Team-19 Project: Fast Game Library
Technical Overview & User Manual

All blue text is interactive

August 30, 2022

Contents

All blue text is interactive. The table of contents jumps to corresponding sections. The cover page contains a link to our GitHub repo.

1 Introduction

This is a performance optimized game library database. The game library consists of 1000 lines of code in 4 separate files; a main file, **GameLib.py**, a 700 line custom hash table based module **Library.py**, a unittest file, **test_lib.py**, and a package manager (also hosts custom exceptions which inherit from the built-in Exception base-class), **utils.py**. Initially, we planned on using a BST or a standalone linked list to store the Game instances, but ended up going with a hash table (we decided to make our own hash table class instead of using a dictionary, Python's built-in hash table) as it was the fastest data structure we could implement.

To achieve this, we began development on a standalone module called Library. First, we created a class called **Game**, where each instance represents a stored game entry. Next, we implemented a hash table class (made up of 3 classes; Node, LinkedList, & HashTable), & finally the Library class, which contains all methods to interact with the game library module using a hash table to store the data. To ensure we developed a module that wouldn't cause issues once we began implementing the main file, we wrote up a large unittest file, which tested each function for errors within the data structure classes. Once each function was implemented & tested, we wrote up a simple main file, GameLib.py, which imports Library as a module, instantiates a Library instance, handles external exceptions, handles startup/shutdown, user input, & calls the corresponding function based on the user's input from the main menu.

2 Data Structures

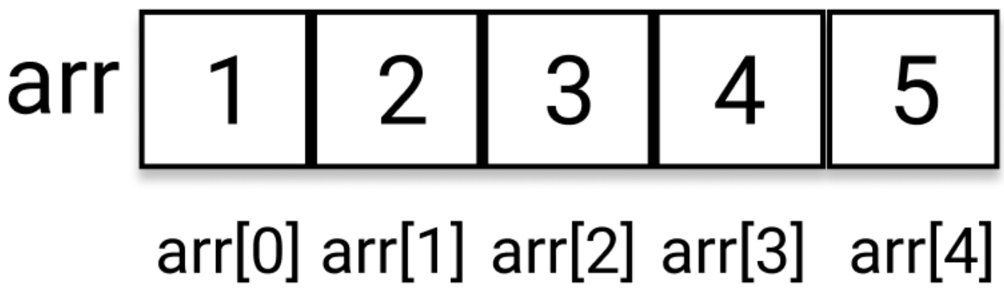
For this overview, you will need basic knowledge of Big-O Notation. As a broad overview, **n** refers to the number of elements in a data set in which operation(s) are being performed. **Big-O** notation describes the time taken for a function to perform operations on **n** elements in the **worst case**, **Big-Θ** describes the **average case**, & **Big-Ω** describes the **best case**. For the most part, you need to understand that **O(1)** (Constant) time complexity is **independent of n**, meaning it always takes the same amount of time even as $n \to \infty$. **O(n)** (linear) time complexity is **directly proportional to n**, meaning as $n \to \infty$, so does the time it takes to complete the operation(s). The most important fact to keep in mind is that $O(1)$ is the fastest possible time complexity, even in the worst case scenario; because of that, data structures/algorithms which boast a constant time complexity are the most desirable.

2.1 List

The most natural choice for storing a series of data or objects is a list. Though lists are familiar, have ample built-in functionality, & are speedy when accessing an element given an index of the element, lists perform all other operations in linear time. Though that's not the worst time complexity around, we would prefer to choose a data structure which will perform as many operations as possible in $O(1)$, especially when thinking about databases.

- **Access:** $O(1)$
- **Insertion:** $O(n)$
- **Deletion:** $O(n)$
- **Searching:** $O(n)$

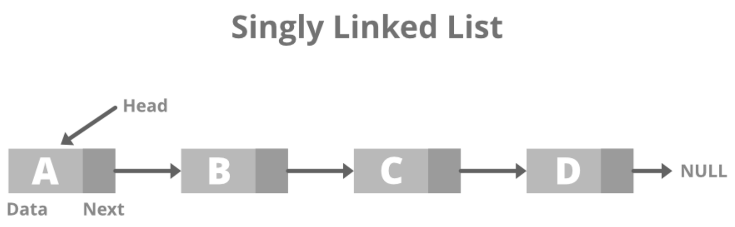
Due to the unfavorable time complexity of most of its main operations, lists are a poor candidate as our main storage data structure for the game library. Therefore, we researched alternatives to lists. Though we did not use a list by itself, a list will be used later on to implement a portion of our final data structure within the Library module.



2.2 Linked List

Linked lists are exactly how they sound; they are containers, AKA **Nodes**, which contain data in a **data** attribute, & are linked together by a **next** attribute, which points to the next Node in the list (**None** if there is no next Node). In memory however, unlike lists, they are not a contiguous block of memory, & are not indexable. Before we continue, take a look at a physical linked list, as well as a bare-bones implementation of a linked list class in Python:

```python
class Node:
    def __init__(self, data=None, next=None):
        self.data = data
        self.next = next

class LinkedList:
    def __init__(self):
        self.head = None
```



The downside to using linked lists over lists is that accessing becomes linear since we only initially have access to the first Node (Head), meaning we must begin any new iterations at the beginning, calling next for each node in the linked list individually in the worst case. The trade-off from built-in lists is that inserting/deleting is now done in constant time. That is, to add/delete an element, all we have to do is reroute where the next attributes of each adjacent node to the correct nodes rather than reforming an entirely new list.

- **Access:** $O(n)$
- **Insertion:** $O(1)$
- **Deletion:** $O(1)$
- **Searching:** $O(n)$

Though this is an improvement, the user will be accessing/searching far more than inserting/deleting, so we needed an even better solution, hopefully with more constant operations.

2.3 Hash Table

Our goal all along has been to find a data structure which can insert, delete, access, & search in constant time complexity. If we could somehow combine the constant time access of a list with the constant time delete/insert of linked lists, then we could achieve this. A hash table is a data structure which stores key:value pairs. In Python, a built-in hash table called a **dictionary** is provided, though we decided to implement our own for reasons discussed later. In our case, it is a list of linked lists; the key (str) is put through a hash function which returns an int. That hashed integer is then used as the index of the bucket to store the value in that corresponds to that key.

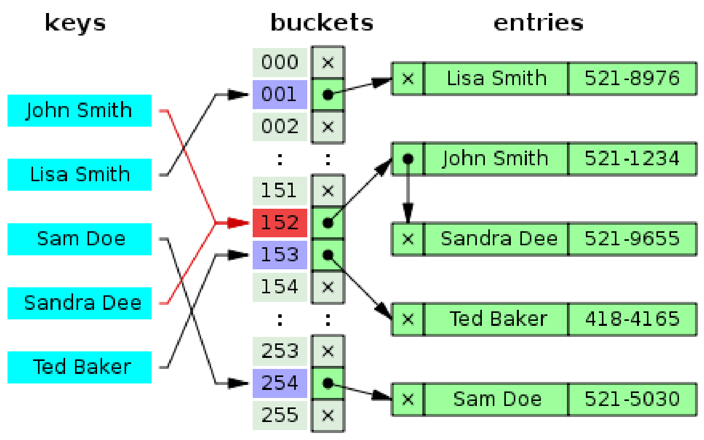
What happens when two keys share the same hash value? That's called a **collision**. Aside from writing a better hash function, we can implement a method to handle collisions called chaining, which is where we **link** keys that had a collision together in a **linked-list** structure stored at the index of collision. By that, we have **list of linked lists**, which is indexable by the hashed value of the key (str).

