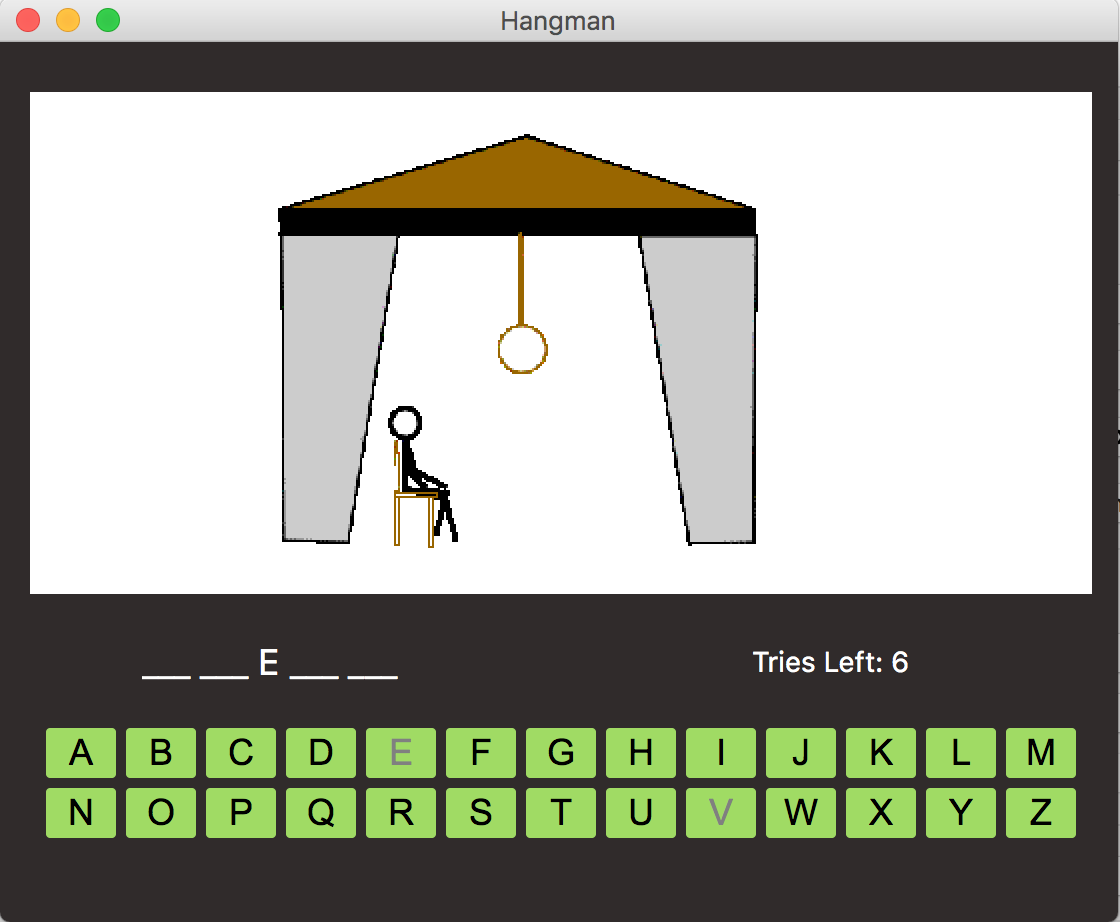
* The system displays the launcher window

