Phase 2:

Source Code

UMGC CMSC 495

Python Game Hub

Lee, Victoria (Member 1)

Mutry, James (Member 2)

Foster, Todasha (Member 3)

Chung, Dajin (Member 4)

Payne, Javon (Member 5)

Ipaye, Oluwatumininu (Member 6)

Table of Contents

[**1. Introduction 3**](#_heading=h.1r049cse9qxx)

[**2. Objectives 3**](#_heading=h.kfw99fy8ffl3)

[**3. Finalize Project Setup 5**](#_heading=h.choctwq6s0q1)

[**4. Finalize Core Functionality 8**](#_heading=h.2or20o4i5o26)

[**5. All Unit Testing and Manual Testing 11**](#_heading=h.7agr04k3xc1x)

[**6. Full Documentation and Code 19**](#_heading=h.hp8n3p74jwa5)

[**6.1. Refactor the Software 20**](#_heading=h.3j3nngn2sgps)

[**6.2. Deploy the Software 20**](#_heading=h.lalojrbizbef)

[**6.3. Maintain the Software 22**](#_heading=h.ctw24meodv2p)

[**7. Source Code (Phase 2) 24**](#_heading=h.82zwrrd8a5zj)

[**8. Conclusion 26**](#_heading=h.69gdfq9nqzh6)

[**9. APPENDIX A: 27**](#_heading=h.1fu20k3jmswi)

[**10. APPENDIX B: 32**](#_heading=h.rmgudgwbzal7)

[**11. APPENDIX C: 41**](#_heading=h.yoro9gihgaju)

[**10. APPENDIX D: 42**](#_heading=h.1i7q7gjyhvsu)

[**12. APPENDIX E: 49**](#_heading=h.dsuvdlwvm35m)

[**13. References 59**](#_heading=h.p5kku9qm9x4x)

# **Introduction**

This document represents the Phase II deliverables of our CMSC Capstone project, a Python-based multi-game application developed with the Pygame framework. Building upon the foundational work established in Phase I—including the integration of Tic-Tac-Toe, Trivia, and Breakout games within a unified interface—this phase focuses on finalizing the application’s functionality, structure, and quality. During this stage, our team has completed and verified the project environment setup, ensuring all source libraries, dependencies, and tools are properly integrated for seamless development and execution. Each game module has been refined to meet production-level quality, with final versions of all core features and mechanics implemented and tested. A critical emphasis in Phase II was placed on unit testing and documentation. All major modules have been systematically tested using Python’s unittest framework to validate correctness, handle edge cases, and ensure reliability. In parallel, we enhanced code readability and maintainability by embedding in-line documentation and external references to support future updates or team onboarding. Team collaboration played a pivotal role in identifying and resolving code inconsistencies, optimizing logic, and applying best practices. Code reviews, peer feedback, and testing cycles were conducted to uphold high standards of code quality, stability, and user experience. Overall, this phase concludes with a stable, well-documented, and fully functional application ready for peer review and final polishing. It marks a significant milestone in our development cycle as we transition toward the finalization of our Capstone project.

# **Objectives**

**Project Purpose:** The objective of this group project is to develop a Game Hub based on the Python language. The Game Hub will provide a user interface, in the form of a main menu, which will then allow users to select and play three different games also based on Python: Breakout, Tic-Tac-Toe, and Trivia. In order to keep track of the different source codes and updates made by team members, Git will be used. This will help the team manage multiple versions of code, resolve errors or conflicts, and in turn collaborate effectively. When or if needed, code will be refactored to fit the narrative of the entire project and to deliver a well-rounded product.

**Project Deliverables:** The deliverables of the project will be a functional and fully interactive Game Hub with the three games mentioned previously. The main menu will allow users to browse the selection of the games and make a choice as desired. Each game will be its own module, able to be called from the main menu’s module. This will allow for easy changes or error fixing, without full disruption to the entire package. There will be audio files used for the main menu, actions, etc. in order to create an immersive experience for users. Scores will be counted during gameplay in order to let the user take note of progress made. Users will have the ability to return to the main menu, restart or exit any game at the end of a round/session. The source code repository will be posted on GitHub for tracking version control and changes made. Components of each source code will be tested for functionality through unit testing.

**Instructions / Procedures:** The following instructions for running the Game Hub are only a summarized version of those found in the README.md file for further assistance.