One of the main reasons we decided to make our own hash table instead of using Python's built-in version is because dictionaries handle collisions using a technique called open addressing. When handling collisions using open addressing, as the **load factor** (λ) of the table increases, for any $\lambda < 0.8$, the amount of time taken to retrieve an element given a key increases linearly. The issue arises is when $\lambda \geq 0.8$, as the time it takes to retrieve an item from the hash table given a key increases exponentially. Though chaining has its own downsides such as cache inefficiency, it experiences a linear time increase for all values of λ.

Technically, a hash table performs all operations in $O(n)$ time complexity because the **worst case** involves having to iterate over the linked list at that index that is a length n (every key:value pair was hashed to that exact index). Given a large enough hash table though, & a well optimized hash function, the statistical probability of a collision occurring is negligible. Even with the infinitesimally small chance that a single collision occurs, it is still not proper to refer to its operations in Big-O notation as constant. Instead, we will describe its time complexity in Big-Θ Notation, which describes the **average case**.

- **Access:** $\Theta(1)$ $\{m \in \mathbb{Z} \mid m \in [1, \infty]\} \equiv$ **Buckets in table**
- **Insertion:** $\Theta(1)$ $\{n \in \mathbb{Z} \mid n \in [1, \infty]\} \equiv$ **Elements in table**
- **Deletion:** $\Theta(1)$ $\{\lambda(n, m) \in \mathbb{R} \mid \lambda \in [0, 1],\ \lambda = \frac{n}{m}\} \equiv$ **Load factor**
- **Searching:** $\Theta(1)$

Below, we have included a physical & 'bare-bones' Python representation of a hash table.



```
class HashTable:
    def __init__(self, size=50):
        self.SIZE = size
        self.arr = [LinkedList() for _ in range(self.SIZE)]

    def hash(self, title_):
        hsh = 0
        for c in title_:
            hsh += ord(c)
        return hsh %self.SIZE
```

2.3.1 Hash Function

A desirable hash function is **computationally efficient**, **evenly distributes keys** (minimizes collisions), & of course, it should **return the same int**, I, **given the same str**, S, x times, where $x \in [0, \infty]$. This is integral to a proper hash table, as we will need to later retrieve the same values using the corresponding key we used to store it. As shown in the code above, the hash function we used isn't very complex. In fact, it is the simplest hash function we know of, though given the program has no intention of being used publicly on a large scale & is being developed on a **strict** time-frame, it's adequate as is. If you're not familiar, $ord(c)$ converts any ASCII character into it's corresponding integer ASCII value. With that in mind, here is the formula for our hash function, where **x** is a **key** (str), & **m** is the **number of buckets in the table** (int):

$$hash(x) = \left[\sum_{n=0}^{len(x)} ord(x[n]) \right] \quad \text{mod } \mathbf{m}$$

3 Classes within Library Module

The Library module is around 700 lines of code, & hosts 5 total custom-developed, speed-optimized classes

3.1 Game

Each Game instance represents a single game entry in Library.

3.1.1 Attributes

- **title (str, optional):** Game Title. Defaults to "N/A"
- **rating (str, optional):** Game Rating. Defaults to "N/A"
- **size (str, optional):** Game size (GB). Defaults to "N/A"
- **price (str, optional):** Game Price (USD). Defaults to "N/A"

3.1.2 Methods

- **__str__(self):** Formats Game instance to str; Game title in brackets
- **__repr__(self):** Formats Game instance's 4 attributes to CSV style str
- **@classmethod stog(cls, line):** Given a list of 4 str, returns Game instance

3.1.3 Supported Syntax

```
#constructs empty Game
gme1 = Game()
# Constructs Game with str attributes
gme2 = Game("ExTitle", "4.5", "50GB", "$20")
# Constructs Game given a list of 4 str attributes
gme3 = Game.stog(["exTitle2", "4.9", "10GB", "$30"])
# Converts Game instance to str, prints title in brackets
print(gme3) # Out: "[exTitle2]"
# Converts Game instance to str, prints all attributes in csv form
print(repr(gme3)) # Out: "exTitle2,4.9,10GB,$30"
```

3.2 Node

The Node class has no standalone purpose; it is only to be used in conjunction with the LinkedList class.

3.2.1 Attributes

- **data (Game, optional):** Game Instance. Defaults to None
- **next (Node, optional):** Next Node in LinkedList instance. Defaults to None

3.2.2 Methods

- **__str__(self):** Formats a Node's contents (Game) to a string
- **@property title(self):** Gets title of Game stored within Node's data attribute

3.2.3 Supported Syntax

```
# Constructs empty Node
n1 = Node()
# Constructs Node with gme3 (from Game section); n1 is next Node
n2 = Node(gme3, n1)
# Returns the title of game in Node
print(n2.title) # Out: "exTitle2"
# Calls Game's dunder str method
print(n2) # Out: "[exTitle2]"
```

3.3 LinkedList

Used in Hash Table to implement chaining method for handling collisions.

3.3.1 Attributes

- **head (Node):** Contains first Node in Linked List. Always initialized to None.

3.3.2 Methods

- **__iter__(self):** Iterator; allows use in for loops
- **__contains__(self,title):** allows use of "in" operator given title

- **__len__(self):** len(LL) returns number of Game instances in LinkedList
- **__str__(self):** Formats linked list into a string
- **__delitem__(self,title):** Deletes an item from LinkedList given title
- **emplace_back(self,game):** Constructs & appends Node with Game to itself

3.3.3 Supported Syntax

```
# Constructs empty linked list
ll = LinkedList()
# Constructs & appends Node with Games (from Game section)
ll.emplace_back(gme2)
ll.emplace_back(gme3)
# Returns number of Games in LinkedList
print(len(ll)) # Out: "2"
# Returns bool; True if Game is in ll
print(gme2.title in ll) # Out: "True"
print(gme1.title in ll) # Out: "False"
# Iterates through each Node in ll
for node in ll:
    print(node,sep=",") # Out: "[ExTitle], [exTitle2]"
# Deletes Game with matching title from ll
del ll[gme2.title]
print(len(ll)) # Out: "1"
# Formats ll as str; physical representation of linked list
print(ll) # Out: "[exTitle2]->[None]"
```

3.4 HashTable

Data structure containing all Games in Library; syntactically identical to dictionaries

3.4.1 Attributes

- **SIZE (int, optional):** Number of linked Lists within list. Defaults to 50
- **arr (list[LinkedList]):** list containing LinkedLists

3.4.2 Methods

- **hash(self,title):** Hash Function
- **__setitem__(self,title,game):** Inserts Game object into HashTable
- **__getitem__(self,title):** Gets Game at hashed index if it exists
- **__delitem__(self,title):** Deletes specified game in Hash Table (by title)
- **__len__(self):** len(HT) returns number of Games in Hash Table
- **__str__(self):** Formats hash table to str (shows contents & links)

3.4.3 Supported Syntax

```
# Constructs HashTable with 2 buckets
ht = HashTable(2)
# Returns hash value of passed in str; dependant number of buckets
print(ht.hash("Howdy")) # Out: 1
# Inserts Game (from Game section) in ht using title (key)
ht[gme2.title] = gme2
ht[gme3.title] = gme3
# Returns number of Games in ht
print(len(ht)) # Out: "2"
# Returns Game with corresponding title (key)
print(repr(ht[gme3.title])) # Out: "exTitle2,4.9,10GB,$30"
# Deletes Game from ht with matching title (key)
del ht[gme3]
print(len(ht)) # Out: 1
# Prints str representation of ht
print(ht) # Out: "[None]\n[ExTitle]->[None]"
```