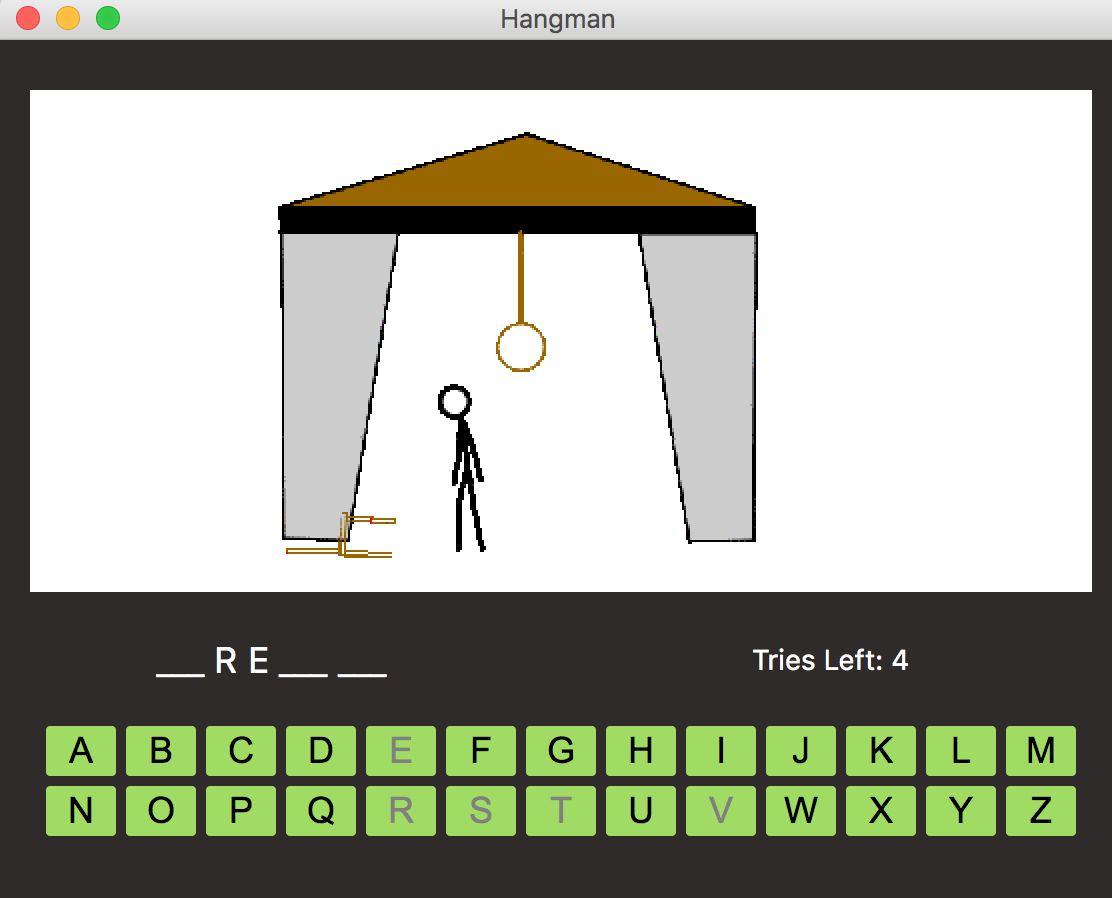
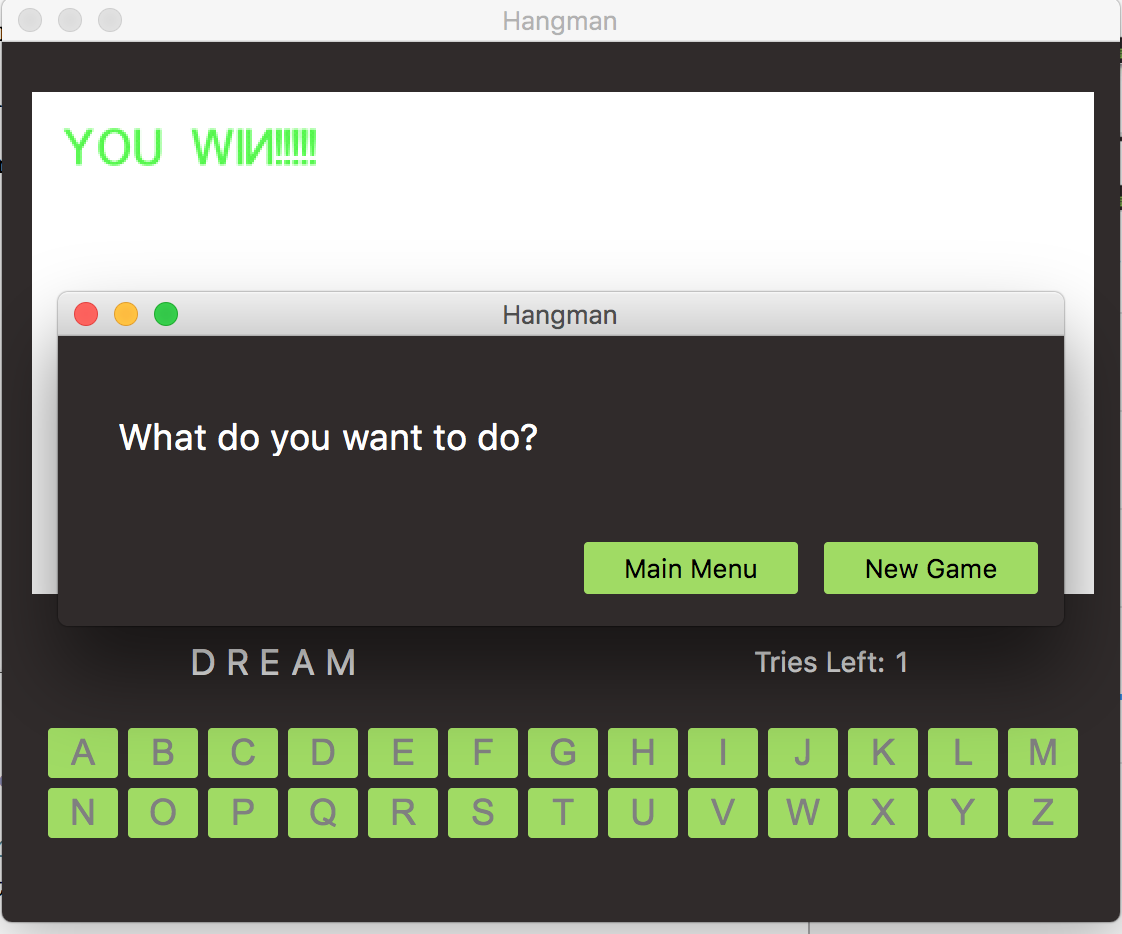
1. The user clicks on the “Hangman” button.

* The system closes the launcher window
* The system opens the Hangman game window