* Link to README.md file: <https://gist.github.com/34c40d49bcef96c70ed35a5a80c33b92.git>
* Verify that Python is installed on your computer. If it is not, it can be downloaded from Python’s official [website](https://www.python.org/downloads/).
* Visit the repository page of the project and download the zip file **OR** clone the repository if you have a Git account. The files can be extracted from the zip file to your desired folder.
* Ensure you have a Python IDE or virtual environment that can be used to run the application. An example of a Python IDE is PyCharm, downloadable from their official website.
* Run your chosen IDE, and then install all dependencies required to run the Game Hub. This can be accomplished by using the following command prompt:

***python.exe -m pip install --upgrade pip***

***pip install -r requirements.txt***

* Open all .py files and import any packages that may be shown as missing through your IDE to ensure all necessities are put in place.
* After performing the aforementioned instructions, the Game Hub may be run by running MainMenu.py in the IDE or follow the instructions on the README.md file above for other ways to run the game.
* The README.md file should also have instructions on how to play the games.

**Functional and Non-Functional Requirements:**

* Games must be fully playable with clear rules.
* Tracking Winners, Losers, Ties, High Scores, and Lives for one or more of the games.
* To track progress on each coding section, we will use Git for Version Control.
* To track the progress of the games/UI Gam Hub testing section, we will use the concept of unit testing to ensure that each functional requirement is working during debug processing.
* Performance: The games must load within 3 seconds and run without noticeable lag.
* Score Board and lives: It should show the winner, loser, tie, the lives, and the scoreboard. If not, then the coder must debug.
* Usability: The user interface must be simple and intuitive, with a responsive design
* To track progress on each coding section, we will use Git for Version Control.
* To track the progress of the games/UI Gam Hub testing section, we will use the concept of unit testing to ensure that each non-functional requirement is working during debug processing.

**Process Model:** This team group will be following the SDLC process model to help develop and deploy the Python Game Hub. This is a waterfall methodology that allows the team to plan in the early stages to prevent major design flaws from developing and below is the SDLC (GeeksforGeeks, n.d.) According to GeeksforGeeks (n.d.) and UMGC (n.d.), the Waterfall model is a linear and sequential approach, where each phase must be completed before the next one begins. In the case of the Python Game Hub, the Waterfall methodology allows for a structured, step-by-step approach to deliver each component of the project on time because the features and games are clearly defined from the outset. **As of now for Unit 6, we will be on the Verification and Maintenance.** In Unit 7, we will recap the Maintenance part with the User Guide Documentation week. Then Unit 8 should be using all parts of the Waterfall method, making sure all parts are completed to submit the documentation and code. The testing and Git version control processes are further detailed in APPENDIX D and APPENDIX E. The Milestones and Gnatt chart updates with the Contribution Report are in APPENDIX A, APPENDIX B, and APPENDIX E. A shows the overall timeline due dates and how our progress compares with units 1-8. B shows the breakdown of each task/milestones completed and costs. APPENDIX C is referring to the delays and emergencies so far to push back the code. APPENDIX E shows the breakdown of the Git issues/milestones completed for that week (screenshots).



# **Finalize Project Setup**

**Project Setup:** This portion is in the code section, scroll to find the source code and find the README.md file and click on it to read the code. That code explains all the steps to install, run, and make the application work. It also includes the documentation and how to play the games, the unit tests, manual tests, GitHub Actions and many more.

**README.md:** The link below renders the code to README.md. This readme file should show how to install, run the application, and install any imports. It will also explain how to play the games and how the unit tests are done. It also teaches the user how the unit test is processed.

<https://gist.github.com/34c40d49bcef96c70ed35a5a80c33b92.git>

**requirements.txt:** The code below renders the code of the requirements.txt. This file should allow the user to import and install dependencies into the PyCharm environment in order to run the application. If it does not work, the user has other options to install them following the directions on the README.md file. If it doesn't work the user must also install these: random; math; time; json; unittest.

**Code of the requirements.txt file:**

pygame

pyautogui

numpy

### **Python Game Hub’s Development Environment:**

The Python Game Hub was developed in a cross-platform environment supporting Windows, macOS, and Linux. The project was implemented using Python 3.9 or later and developed in IDEs like VSCode and PyCharm, as well as directly through the command line, terminal, or PowerShell. The primary game interface and functionality were built using Pygame for rendering graphics, animations, and audio. The development environment ensured compatibility across operating systems, with a display resolution of at least 600x400 to maintain visual consistency. Continuous integration was handled via GitHub Actions, running automated unit tests on different environments (Windows, Mac, Ubuntu).