3.5 Library

Represents the Game library; wrapper for data structures; highest abstract class

3.5.1 Attributes

- **size (int, optional):** size of the dataBase. Defaults to 50
- **dataBase (HashTable[Game]):** Hash Table which contains all stored Games
- **MEMDIR (str):** Persistent memory directory. Defaults to LibMem.csv
- **titles (list[str]):** For lexicographical sorting lambda in printLib()
- **modified (bool):** True when a game(s) has been imported, added, deleted

3.5.2 Methods

- **exportLibrary(self):** Creates a backup of Library
- **resetLib(self):** Prompts user; erases all Games permanently from memory
- **@staticmethod instructions():** Prints Instructions
- **@staticmethod promptMainMenu():** prompts & grabs user selection
- **search(self):** Searches Library by user input, prints matching Game attributes
- **importGames(self):** Imports Games from a user-specified CSV into Library
- **saveAndExit(self):** Saves & Exits (writes any changes made to memory)
- **exitNoSave(self):** Exits program without saving changes
- **loadMemory(self):** Loads in CSV at the path held by self.MEMDIR on startup
- **addGame(self):** Adds user defined Game to Library's dataBase
- **delGame(self):** Deletes a Game instance given a Title
- **printLib(self):** Prints library in a user friendly way
- **printdataBase(self):** Prints Library to terminal as a LinkedList
- **__len__(self):** len(Lib) returns number of Games in Library
- **__str__(self):** Formats Library's HashTable (dataBase) to str

4 Main Program Flow

This section contains a detailed overview of the state of the program & related files from pre-launch to shutdown.

4.1 Pre-launch

Pre-launch describes the program's prior to compilation/execution.

4.1.1 Files

All located within the same directory, Fast Game Library consists of 3 required py files, as well as 1 required CSV file; The main file, **GameLib.py**, the Library module, **Library.py**, the dependency manager as well as the host of a number of custom exception classes which inherit from the built-in Exception class, **utils.py**, & finally, the CSV file which acts as the persistent memory for the Library class, **LibMem.py**. A backup of the default games within LibMem.csv exist in a file named **backup.csv**. Additionally, a tester-file, **implib.csv** is present to test the error detection of the importLibrary method.

4.1.2 Persistent Memory

The program is able to simulate persistent memory (stores data to use on next boot) by reading/writing to **LibMem.csv**. Any data stored in this file will be used to initialize the main Library instance. As default, the memory is loaded with 35 game entries, which will populate Library's HashTable, self.dataBase, at run-time. A backup of the default games can be found in backup.csv

4.2 Run-time

4.2.1 Dependency Validation & Installation

When compiled from the main file (GameLib.py), utils.py is first executed. utils contains 3 sections; the first hosts 5 custom exception classes which all inherit from Python's base-class, Exception. The second part checks if the **Colorama** package is installed; if not, it uses a sub-process to pip install Colorama. Lastly, it checks which version of Python it has been compiled from. Because of the match-case statements used in the main & module file, the program will throw a syntax error if the version is lower than Python 3.10. To avoid confusion, an error is displayed to upgrade Python if the requirements are not met at run-time.

4.2.2 Library Initialization

Assuming the program was compiled from GameLib.py, the main if statement will be entered. First a Library instance will be created with a size of 25 (25 buckets in self.dataBase, a HashTable attribute of the Library class. It can be any integer from 1 to the maximum integer allowed in 64-bit systems), the terminal is cleared of all text, & the user is asked for their name. Once a name is entered, it will format the name such that the first letter of the name is capital, & all trailing letters are lowercase, then output a welcome statement with the name. If the entered name contains non-alphabetical characters, it will also display **You have an interesting name...**. After that, a sleep is entered for 3 seconds, & enters a while True loop, where it begins iteration at nested try blocks.

4.2.3 Selection Handling

Starting, the class method proimptMainMenu is called, which displays the text-based main menu to the terminal, & asks the user for an input within the options range (1 through 11). If in range, it returns the integer response to main, & sets **sel** equal to that validated selection. Assuming no exceptions are raised, a **match-case** statement containing 11 cases is entered, each representing an option from the main menu function within the Library class. The value of sel is then matched to a case, & the case block is entered, where it will make a corresponding call to an instance, or class method the Library class.

4.2.4 Exceptions

Within the first/outermost try block, 2 except blocks an be entered.

1. **InvalidSelection** is raised when the user entered a value for sel which is not an integer within the interval [1,11]. This will call the invalidSel function, passing in the exception unique identifying number, 0, as well as the exception's description itself. An error showing the character(s) the user input followed by a prompt telling the user to use integers within the accepted range will be shown in terminal for 6 seconds, then the main menu will reappear.

2. **Exception** is used to handle all other exceptions that may be thrown that non-intentional exceptions. When this happens, unknownException will be called within main, passing in the exception's unique identifier, 1, as well as the exception's description. A message is then displayed to terminal that an unknown exception has been raised, & its description will be displayed on screen for 6 seconds. Assuming a non-fatal exception, the main menu will reappear.

Whithin the second/innermost try block, 3 exceptions can be handled.

1. **SaveExit** is an exception raised by Library's instance method, SaveAndExit. Within that function, which is selection number 10, if the user has modified any data in the Library which differ from the persistent memory, all data in persistent memory will be overwritten by the game data found within the Library at the time of the call, then the exception is raised to enter the safeExit method in main, which will display a message that states the program is saving, then it will prompt goodbye.
2. **ExitNoSave** is an exception raised when item 11 on the main menu is selected, which simply prompts the user yes or no if they want to exit. Persistent memory is not written over in this case, though safeExit is called from main, which will display a message saying goodbye, then the program exits.
3. **Exception** is used to handle all other exceptions that may be thrown that are non-intentional. It has the unique exception identifier, 2, which will be passed into the unknownException method within main, displaying a message stating an unknown exception has been raised, as well as the exception's description.

4.2.5 Modified Data Detection

To determine if Library needs to modify data at shutdown, a Boolean attribute called **modified**, which is initialized at runtime as False, is used. Anytime a modifying function is called, modified is set to True.

4.2.6 Interrupts

When a valid selection is received, the selection will be used to call the corresponding method within Library's class. The main program is interrupted, & all program control is transferred to the Library module until the callee returns, or raises an exception, which will allow main to resume the normal loop.

4.3 Shutdown

4.3.1 Persistent Memory

When a SaveAndExit call is made by the user to main menu, if modified is set, the Library instance will first write all data to the persistent memory, then raise a SaveExit exception, which will stylistically shutdown the program, with the message displayed depending on whether or not a save has been completed.

5 User Manual

[WARNING]: Python 3.10 or above, & pip are REQUIRED to compile this program!
[WARNING]: Running this program in external terminal is REQUIRED!
[WARNING]: If LibMem.csv is empty, games will NOT appear; if that file is empty upon startup, please select 8 on the main menu after entering your name, then type in "backup.csv", then select option 10 (save & exit). Now run the program.
[WARNING]: Colorama & other formatting has been designed based on WSL2 ran on Windows Terminal; any other terminal may cause formatting issues

You can check your version of Python by executing command:

```
python --version
```

5.1 Basic Usage

5.1.1 Compilation

To compile this program, run from external terminal with Python 3.10 or above with pip installed; Ensure all necessary files are in the same directory, cd into the directory from terminal, then **run command**:

```
python GameLib.py
```

[NOTE]: If you see any [ERROR] messages, your system does not meet the aforementioned requirements from the warnings above; you will need to update/install necessary dependencies before trying again.