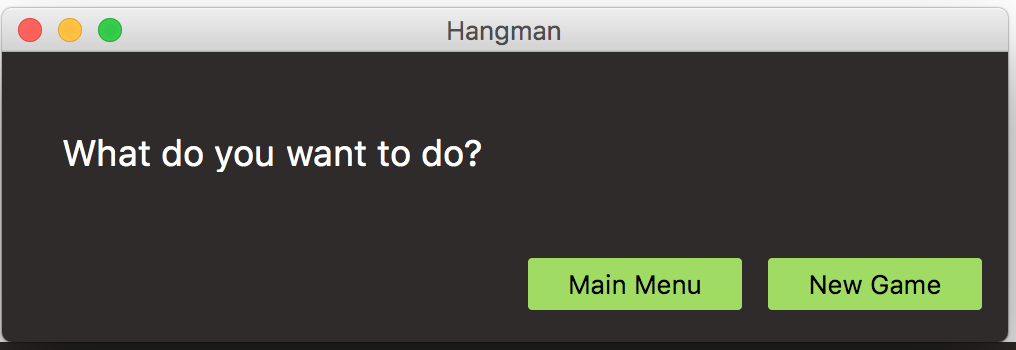
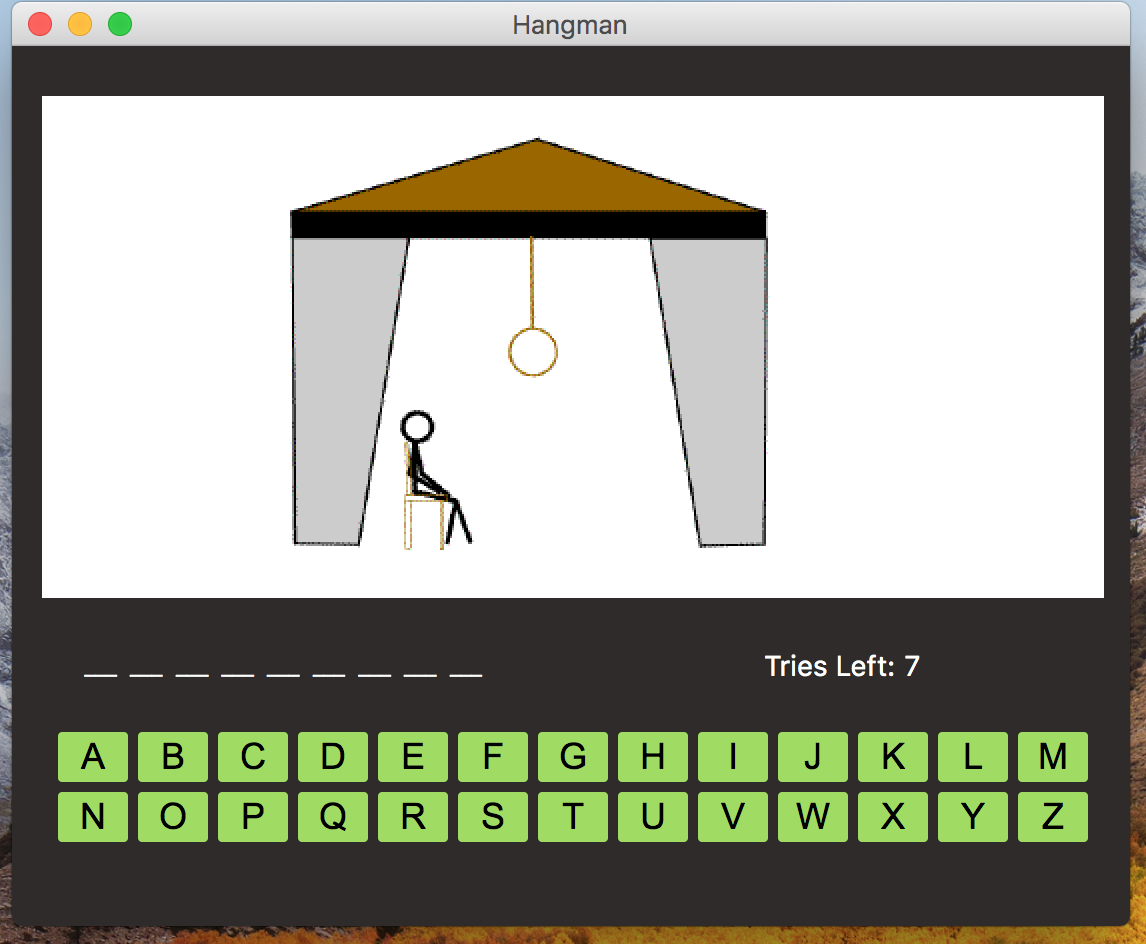
# Scenario | The user Wins the Game

1. The user selects a button from the interface.
   * The system disables the button
   * The letter from the button was found in the word, the system does not update the tries label
   * The system does not update the guess field
   * The system does not display an animation
2. The user selects another button from the interface