**Python Game Hub’s Completion and Verification of All Project Steps Setup:**

All major development and setup phases have been completed and verified. This includes:

* Final implementation of all three mini-games (Tic Tac Toe, Trivia, and Breakout).
* Development of a functional and interactive Main Menu (MainMenu.py) to navigate between games.
* Execution of unit tests using unittest with 13 tests in the main branch and 24 tests in the James branch.
* Successful integration and validation of core functionalities including AI mechanics, question validation, and collision detection.
* Manual testing verified via a structured 20-point checklist in the CMSC 495 Project Test Plan.xlsx. This accounts for integration and user-interaction testing.
* Ensured that all necessary assets (e.g., questions.json, sound files like brick.wav, wall.wav) were properly linked and functioning.
* Refer to APPENDIX A to see the Tasks, milestones, and overall completion progress comparing it for Units 1-8.
* Refer to APPENDIX B to see the expansion of the job contribution, tasks, and milestones completed.
* Refer to APPENDIX D to see other Testing expansions completed.
* Refer to APPENDIX E to see Git (version control) issues, tasks, milestones completed.

### **Python Game Hub’s Coding Tools:**

* **Python 3.9+**: Primary programming language.
* **Pygame**: Core library used for game logic, graphics, and audio.
* **PyAutoGUI**: Used for handling GUI prompts (e.g., replay options).
* **Numpy:** Used for handling timer, or mathematics code.
* **Unittest**: Python’s built-in framework for unit testing.
* **Git**: For version control and branch management.
* **GitHub**: For collaboration and CI/CD integration via GitHub Actions.
* **IDE Tools**: PyCharm, or similar environments like Visual Studio Code, were used for writing and debugging code.

### **Python Game Hub’s Source Libraries:**

* python 3.x – this is the basic library for python including pip to install imports and ve.
* pygame – For all game-related graphics, event handling, and sound.
* json – To parse and load question data for the Trivia game.
* math – Used for calculations in graphics (e.g., countdown timer arcs).
* random – For shuffling trivia questions and randomized game behavior.
* sys, time – General system-level operations and timers.
* unittest – Automated testing framework.
* numpy – Used where numerical array management is required.
* pyautogui – For GUI prompts like replay confirmation screens.

### 

### 

### **Python Game Hub’s Dependencies:**

**README.md:** The link below renders the code to README.md. This readme file should show how to install, run the application, and install any imports. It will also explain how to play the games and how the unit tests are done. It also teaches the user how the unit test / manual tests are processed.

<https://gist.github.com/34c40d49bcef96c70ed35a5a80c33b92.git>

**requirements.txt:** The code below renders the code of the requirements.txt. This file should allow the user to import and install dependencies into the PyCharm environment in order to run the application. If it does not work, the user has other options to install them following the directions on the README.md file. If it doesn't work the user must also install these: random; math; time; json; unittest.

**Code of the requirements.txt file of the essential dependencies include:**

pygame

pyautogui

numpy

Note: All required dependencies were either manually installed using pip or listed in the requirements.txt file for easy setup. The essential dependencies include:

To install, the following command can be used:

*pip install -r requirements.txt*

These dependencies were essential for enabling graphical rendering, UI interaction, audio support, and automation tasks throughout the game experience.

**Git, Unit Testing, and Debugging:** The project will use Git for version control to manage code changes and facilitate collaboration among developers. Each developer will create feature branches for specific tasks, making incremental changes and committing them with descriptive messages. After completing their work, they will push their changes to the remote repository and create a pull request (PR) for code review. The project lead will review and approve the PRs, merging them into the main branch once the code meets the required standards. Unit testing will ensure the core game mechanics, user interface, and features like high scores function correctly, while debugging will address any issues in gameplay, UI, performance, and cross-platform compatibility. Regular updates will be made to keep the main branch stable and up-to-date, and any merge conflicts will be resolved promptly to maintain consistency. Note: The testing and Git version control processes are further detailed in APPENDIX D and APPENDIX E. D shows more breakdown on the Testing section while E shows the breakdown of the Git issues/milestones completed for that week. Meanwhile APPENDIX C is referring to the delays and emergencies so far to push back the code.