5.1.2 Startup

When the program starts, it will ask you for a name. Regardless of the upper/lowercase combination you input, your name will be displayed back to you in the form: Xxxxx Xxxxx ...

The only exception is if the name you input contains characters that aren't in the English alphabet; the program will return the exact string you input, & display "You have an interesting name..." underneath the welcome prompt.

Prior to the main menu being displayed, the program will be loading in a CSV file, **LibMem.csv**, in the background which contains the **persistent memory** for the program; anytime you start, or save & exit, the program will write to this file such that the next time you boot, those same changes you made will still be there. That means if nothing's in that file, no games will be in storage at boot. If you accidentally deleted all games & want them back, select **Import Library**, then input **backup.csv** as the file to import. Once the import is complete, you should have 35 games loaded in (assuming it's in the same directory, &/or that you didn't accidentally delete the backup as well.

5.1.3 Option Selection Input on Main Menu

Menu items are selected by entering input, $\{sel \in \mathbb{Z} \mid sel \in [1, 11]\}$. If your selection is not in that interval, you will see an error screen. After 4 seconds the main menu will reappear.

5.2 Main Menu Options

5.2.1 Search

This option will allow you to search for any game which is stored in the library at the time of the search & will return the attributes from the game with a matching title to the terminal. You can also input 'back' to return to the main menu. If the game isn't stored in the Library, you will receive an error stating the game was not found, then you will be redirected back to the input screen after a few seconds.

5.2.2 Add Game

This option will allow you to manually add game entries one-by-one. On the input screen, you can either enter the title of the game you want to add or enter 'back' to return to the main menu. If your input is a title, it will ask you to input the rest of the game's data. Once you enter the 4 attributes, it will attempt to add it to Library's database. If the game already exists, or you entered a game with a title of len == 0, you will receive a corresponding error, then you'll be redirected back to the input screen. If the game you input is valid & not a duplicate, the game will be added to the database, & "**Game Successfully Added to Library!**" will appear on screen. After 4 seconds, you will return to the input screen.

5.2.3 Delete Game

This option, functions inverse to Add Game; You will be prompted for a game title that you'd like to delete, or 'back' to return to the main menu. If you enter a title, & the game exists in the Library, "**Game Successfully Deleted!**" will appear on screen. If the game is not found, "**[ERROR]: Game not found**" will appear. Regardless if found or not, you will be prompted to hit enter to continue. When you do, you will return to the input screen.

5.2.4 Instructions

This option will print the instructions to the terminal. Admittedly, they're more of a description of the program rather than instructions, but no worries; instructions are always on screen at all times.

5.2.5 Print Library

This option will print the current library in lexicographical order from top to bottom in a nice, user-friendly format. It will also display how many games are in your library (be sure to scroll to the very top if you can't see the game count; the list of games is long). Once it prints, it will prompt you to press enter when you'd like to return to the main menu.

5.2.6 Print Database

This option is similar to Print Library, except instead of printing in a user-friendly format, it prints Library's database in hash table form; it'll be formatted as a vertical lists with horizontal linked lists of games. Any empty slots, or end of linked lists will show up as "**[None]**"

5.2.7 Delete Library

This option will permanently delete all data in LibMem.csv, as well as the current data in Library's database. This is not reversible, not even if you exit without saving. If you accidentally do this, you can import "backup.csv" to restore the original library, or create an exported file before deleting, the import that file. You will be asked if you'd like to delete. If you select "N" you will be taken back to main menu, otherwise, you'll see "**Library Successfully Reset!**".

5.2.8 Import Library

This option will allow you to import a csv containing game entry data into your library, or type in ’back’ to go back. When it reads the file, if the title of a game has len == 0, it will be counted as an empty entry, & will not be imported. If it already exists in the library, then it will be counted as a duplicate entry. Else, it will be added to the Library. At the end of the import, statistics on what happened during the import (Numbers of successful imports, duplicate imports, & empty imports) will appear on the screen. You will then be asked to hit enter to return to the input screen again.

5.2.9 Export Library

This option will ask you to input a filename for the file you’d like to export your current library to, or type ’back’ to go back to the main menu. If the filename already exists in the directory, it will ask if you’d like to overwrite the file. If no, you’ll be taken back to the input screen. If yes to overwrite, or you didn’t get that warning, it will write all of the current library’s game data to a csv file with the name you chose. The data will be formatted just like the csv’s library can read in, that way you can use the exported files as a backup.

5.2.10 Save & Exit Program

As the name suggests, this option will first prompt you to select yes or no to saving & exiting. If no, you’ll return to the main menu. If yes, then any changes (imports, adds, deletes) made to the library (that differ from LibMem.csv) will cause the program to re-write all of Library’s data to LibMem.csv, then exit the program.

5.2.11 Exit Without Saving

This option will prompt you yes or no to exit without saving, warning you about data loss. If no, then you’ll return to main menu. If yes, then the program will simply exit without saving any of the changes you’ve made. The next time you boot, whatever is in LibMem.csv before you booted last time will be the exact same data.

5.3 File Management

- **[REQUIRED FILES] (Place in Same Directory):**
  1. **GameLib.py: Main Program**
  2. **Library.py: Module GameLib.py Runs off of**
  3. **utils.py: Dependency Manager & Custom Exception Host**
  4. **LibMem.py: Persistent Memory**
- **[RECOMMENDED FILES] (Place in Same Directory):**
  1. **backup.csv: backup of LibMem.csv with 35 games in it (just in case you accidentally delete it)**
  2. **implib.csv: will test the error checking of the Import Library selection**

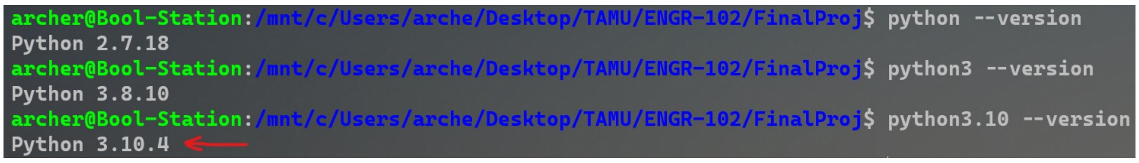
6 Visual Tour

[NOTE]: As stated in the user manual, compilation must occur from an external terminal (preferably WSL2 using Windows Terminal). Due to the difference in how each terminal interface displays colors & ASCII characters, it is not guaranteed that the program will appear similarly, nor correctly, for every possible permutation of terminal on their respective operating systems.

6.1 Compilation

6.1.1 Manual Dependency Validation

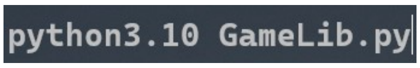
Depending on which terminal and/or operating system you are using, as well as how many previous versions of python you've installed, the prefix "PythonX," where X is a version number (if any) may be different on your terminal. In my case (WSL2 on Windows Terminal), "Python3.10" is the correct prefix for Python version 3.10.4. The flag to check version is "–version" for all versions.



```
archer@Bool-Station:/mnt/c/Users/arche/Desktop/TAMU/ENGR-102/FinalProj$ python --version
Python 2.7.18
archer@Bool-Station:/mnt/c/Users/arche/Desktop/TAMU/ENGR-102/FinalProj$ python3 --version
Python 3.8.10
archer@Bool-Station:/mnt/c/Users/arche/Desktop/TAMU/ENGR-102/FinalProj$ python3.10 --version
Python 3.10.4
```

6.1.2 Execution in Terminal

Below is the command to run the program from the main file (within the same directory as all other required/recommended files). The prefix "PythonX," where X is a version number (if any) may be different on your terminal.