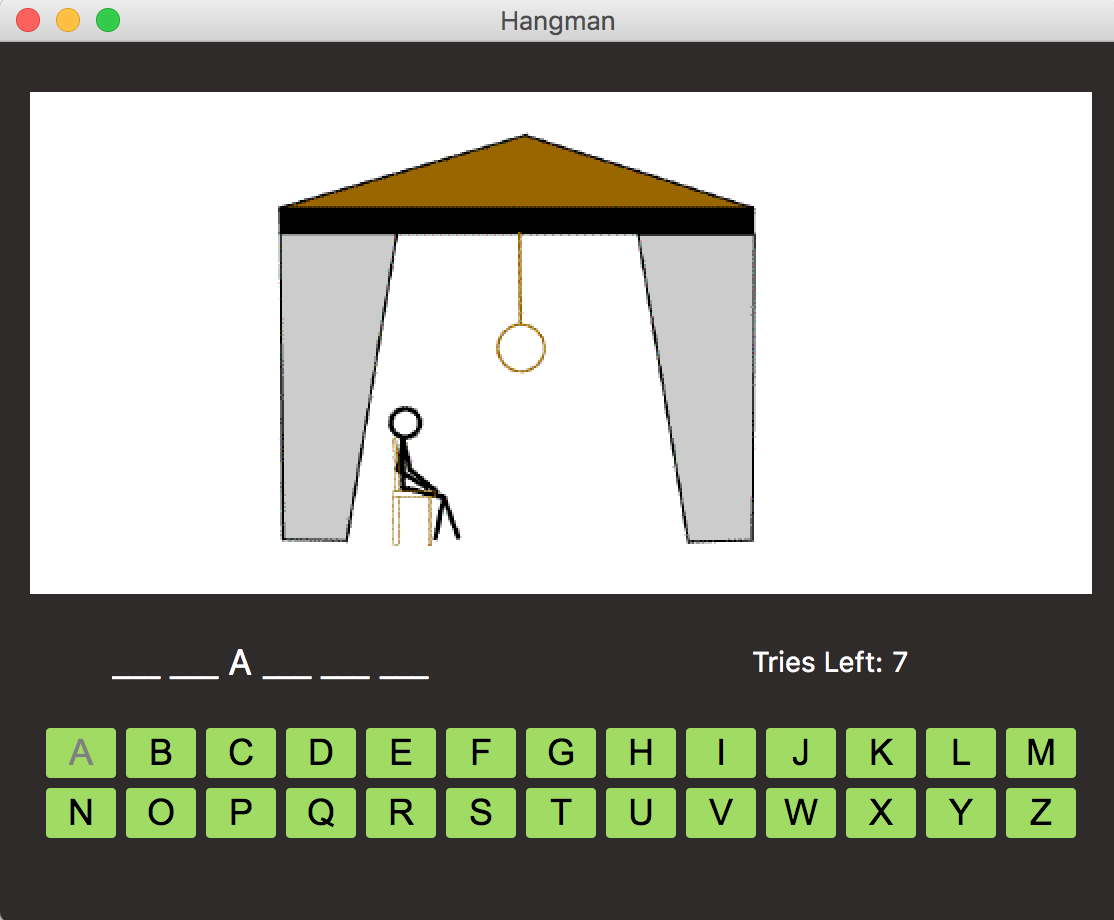
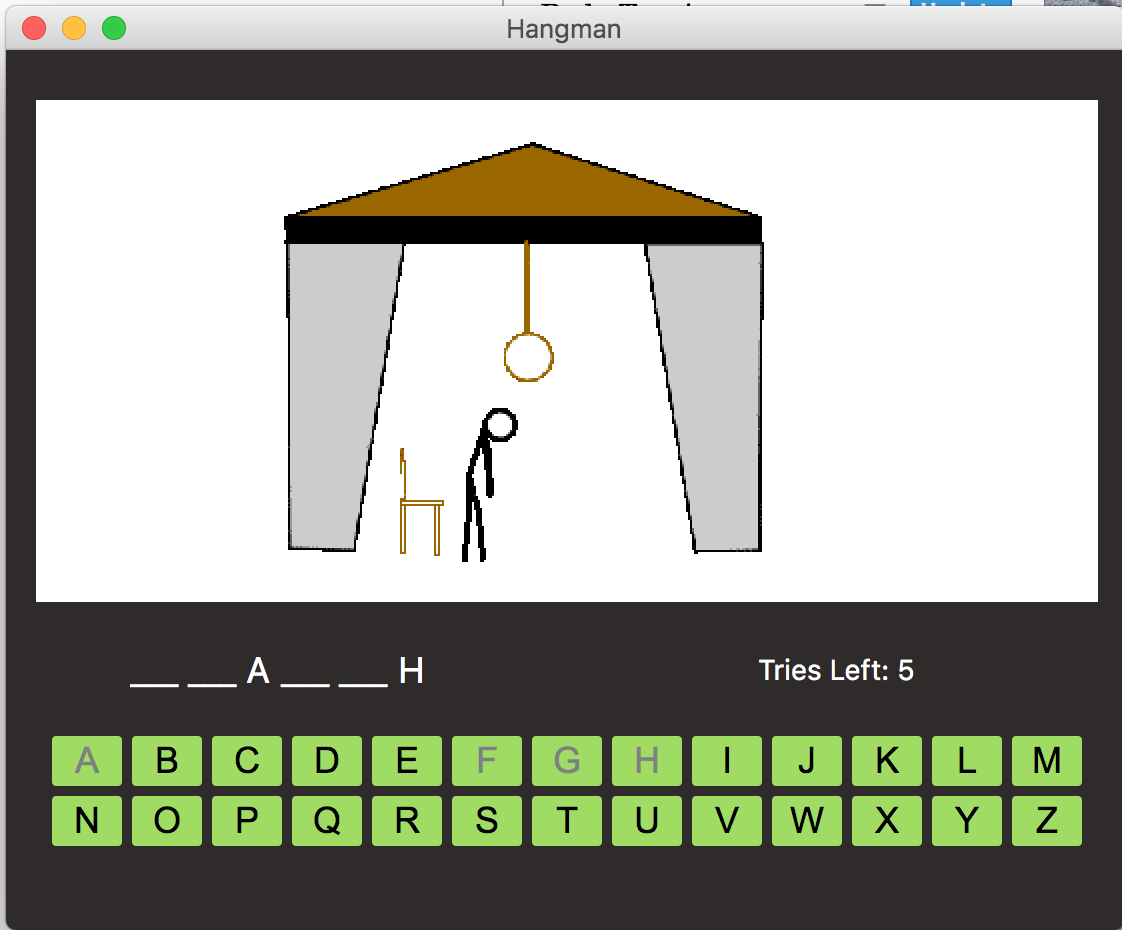
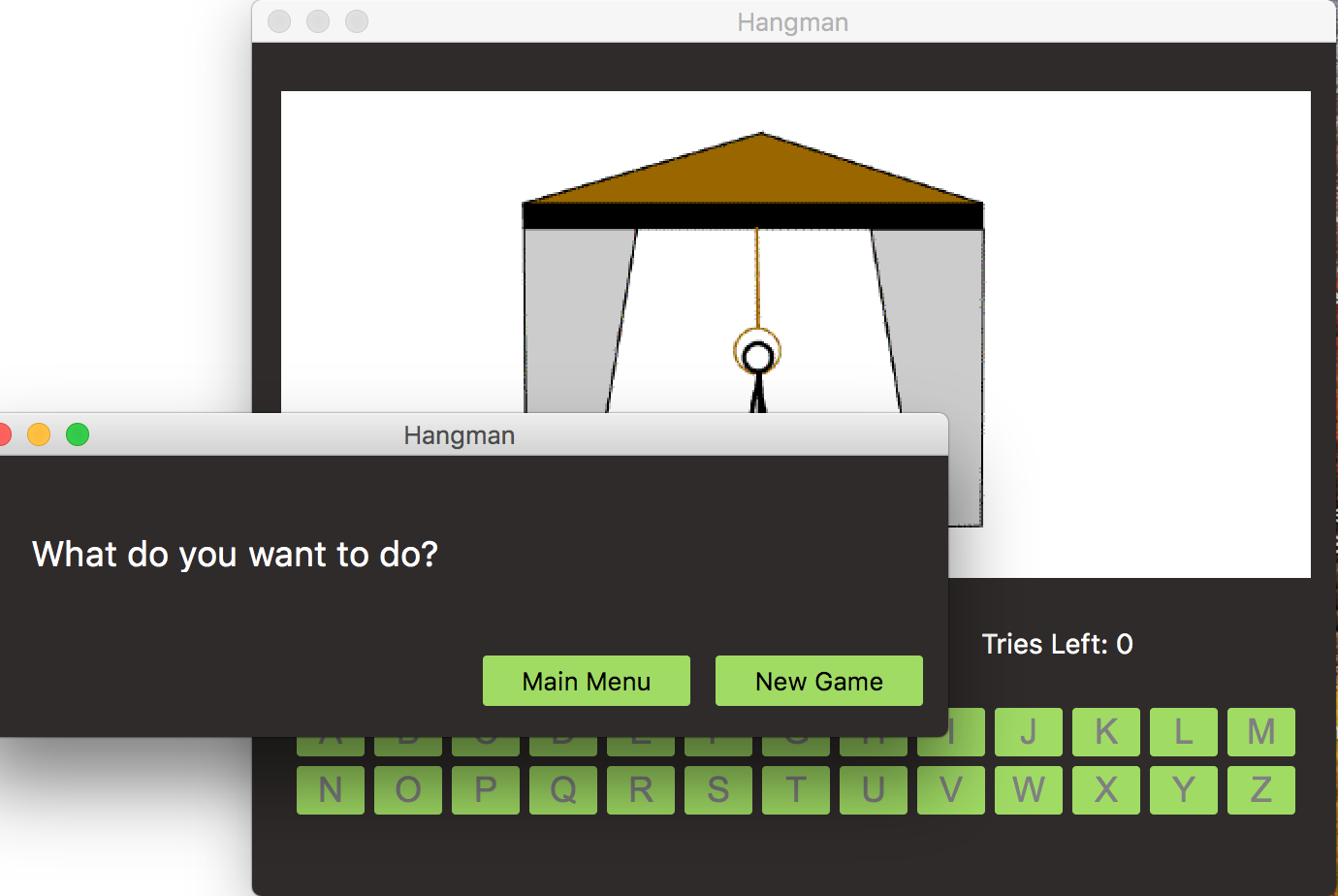
* The system disables the button
* The system updates the tries label because the user guessed wrong.
* The system displays an animation