# **Finalize Core Functionality**

**Python Game Hub’s Production Level Core Functionalities:**

**Implementation:** The implementation phase involved building a multi-game desktop application in Python using the Pygame library, which provides robust support for real-time graphics, audio playback, and user input handling. The project follows a modular design, where each mini-game—Tic Tac Toe, Trivia, and Breakout—is implemented as a standalone Python module and integrated into the application through a centralized menu system. The completions of the tasks/milestones are in APPENDIX A and in APPENDIX B and in APPENDIX E. A shows the overall timeline due dates and how our progress compares with units 1-8. B shows the breakdown of each task/milestones completed and costs. E shows the breakdown of the Git issues/milestones completed for that week(screenshots). For E, the screenshot of it are at the end of APPENDIX E. Meanwhile APPENDIX C is referring to the delays and emergencies so far to push back the code.

**Main Interface:** The main interface (MainMenu.py) provides a visually engaging user experience, featuring a gradient background, hover-sensitive buttons with rounded borders, and click interactions. Each menu option corresponds to a game module, and navigation is managed via a dedicated navigate() function that launches the selected game. Background music plays continuously in the main menu, and is stopped and resumed as users switch between games.

**Breakout:** in Breakout, three core classes—Ball, Striker, and Block—are used to model interactive game entities. Collision detection is implemented between the ball and blocks or walls, with sound effects triggered using the pygame.mixer.Sound() module. Block health varies by color, and each block is removed upon destruction, incrementing the player’s score. A game over condition is triggered when the player loses all lives, prompting a replay prompt using a graphical alert from the pyautogui library.

**Tic Tac Toe:** In Tic Tac Toe, the implementation includes a graphical board built with grid lines and drawn symbols (X and O). Player moves are registered through mouse clicks. The AI opponent uses the minimax algorithm to evaluate all possible future game states and choose the optimal move. The AI logic is recursive and includes base cases for win, loss, and draw conditions. After each round, the game displays a result message and a custom-designed “Play Again” button, which resets the game state upon interaction.

**Trivia:** Trivia includes a collection of multiple-choice questions that are randomly shuffled at runtime. The user can play with 1 player or 2 players where it stores the string user’s name and score tracking for 1-2 player mode. Each question features four answer options displayed on screen. The game uses a circular donut-style countdown timer rendered using trigonometric functions to visually indicate the remaining response time. If the player selects the correct answer within the time limit, their score increases; otherwise, the question is skipped. After the final question, a “Game Over” screen appears with a prompt to play again.

**Sound and Media:** sound and media assets (e.g., background music and effects for success, failure, and interactions) are handled efficiently using the pygame.mixer module. The application is built to be scalable and maintainable, allowing for easy addition of new games or enhancements to existing ones. Iterative development practices were followed, allowing for continuous testing, feedback, and refinement at each step of the build process.

**Python Game Hub’s Critical Chosen Features:** The Python Game Hub project integrates a specific selection of critical features designed to enhance user experience, modularity, and gameplay variety. At its core, the Game Hub offers a centralized Main Menu interface built with Pygame, allowing users to seamlessly navigate between three mini-games: Tic Tac Toe, Breakout, and Trivia. Each game is structured in its own file to support independent execution, testing, and debugging. The Tic Tac Toe game emphasizes simple logic and turn-based strategy, the Breakout game introduces real-time physics and arcade-style action, while the Trivia game utilizes external JSON files to deliver dynamic, category-based questions. Additional key features include unit testing for each game module, ensuring code reliability, and scalable design that allows new games to be integrated easily into the menu. This focus on interactivity, modularity, and quality assurance reflects the team’s commitment to both functionality and future expandability.

**Python Game Hub’s Critical Chosen Algorithms:** Our team will utilize Software Engineering, Data Structures and Algorithms, and some LeetCodes to explain our algorithms chosen per each game and py file. Refer to (Leetcode, n.d.); (Tsui, Karam, & Bernal, 2014); (Williams & Zhang, 2020); (Kleinberg & Tardos, 2006); (Erickson, n.d.); (Mount, n.d.); (Nievergelt, n.d.); (Liang, 2023) for more details on these concepts.