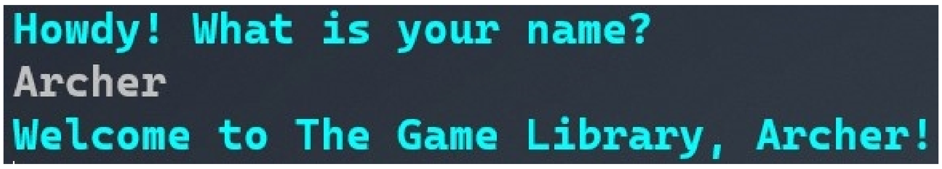


```
python3.10 GameLib.py
```

6.2 Name Input

6.2.1 Name Input & Displayed Formatting

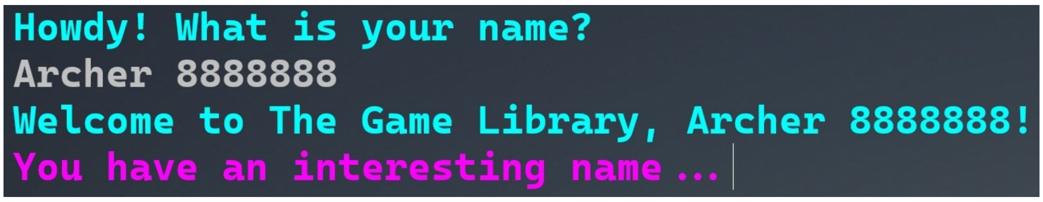
Below is the first thing displayed on the user's terminal at runtime assuming no dependency errors, and all alphabetical characters entered for the name. As an example, even if a user entered "ArCHeR." It would be corrected to the same form you see below, then displayed back to the user.



```
Howdy! What is your name?
Archer
Welcome to The Game Library, Archer!
```

6.2.2 Alternate Greeting

Below is an example of a user inputted name which results in an alternate greeting (When the name contains characters that are not in the English alphabet).



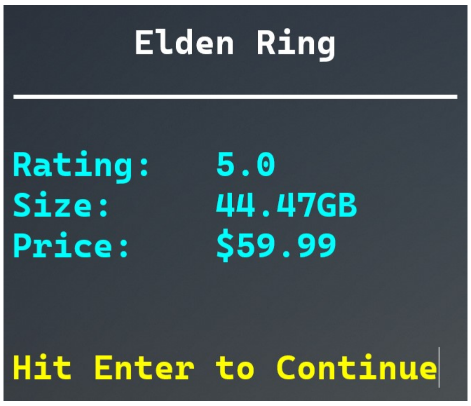
```
Howdy! What is your name?
Archer 8888888
Welcome to The Game Library, Archer 8888888!
You have an interesting name...
```

6.3 Main Menu

6.3.1 Search

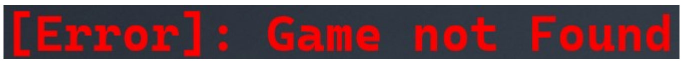
When searching, input is corrected using strip, as well as a custom global method called **nameForm**. nameForm is the same method that corrects input for displaying names in main, except in Library.py it has been modified to handle game titles.

Below is the expected output for a search done for "ElDeN RinG," which is corrected, then returns the attributes of the corresponding game.



```
          Elden Ring
──────────────────────────────────

Rating:    5.0
Size:      44.47GB
Price:     $59.99


Hit Enter to Continue
```

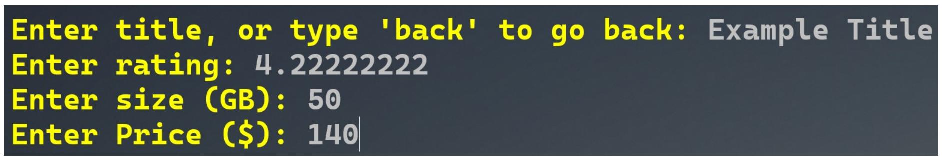
Below is the expected output for a search done on a non-existent game entry.



```
[Error]: Game not Found
```

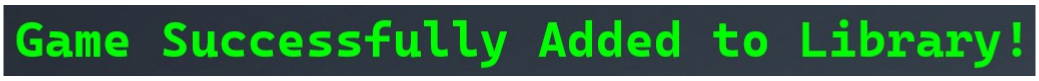
6.3.2 Add Game

Below is the expected output for a game entry added to the Library



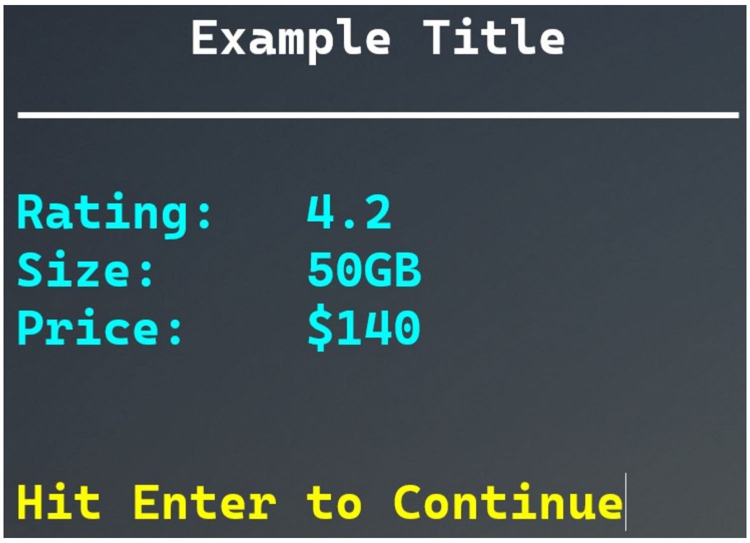
```
Enter title, or type 'back' to go back: Example Title
Enter rating: 4.22222222
Enter size (GB): 50
Enter Price ($): 140
```

Below is the prompt the user will see if the game entry has been successfully added to the library. If addition was not successful (because the title already exists within library), then the user will receive a momentary error message, then will be returned to the entry screen.



```
Game Successfully Added to Library!
```

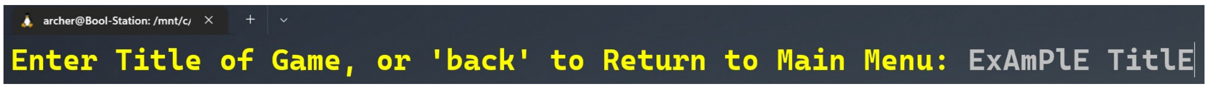
Below shows how the user can confirm the entry has indeed been added past the success screen (using the search option). Notice how the rating has been truncated to 1 decimal place, "GB" has been appended to the size, & "$" has been prepended to the price.