1. The user selects another button from the interface
   * The button is disabled
   * The letter from the button was found, the system does not update the tries label
   * The system does not update the guess field
   * The system does not display an animation
2. The user has guessed all letters
   * The system disables all buttons on the interface
   * The system updates the animation label showing “You Win”
   * The system displays a confirmation box

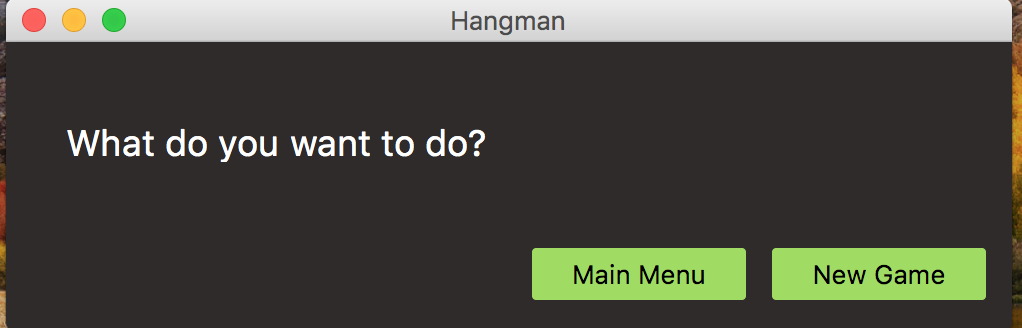
# Scenario | The user selects New Game from the confirmation box

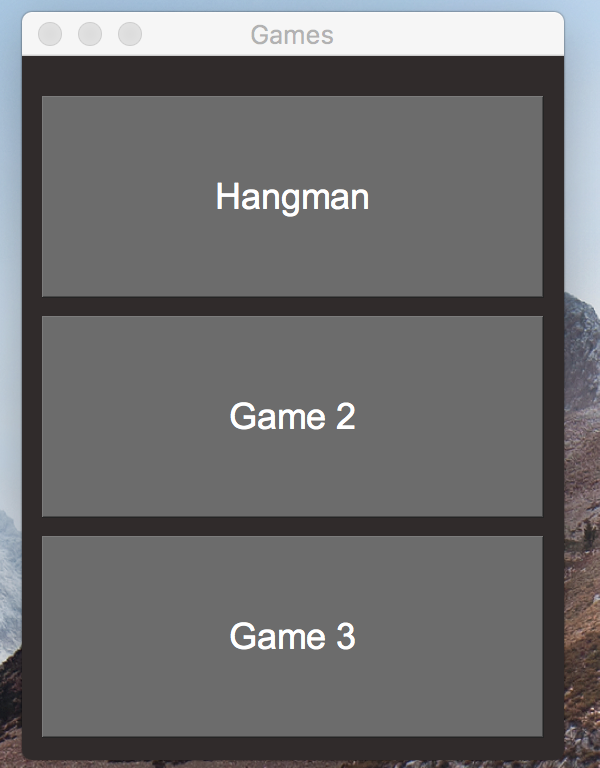
1. The user clicks the “New Game” button.
   * The system closes the current Hangman window
   * The system closes the current confirmation box
   * The system opens a new Hangman Window