**MainMenu.py:** The MainMenu.py algorithm serves as the central UI for navigating between multiple games—Tic Tac Toe, Trivia, and Breakout—implementing a modular and event-driven architecture commonly seen in software systems and frontend development. The goal behind this was to make it as simplistic as possible, then integrating more code changes to the algorithm. It uses a main loop to constantly listen for mouse input events, similar to how real-time systems or frontends handle input. It utilizes the event-handling pattern, which is similar to how real-time systems or game engines manage input, similar to callback-based interfaces in GUI programming or web development. Each menu option is rendered as a text-based button, and bounding box collision detection (check\_hover) is used to detect mouse hover or clicks—similar to LeetCode’s problems. The menu options are stored in a list and dynamically drawn with enumerate, a common iteration pattern. When clicked, the selected option is passed into a command-style dispatch function (navigate), which launches the appropriate game, following modular design and separation of concerns. The background gradient is generated using a loop, resembling procedural rendering techniques. Overall, the MainMenu.py algorithm applies key software engineering practices like clean UI separation, modularity, and basic algorithmic logic, similar to LeetCode-style problem solving and real-world frontend design.

**Breakout.py:** The core algorithm in the breakout.py game is centered around real-time 2D collision detection and dynamic game state management, which maps closely to classic software engineering and LeetCode concepts like state machines, and list manipulation during iteration. The goal behind this was to make it as simplistic as possible, then integrating more code changes to the algorithm. The game loop serves as the critical engine, continuously checking for user input (keyboard or mouse), updating the ball and paddle positions, and managing collisions. The most important algorithm is the collision detection mechanism, where the game uses bounding box logic to determine if the ball intersects with the paddle or any of the blocks. This mirrors problems like LeetCode’s problems and for this class, each block has health, and on each hit, its health is reduced; if it reaches zero, the block is removed from the list of active blocks. To safely modify the list during iteration, the code uses a slice (listOfBlocks[:])—a practice common in in-place list modification scenarios seen in algorithmic challenges. The game also tracks the number of lives and resets the ball when it falls below the screen, acting like a finite state machine that responds to game events like “ball missed,” “block destroyed,” or “game over.” Additionally, OOP principles are applied to encapsulate behavior for the Striker, Ball, and Block classes, following software design best practices. Overall, the game logic elegantly combines collision detection algorithms, list handling patterns, and state management—all of which are foundational in both game development and algorithm design interviews.

**TicTacToe.py:** The core of TicTacToe.py revolves around implementing a turn-based game loop using 2D arrays (NumPy matrices) to represent the board state—an ideal fit for grid-based problems commonly seen in LeetCode. The goal behind this was to make it as simplistic as possible, then integrating more code changes to the algorithm. The game uses a Minimax algorithm with recursion and backtracking for the AI, a classic strategy optimization technique often used in game theory and is also demonstrated similarly in Leet Code’s problems. Win checks use row, column, and diagonal traversal, leveraging NumPy slicing for clean and efficient logic. From a software design standpoint, the code cleanly separates concerns: drawing (draw\_grid, draw\_marks), game logic (check\_win, ai\_move), and UI flow (display\_result, wait\_for\_restart), following the Single Responsibility Principle (SRP). Event handling in the main loop reflects finite state machine behavior—tracking whether it’s the player’s turn, checking for win/draw conditions, and switching to AI control. Overall, this game showcases recursion, grid traversal, and AI logic—all central topics in both algorithm interviews and software engineering design.

**Trvia.py:** The critical algorithm logic of Trivia.py is managing trivia questions stored in the Questions.json file, leveraging key data structures like dictionaries (for categories and questions) and lists (for storing answers and scores). The goal behind this was to make it as simplistic as possible, then integrating more code changes to the algorithm. The game randomly selects questions using the random.shuffle and random.sample methods, which are similar to LeetCode problems. The trivia.py algorithm incorporates key software engineering principles, data structures, and algorithm concepts. It uses modularity by organizing the code into functions, ensuring clear separation of concerns and easier maintainability. Error handling is implemented with try-except blocks, which helps the game gracefully handle issues like missing files or invalid JSON data. The program follows an event-driven programming model, responding to user input such as mouse clicks and key presses to update the game state, demonstrating strong state management techniques. From a data structures perspective, the game efficiently utilizes dictionaries for storing questions and player scores and lists for answers and player names, ensuring quick lookups and easy data handling. The game also employs randomization algorithms like random.shuffle() and random.sample() to select random questions for each player, which mirrors common algorithmic techniques for random selection and sampling. Additionally, the timer functionality demonstrates real-time constant-time operations, keeping track of countdowns. Overall, the game blends software engineering best practices with fundamental algorithmic techniques and data structures.