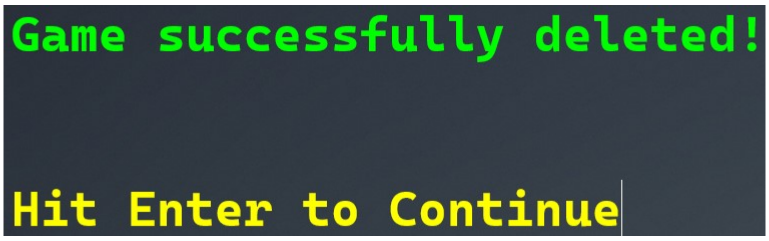
```
            Example Title
_________________________________

Rating:    4.2
Size:      50GB
Price:     $140


Hit Enter to Continue
```

6.3.3 Delete Game

Below shows the game added in the previous section being deleted

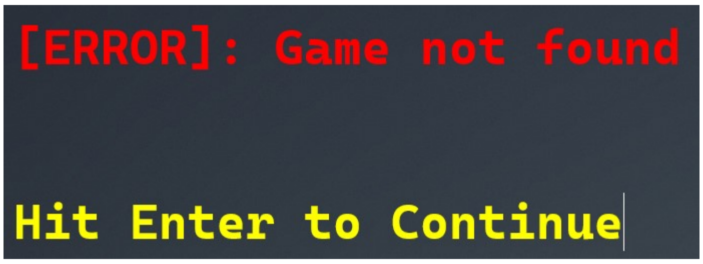


```
archer@Bool-Station: /mnt/c/
Enter Title of Game, or 'back' to Return to Main Menu: ExAmPlE TitlE
```



```
Game successfully deleted!


Hit Enter to Continue
```

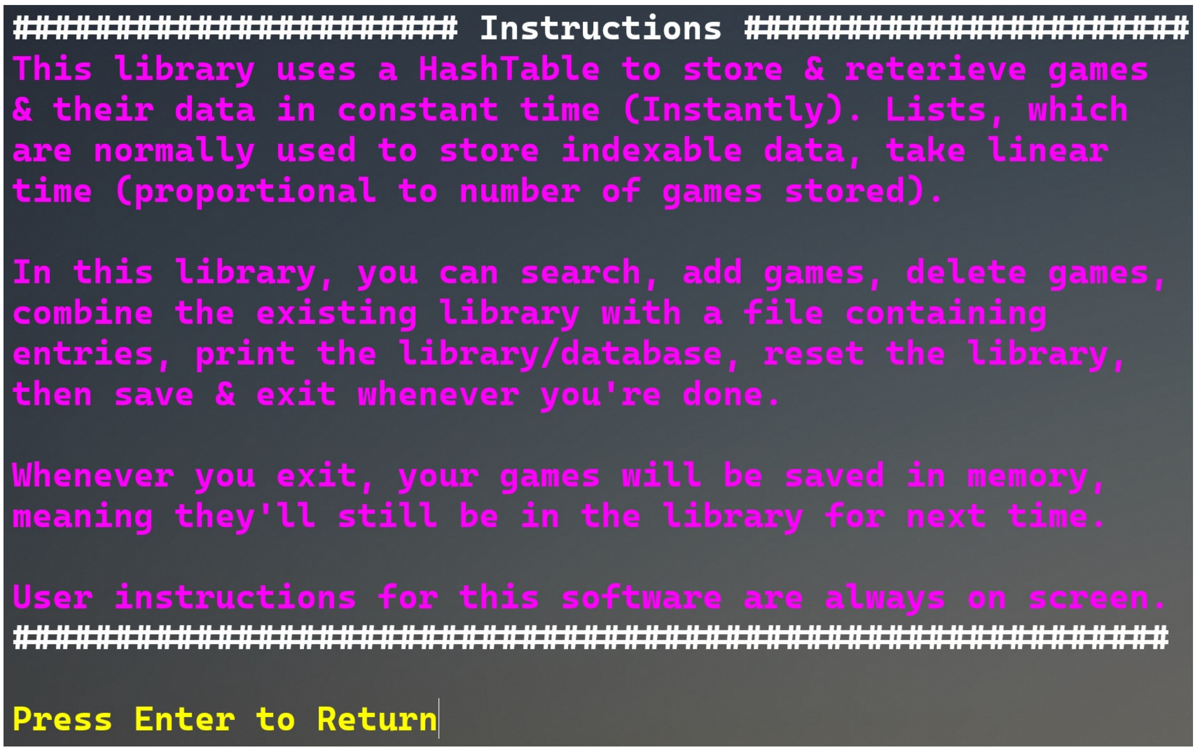
Below shows the expected output when a game which does not exist is requested for deletion



```
[ERROR]: Game not found


Hit Enter to Continue
```

6.3.4 Instructions

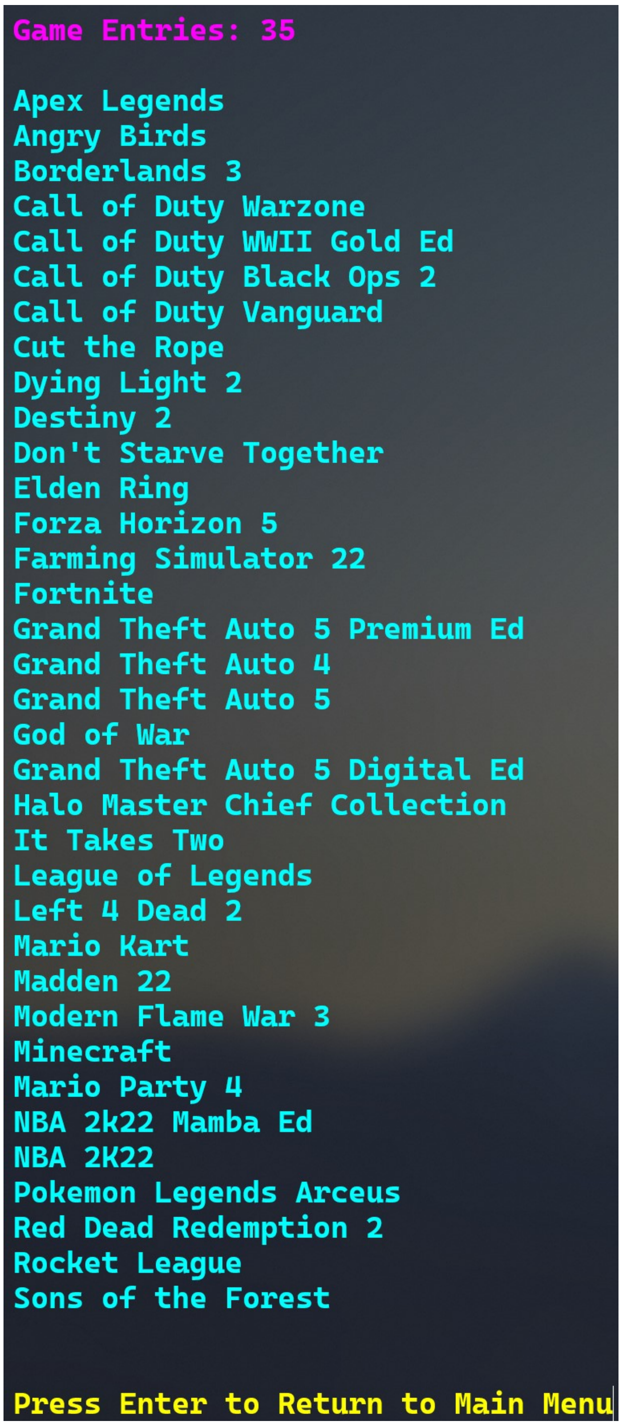
When 4 is selected, a screen will appear with text. Admittedly, the text is more of an overview than instructions, though instructions are always on screen.



```
###################### Instructions ######################
This library uses a HashTable to store & reterieve games
& their data in constant time (Instantly). Lists, which
are normally used to store indexable data, take linear
time (proportional to number of games stored).

In this library, you can search, add games, delete games,
combine the existing library with a file containing
entries, print the library/database, reset the library,
then save & exit whenever you're done.

Whenever you exit, your games will be saved in memory,
meaning they'll still be in the library for next time.

User instructions for this software are always on screen.
#########################################################

Press Enter to Return
```