# Scenario | The User loses the Game

1. The user selects a button from the interface.
   * The system disables the button
   * The system found the letter in the word, the tries label is not updated
   * The guess field is not updated
   * An animation is not displayed
2. The user selects a button from the interface
   * The system disables the button
   * The system did not find the letter in the word
   * The tries label is updated
   * An animation is displayed
3. The user selects a button from the interface
   * The system disables the button
   * The system did not find the letter in the word
   * The system reaches a tries count of 0
   * The tries label is updated
   * All buttons are disabled
   * The last animation is displayed
   * The system opens a confirmation box

# Scenario | The user selects “Main Menu” from the confirmation box

1. The user clicks the “Main Menu” button on the confirmation box
   * The system closes the current Hangman window
   * The system closes the current confirmation box
   * The system opens up the launcher window



## Source Code Structure

|  |  |
| --- | --- |
| **Code Directory** | |
| **Directory** | **Usage** |
| hangman | This is the main directory that contains all other directories including:   * bin * uiFiles * wordFiles * gifs |
| /bin | This contains all the executable code for the system. It includes:   * **createDb.py** – This is the separate program that was used to create the SQLite database from the text file. * **db.py – this is the module used by the system to retrieve a word from the SQLite database** * **dbBackup.py –** this is the module that will only be called up should the database fail to load. This module will then retrieve the word from the text file located in hangman/wordFiles/wordListBackup.txt * **launcher.py** – this is the file containing the class for the main window where the user selects which game to play. * **mainwindow.py** – The hangman’s game window class * **confirmbox.py –** Class for the confirmation GUI where the user selects “Main Window” or “New Game” * **restartGame.py –** This module handles the system exit or restart. It gets called after the user selects an option from the *confirmbox.py* class file |
| /gifs | This directory contains the animated gifs the system uses on the QLabel window within the *mainwindow.py* class file. It contains:   * **action1.gif**- animation for first missed try. * **action2.gif**- animation for second missed try. * **action3.gif**- animation for third missed try. * **action4.gif**- animation for the fourth missed try. * **action5.gif**- animation for the fifth missed try. * **action6.gif**- animation for the sixth missed try. * **action7.gif**- animation for the seventh missed try. * **winner.gif**- animation for the winning the game session. |
| /uiFiles | This directory contains the .ui files generated by QT for creating the GUI. They are not necessary for this system to function, but are included as a backup as they contain the layout of the GUI. |
| /wordFiles | This directory contains both the database and text files used to retrieve the word for the game. These are:   * **wordlist.db** – This is the database itself. This should not be deleted. If deleted, the game will proceed to use the .txt file to retrieve the word. * **wordListBackup.txt** – This contains all the words used to create the database. It also serves as the backup should the database fail. |

# Code Architecture

This system was built using a combination of modules/ libraries, and custom scripts. Some of the modules like db and dbBackup were specifically made for this program. The GUI scripts were also customized to fit the project requirements. Everything in this program was built to stand alone, which means that each file can be run independently. Custom scripts can also be used in other projects with very little modification. I wanted a system where the programmer was able to work on one file without worrying about creating issues somewhere else. This allows for independently testing each module and eases the debugging process.

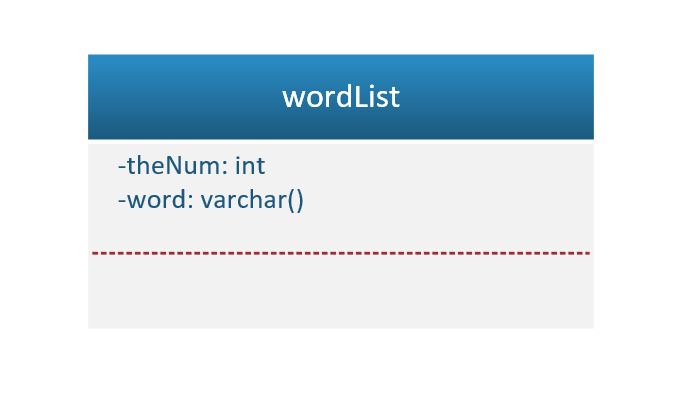
## **Database or Data Store**

This program will use the SQLite database to store a list of words from which the system can randomly pull. With SQLite, the user is not required to install additional software. This database comes embedded with the system. This database also eliminates the need for having a large array of values within the source code. By using a database, the program does not have to allocate a large amount of memory with each game session.

* Database name/file – **wordlist.db**
* Entity – **wordlist**

Attributes

* + theNum – holds an **in**t value to be used as a reference point when selecting the word.
  + word – holds the words connected to int values
* Design



Programming Language | Python 3.7

This program was written using Python version 3.7.0. The game imports the functionality of built in python libraries such as sqlite3, sys, random, and os. PyQt5 exists as an external library that has to be downloaded from the vendor in order to use.

Project Classes

Classes within the project are used to abstract re-usable pieces of code. Classes are also used to group related values, known as properties. The project utilizes these classes:

### **Ui\_launcherWin** | Location: launcher.py

This class represents the window object for the game’s launcher. It inherits attributes from the PyQt5 module. This class is used to build the object containing the buttons: Hangman, Game 2, and Exit. Class functions include:

* **openHangman()** – launches an instance of the *Ui\_MainWindow* class located in mainwindow.py.
* **setupUi()-** creates GUI objects and assigns initial attributes.
* **retranslateUi()-** translates created objects as part of the *QCoreApplication* in PyQt5
* **quit()** – calls the *sys.exit()* function from the sys module to terminate the current process.

### **objRef** | Location: launcher.py

This class holds the address location of the **hangmanOBJ** object used to open the game window, which calls the *Ui\_MainWindow* class located in the **mainwindow.py** file. Acts as a reference so that it can be accessed outside of the *Ui\_launcherWin* class. Functions include:

* **setObjectRef()-** takes in the object passed within the **openHangman()** function and stores it in memory.
* **getObjectRef()-** returns the object”s memory location.