# **All Unit Testing and Manual Testing**

**Unittest (Python file):** During Unit 5, we have implemented all the possible unit tests and have not implemented any more due to overengineering and repetition. We have imported the Unit 5 Testing to Unit 6. To make up for this, there is now new GitHub Actions and Manual Testing to make up for it.The test.py script for the Python Game Hub is structured to validate a wide range of scenarios across the application’s three core games—Tic Tac Toe, Trivia, and Breakout—as well as the main menu interface. Functionality tests are designed to ensure that each component behaves as expected under normal use. These include verifying correct navigation between menu options, detecting win and draw conditions in Tic Tac Toe, registering block collisions and scoring in Breakout, and managing answer selection and timeouts in the Trivia game. The tests confirm that users are able to interact with the games fluidly, and that appropriate responses (such as displaying results or progressing to the next question) occur in real time.

Performance testing ensures that the system remains stable and responsive even under stress or edge cases. For example, Breakout is tested for how it handles rapid, simultaneous block collisions, and Trivia is tested for behavior when the full list of questions has been answered. These tests also check whether repeated use of sound effects—such as for ball bounces or correct answers—maintains consistent performance without causing lag, audio distortion, or crashes. Additionally, the use of pop-up alerts in Breakout’s game over state is validated for timing and correctness.

Usability tests focus on the overall user experience. These tests examine whether visual elements like menu highlights or countdown timers provide clear feedback to the user. For instance, hovering over menu options should change their appearance to indicate interactivity, and the donut-shaped timer in Trivia should visually warn the player as time runs out by changing colors from green to yellow to red. Tests also ensure that intuitive controls—such as using both keyboard and mouse for the Breakout paddle—function seamlessly. “Play Again” options at the end of games are tested to make sure they correctly restart the game or exit without unexpected behavior.

Altogether, the tests in test.py provide a thorough quality assurance framework for the Game Hub. They not only validate that the individual games work as intended but also that the overall software provides a consistent, user-friendly experience with reliable performance. These tests are crucial for identifying bugs early and ensuring the software is polished and production-ready.

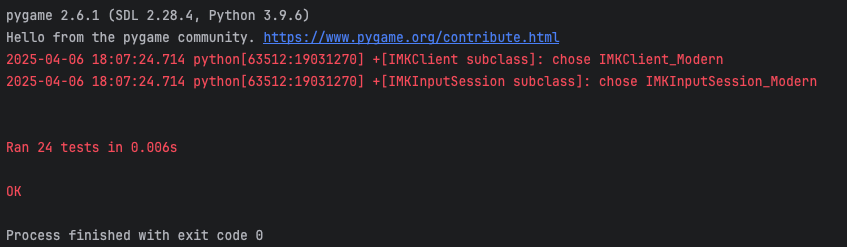
**test.py:** The link below renders the old **test.py from unit 4 (main branch)**. This file contains all the Unit Testing code and the 13 ways of testing our application using the import unittest. We added this to unit 5 to show we used GItHub ACtions to run it through and also confirm it from past testing of the IDE in unit 4.

<https://gist.github.com/VictoriaRaven/e1c8523412d6f895a7a91fc55c90b8df>

**test.py:** The link below references the new version of **test.py (james’s branch) from unit 5**. This file serves as the centralized unit testing module for the application and utilizes Python’s built-in unittest framework. It contains a total of 24 unit tests, structured as follows: 5 tests for the Main Menu, 6 for the Trivia module, 5 for Tic-Tac-Toe, and 8 for Breakout. Each test suite is implemented using the import unittest, ensuring modular and maintainable validation of core application components. This is run through James’s branch and directly through IDE or GitHub Actions.