6.3.5 Print Library

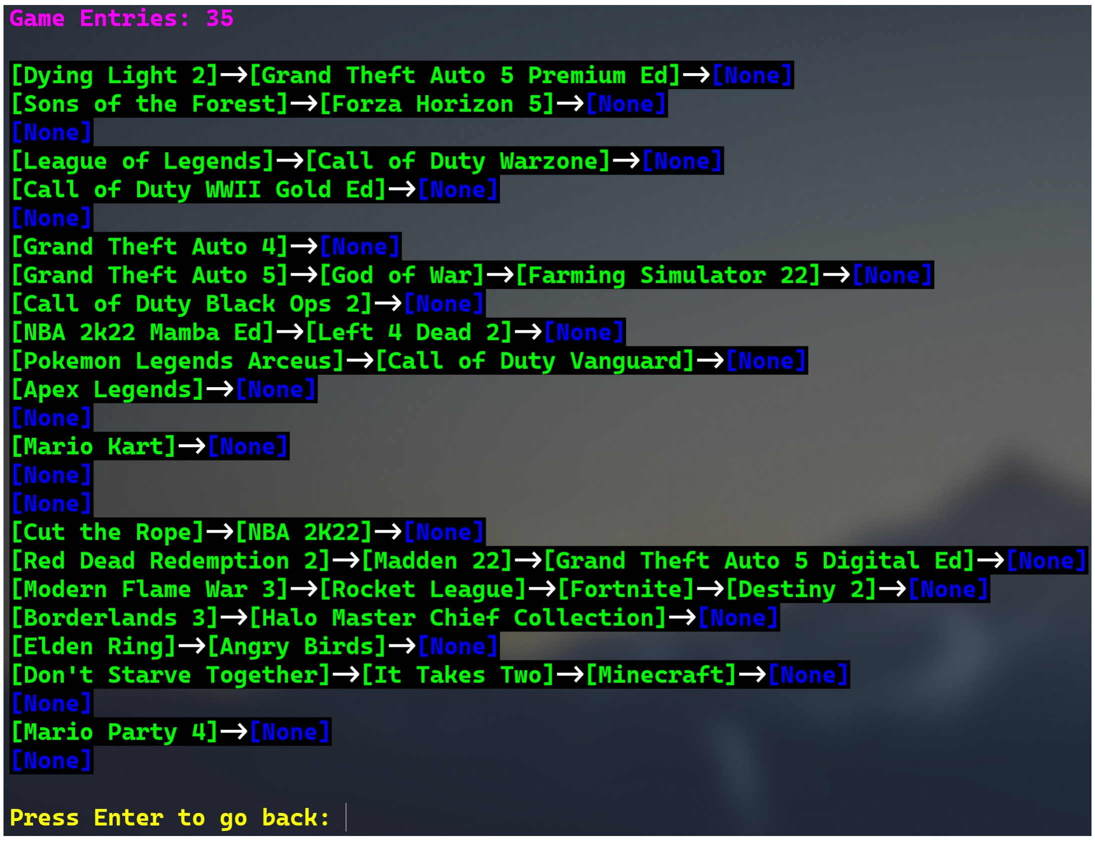
Below shows the user-friendly library print option, which prints each entry in lexicographical order by title



```
Game Entries: 35

Apex Legends
Angry Birds
Borderlands 3
Call of Duty Warzone
Call of Duty WWII Gold Ed
Call of Duty Black Ops 2
Call of Duty Vanguard
Cut the Rope
Dying Light 2
Destiny 2
Don't Starve Together
Elden Ring
Forza Horizon 5
Farming Simulator 22
Fortnite
Grand Theft Auto 5 Premium Ed
Grand Theft Auto 4
Grand Theft Auto 5
God of War
Grand Theft Auto 5 Digital Ed
Halo Master Chief Collection
It Takes Two
League of Legends
Left 4 Dead 2
Mario Kart
Madden 22
Modern Flame War 3
Minecraft
Mario Party 4
NBA 2k22 Mamba Ed
NBA 2K22
Pokemon Legends Arceus
Red Dead Redemption 2
Rocket League
Sons of the Forest


Press Enter to Return to Main Menu
```

6.3.6 Print Database

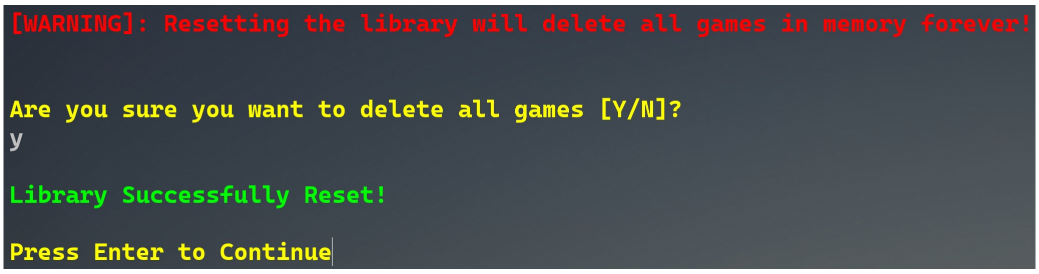
Below shows the developer-friendly view of the game Library. This view shows the hash table within the Library class as a physical representation. It's a vertical list with horizontal liked lists at each position that a collision has occurred. Slots that say "[None]" have not been indexed by a key just yet (empty linked list). Usually, the hash table would have hundreds of thousands of buckets, though for easier display purposes, it only has 25 buckets, making collisions more likely. Lastly, the "[None]" at the end of each list is correct; it is displayed to indicate when a linked list ends, & thus must be printed after the game links to accurately represent a linked list.



```
Game Entries: 35

[Dying Light 2]→[Grand Theft Auto 5 Premium Ed]→[None]
[Sons of the Forest]→[Forza Horizon 5]→[None]
[None]
[League of Legends]→[Call of Duty Warzone]→[None]
[Call of Duty WWII Gold Ed]→[None]
[None]
[Grand Theft Auto 4]→[None]
[Grand Theft Auto 5]→[God of War]→[Farming Simulator 22]→[None]
[Call of Duty Black Ops 2]→[None]
[NBA 2k22 Mamba Ed]→[Left 4 Dead 2]→[None]
[Pokemon Legends Arceus]→[Call of Duty Vanguard]→[None]
[Apex Legends]→[None]
[None]
[Mario Kart]→[None]
[None]
[None]
[Cut the Rope]→[NBA 2K22]→[None]
[Red Dead Redemption 2]→[Madden 22]→[Grand Theft Auto 5 Digital Ed]→[None]
[Modern Flame War 3]→[Rocket League]→[Fortnite]→[Destiny 2]→[None]
[Borderlands 3]→[Halo Master Chief Collection]→[None]
[Elden Ring]→[Angry Birds]→[None]
[Don't Starve Together]→[It Takes Two]→[Minecraft]→[None]
[None]
[Mario Party 4]→[None]
[None]

Press Enter to go back:
```