### **ButtonList** | Location: mainwindow.py

This class is used to hold the memory locations of every button within the game (from A to Z). The buttons need to be referenced in order to change their state (enabled or disabled). Functions include:

* **addButton()-** takes in the button object as a parameter to store its memory location.
* **disableAllButtons()-** loops through the list containing all buttons to disable them.

### **GameSession** | Location: mainwindow.py

This class is used to contain all the variables necessary for calculations throughout the game session. Variables within this class include:

* **self.word** – stores the word randomly selected from the database. Can be accessed with *getWord().*
* **self.tries** – stores the number of tries a user is allowed( value is set to 7). If this is changed, then a new animation will have to be added to the **self.gifList** variable. This is the variable that gets changed every time a user fails to guess correctly. This is updated by the *setTries(arg)* function and its value can be accessed by the *getTries()* function.
* **self.totalGameTries** – stores the total amount of tries (7). This value does not change throughout the session. Use *getTotalGameTries()* to retrieve this value.
* **self.guessField –** Holds a string of the “ \_\_\_ “ symbols multiplied by the length of the chosen word. This variable is used in the guess field of the GUI. Can be modified with *setGuessField()* and accessed with *getGuessField().*
* **self.guessList –** holds a list of the guesses the user has taken. To get this value, use ***getGuessFieldList()****.*
* **self.guessField2** – empty guess field used to hold the updated value after adding letters
* **sefl.gifList** – holds the directory locations of all the animations used in this game. Use *getGIF(index)* to access a specific gif.
* **self.rightGuessess** – holds a count of the right guesses. Used to calculate if the user wins. Can be modified with *setRightGuessess(arg2)* and can be accessed with *getRightGuessess()*.

### **Movie** | Location: mainwindow.py

This class is used to hold the movie object that is created within the *retranslateUi()* function of the **Ui\_MainWindow** class. Functions include:

* **setMovieLabel(movOBJ**) – saves the memory location for the movie object.
* **getMovieLabel()** – returns memory location of the movie object.

### **Ui\_MainWindow** | Location: mainwindow.py

This is the main class used to build and display the GUI using inherited attributes from PyQt5. Functions include:

* **playAgain()-** opens up the confirmation box by calling the **Ui\_Form** class located in *confirmbox.py*
* **setupUi()** – this function creates all the buttons and widgets
* **retranslateUi()**- retranslates all buttons and widgets in the **QCoreApplication** form.
* **playanimation(num)** – this function calculates the index of of the **self.gifList** variable in the **GameSession** class and sets the label from the movie object (**Movie** class) to play the animation.
* **play (outcome)** – this function is only called when the user wins the game. It updates the movie label to show the *winner.gif* animation.
* **testLetter(theButton,triesLabel,theLetter)** – this function is connected to every button click during the game session. When a button is clicked, this function takes in the button’s letter (theLetter), its memory location (theButton), and the memory location of the tries label (triesLabel). It determines whether the letter is in the word or not, then updates the tries label, updates the guess field, calls the playAnimation () function, and determines if the game has been won or lost.

### **Ui\_Form** | Location: confirmbox.py

This class is used to create and display the confirmation box when the user wins or loses the game. Functions include:

* **setupUi** – creates the buttons and widgets.
* **retranslateUi** – retranslates into QCoreApplication.
* **buttonaction(button, confirmation)** – connected to the two buttons. It takes in the buttons memory location (button) and the button’s assigned value (confirmation).

Project Modules

The following are python modules that were imported for this project:

* **sqlite3 –** database creation and data retrieval.
* **sys –** session termination.
* **os –** process handling**.**
* **platform – determine user platform(‘Darwin’,’Linux’,’Windows’)**

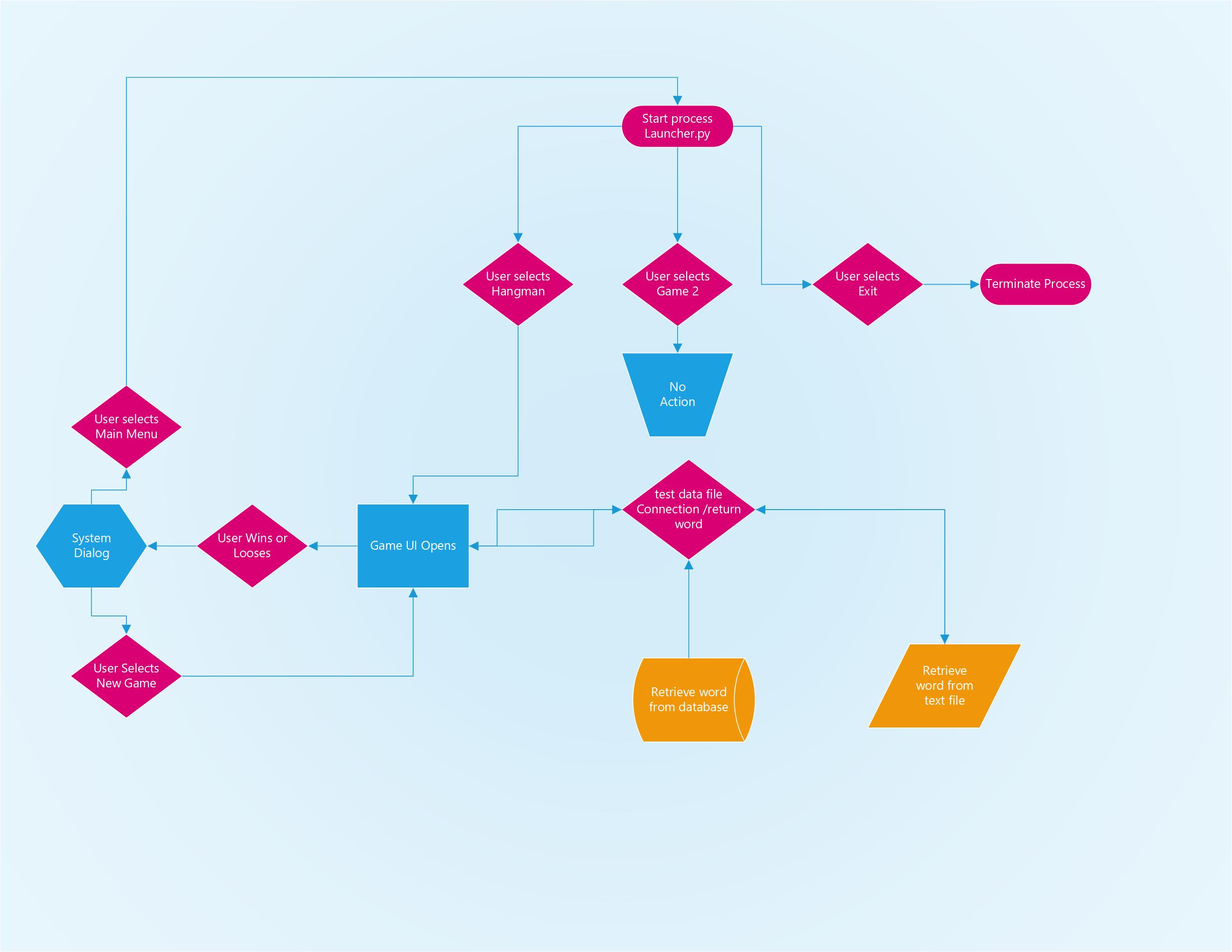
GUI Module:

* **PyQt5**

Custom Modules:

* **dbBackup** – holds the code to retrieve the word from the backup text file.
* **db** – holds the code to retrieve the word from the database.
* **restartGame** – terminates the current process, and replaces it with a new one.
* **confirmbox** – contains the Ui\_Form class.
* **mainwindow**- contains Ui\_MainWindow, GameSession, Movie, ButtonList classes.

Program Start and End Flow

This system diagram illustrates the states the process undergoes while the system interacts with the user. It covers data retrieval, the decisions the system makes based on user input, and how the system handles the process upon execution or termination.

Summary

This document’s purpose is to familiarize the end user or developer with the system and its functionality. If you would like to learn more about the program and its goal, please refer to the executive summary on page 3. This documentation introduces the various use cases between the user and the system (pages 4-10). This will familiarize you with the interaction between the system and user. The program has been divided into a comprehensive structure to enable developers to navigate the system and its modules with ease. For more information about this program’s structure, refer to pages 11-12. The chart in this section will explain the modules and files found inside the system’s directory. For developers or anyone looking to modify the program, refer to pages 13-15, which describes all the classes used in the system. This section will introduce and cover each class in further detail. It also discusses all the functions within each class. The project modules section on pages 15-16 will cover all the internal and external modules imported into the source code. There are several built in libraries in use as well as external and custom imports. This documentation also includes a system flow design to illustrate the many states of the process. To see the diagram, refer to page 16-17 of this document.

# 

# APPENDIX B (BUILD AND RELEASE PROCESS)

Current Build: 1 (stable) - Application is able to run, but currently does not have an exe file.

Bug: final dialog opens immediately after the user wins/ looses the game, blocking the final animation. Fix: Time delay to be implemented to solve this.

# APPENDIX C (CLIENT INSTALLATION INSTRUCTIONS)

**Requirements**

1. Python 3.7
2. PyQt5

**(Follow the instructions on page 19 (Appendix D) before proceeding to the next steps.**

You will need git installed on your system.

MAC

With Homebrew install git via terminal by typing: $ brew install git

Linux(Debian)

On the console, type: $ sudo apt install git-all

Windows

Visit: <https://git-scm.com/download/win> for the installer and choose 32-bit or 64-bit setup.

**To Install Hangman**

1. Create a directory in your computer where you would like to install the program
2. Travel to the directory created
3. Open a terminal session
4. Clone the gitHub repository directory by typing: git clone <https://github.com/shernand1273/hangman>
5. Navigate to the directory you just created and type: $ git status
6. You should see a message indicating you are in the branch -> master.

**Launching the game**

1. travel to the hangman directory residing on your computer via command / console / terminal
2. cd into the /bin directory
3. look for the launcher.py file
4. type:

-python launcher.py (windows)

-python3 launcher.py(macOS / Debian)

# APPENDIX D (DEVELOPER SETUP INSTRUCTIONS)

**Install PyQt5**

The easies way to install PyQt5 is to visit: **<https://www.riverbankcomputing.com/software/pyqt/download5> ,** then download the .tar.gz (macOs /Linux) or .zip (windows) file.

**Install PyQt5 via PIP**

* + Open a (command/ terminal / console) session and type: $ pip3 install pyqt5
  + Verify the installation by typing: pip3 list
  + You should see PyQt5 listed.

\*\*\*NOTE\*\*\* Python 3.7 comes with PIP3 included. If PIP3 is not working for any reason. Follow the following for your system:

**MAC**

* + - On Terminal, type:
      * + $ curl -0 <https://bootstrap.pypa.io/ez_setup.py>
        + python3 ez\_setup.py
        + curl -0 <https://bootstrap.pypa.io/get-pip.py>
        + python3 get-pip.py
        + type: $ “pip - -version” to verify installation

**Linux(Debian)**

* + - Open a console session and type: $ sudo apt-get install python3-pip3
    - To verify installation, type: $pip3 - -version

**Windows**

* + - Installation instructions can be found on: <https://dev.to/el_joft/installing-pip-on-windows>