<https://gist.github.com/d214163413d1dc79a18379b67e661d79.git>

**Screenshot:**

****

**GitHub Actions Testing with .yml files (6 of them):** This link below renders the code to the updated **.yml** files to automate the unit tests for our application. It is highly recommended that you run the test.py following the instructions of the IDE, but this method will also confirm it as a double checker to make sure it works through Git’s (CI/CD) pipeline. GitHub Actions Automated testing was tested and created by Victoria to confirm and validate the unittest of James’s test.py files. There are three GitHub Actions for the old test.py file in the main branch and three GitHub Actions for the new test.py in the james branch (test.yml, test2.yml, test3.yml, testJames.yml, testJames2.yml, testJames3.yml). The first three without James label should test the old test.py on the main branch with 13 unittest passed. The next three with the James label should test the new test.py on the James branch with 24 unittest passed. This should allow our team to double check with Jame’s manual IDE python runs for the test.py files and make sure it is working. The test.py (old), test.py (new), and CMSC 495 Project Test Plan.xlsx were created and tested by James and the GitHub Actions was made by Victoria while integrating it to the branches and test.py files.

Link to the test.yml file:

<https://gist.github.com/0caa921374cb8b454b3e0519fdbe7267.git>

Link to the test2.yml file:

<https://gist.github.com/828662e48bdb01e9c6dfeaa800fdbc51.git>

Link to the test3.yml file:

<https://gist.github.com/6bc3545fec63e50b31b83bc587992c54.git>

Link to the testJames.yml file:

<https://gist.github.com/83ddfc5588f6dc35678e9f3f61e30662.git>

Link to the testJames2.yml file:

<https://gist.github.com/7ea415e4d4c26de0378e084cb7392390.git>

Link to the testJames3.yml file:

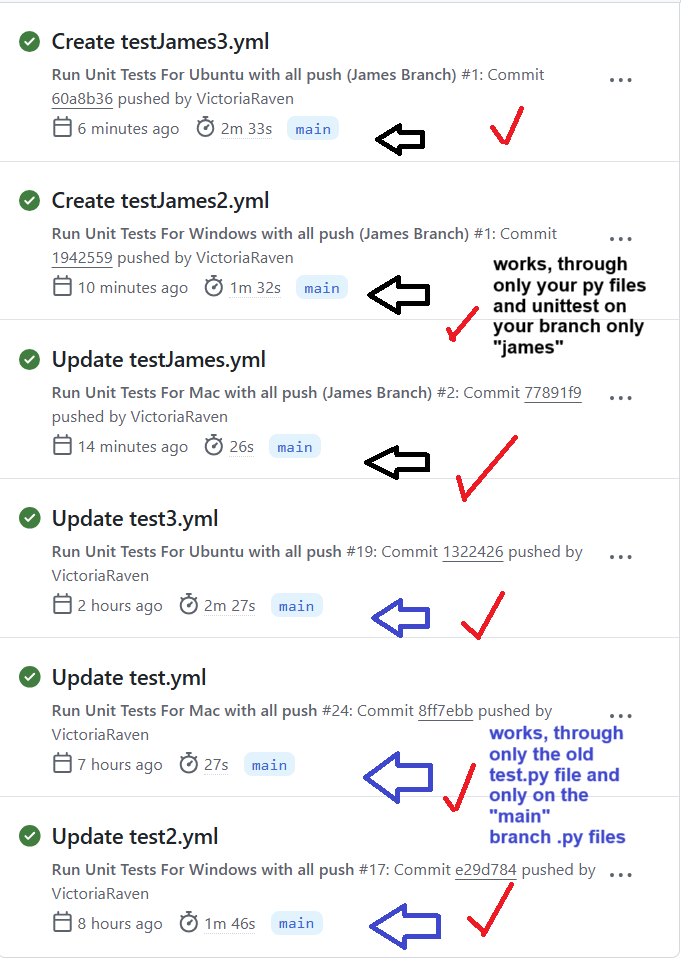
<https://gist.github.com/9ebc800353d91b027d9912409596f271.git>

Link to the old test.py file: <https://gist.github.com/VictoriaRaven/e1c8523412d6f895a7a91fc55c90b8df>