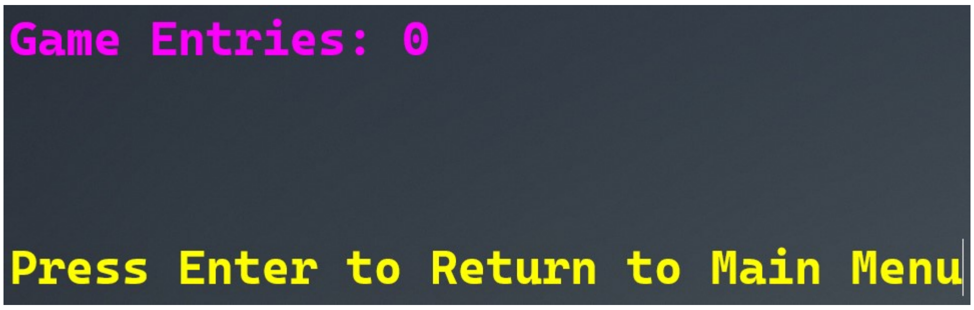
6.3.7 Delete Library

Below is the expected output for when the Delete Library option is selected, & the user inputs "Y." Not shown are the changes made to the persistent memory file, which is completely wiped of data (it now contains 0 lines of data).



```
[WARNING]: Resetting the library will delete all games in memory forever!


Are you sure you want to delete all games [Y/N]?
y

Library Successfully Reset!

Press Enter to Continue
```

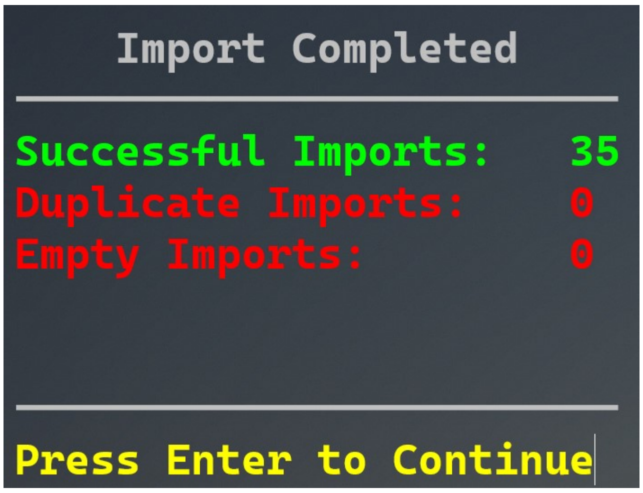
Below shows the Print Library option after the Delete Library has been used.



```
Game Entries: 0



Press Enter to Return to Main Menu
```

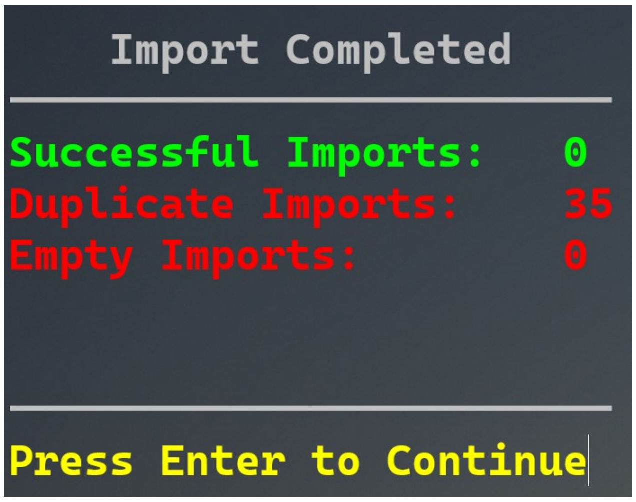
6.3.8 Import Library

Below shows the expected output for a successful import of the backup file "backup.csv" after the library was deleted in the previous section.



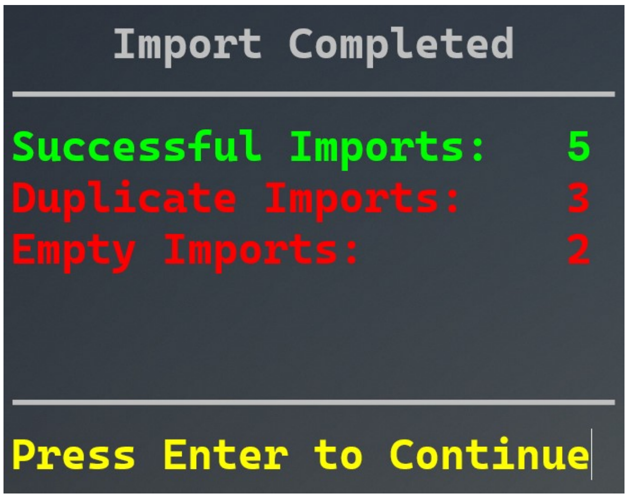
```
    Import Completed
─────────────────────────
Successful Imports:   35
Duplicate Imports:    0
Empty Imports:        0


─────────────────────────
Press Enter to Continue
```

Below shows an attempt to import the same file again, which results in 0 files being imported, as all are now duplicates.



```
      Import Completed
──────────────────────────
Successful Imports:    0
Duplicate Imports:     35
Empty Imports:         0



──────────────────────────
Press Enter to Continue
```

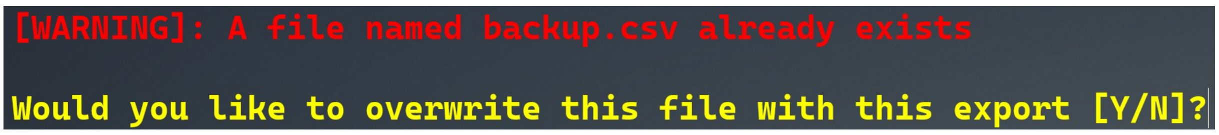
Below shows an attempt to read in a file with blank & duplicate entries (found in the "implib.csv" file).



```
      Import Completed
──────────────────────────
Successful Imports:    5
Duplicate Imports:     3
Empty Imports:         2



──────────────────────────
Press Enter to Continue
```

6.3.9 Export Library

Below shows the expected output when the user tries to export the library to a cvs with the same name of a file that already exists within the directory. If "Y" is selected, the existing file will be overwritten. If the file name does not exist in the directory, then a new file with the user inputted name will be created with all game entries present in Library's database at the time of export.

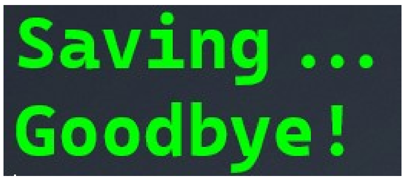


```
[WARNING]: A file named backup.csv already exists

Would you like to overwrite this file with this export [Y/N]?
```

6.3.10 Save & Exit Program

When 10 is selected, a prompt will ask if you'd like to save & exit. If "Y", then a prompt will appear, which simulates save time (saving happens in an instant due to the low amount of games, so we added a fake save screen for dramatic effect).

If 11 is selected, the prompt will warn that any changes made will not be saved, then it simply says "Goodbye" instead of "Saving..." then "Goodbye."



```
Saving ...
Goodbye!
```

7 Conclusion & Contact Info

Though there are MANY more details about this program missing from this document (including the HashTable class's near identical syntax to Python's built-in dictionary class, & how we achieved that using dunder methods), I couldn't fit it all in within the time constrains.

IF YOU HAVE ANY QUESTIONS, CONCERNS, OR RUN INTO ANY ISSUES, PLEASE EMAIL ME AT: ARCHER.SIMMONS@TAMU.EDU

```
int main
       ::      << "Hello World" <<     ::
    return 0
```