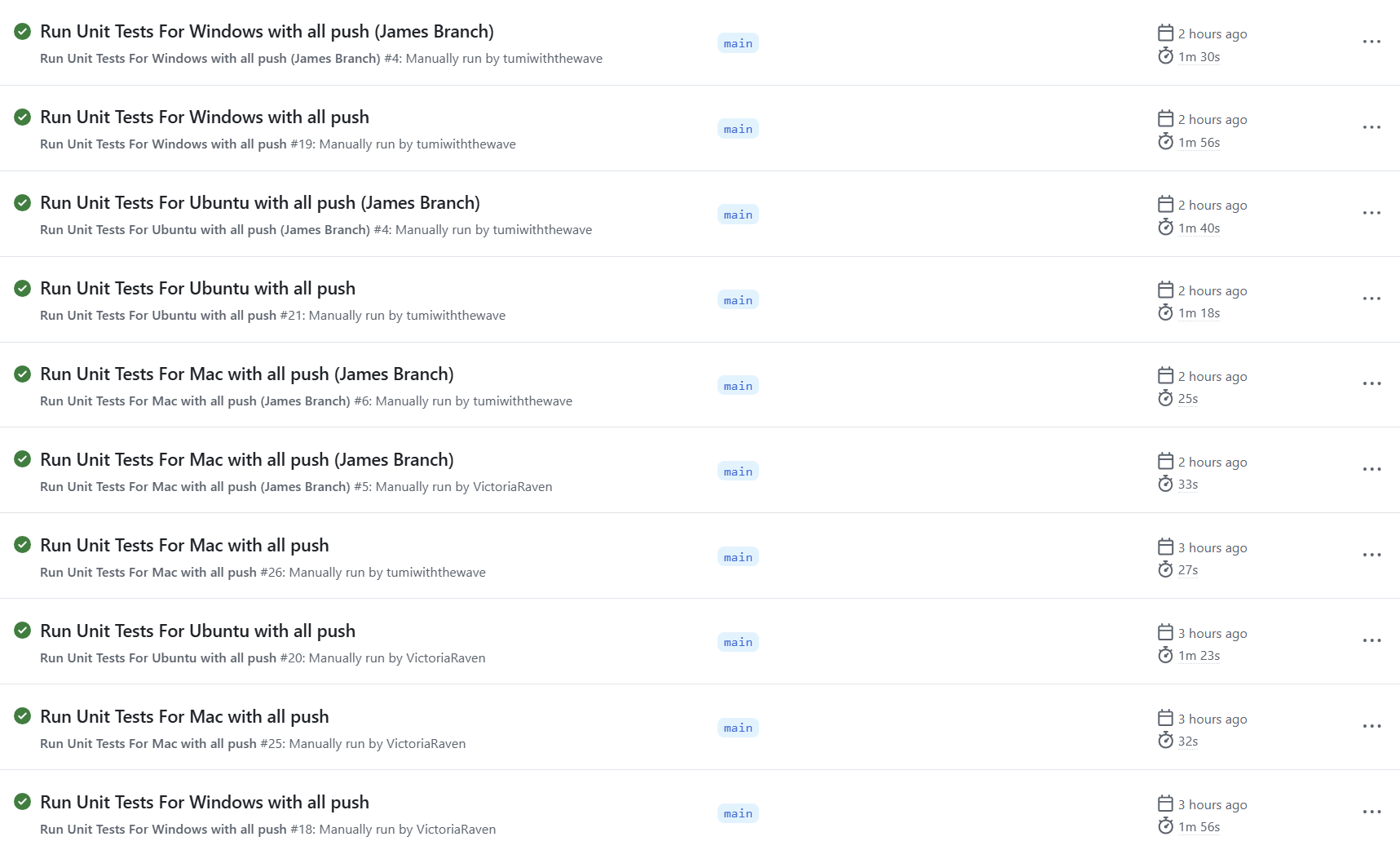
Link to the new test.py file:

<https://gist.github.com/VictoriaRaven/454a8b7026c67efb9e07d562688357fd>

**Screenshots of all 6 .yml files and that it works with the old test.py and new test.py:**



**Additional GitHub Action ScreenShots testing on all ends:**

****

**Refer to** [**APPENDIX D**](#_heading=h.ptzpvhv1n8uq) **for more Screenshots that the unit tests work for all 6 showing proof.**

**Git (Version Control) (main) branch only (Unit 6):** This test should be in the .github/workflows of files: **testFiles.yml** and **testFiles2.yml**. The first one tests if the main branch has all the core game files (detects it’s not missing/exists) for Mac and Ubuntu. The second one tests if the main branch has all the game files (detects it’s not missing/exists) for Windows. It uses **Node.js** and **Git Actions commands(Markdown)** to check if the files exists.These two tests ensure that our members when merging did not change the file names or delete/remove it from the main. This ensures that there are no core files missing when prepping for pull/push/merge, and deployment for Unit 8.

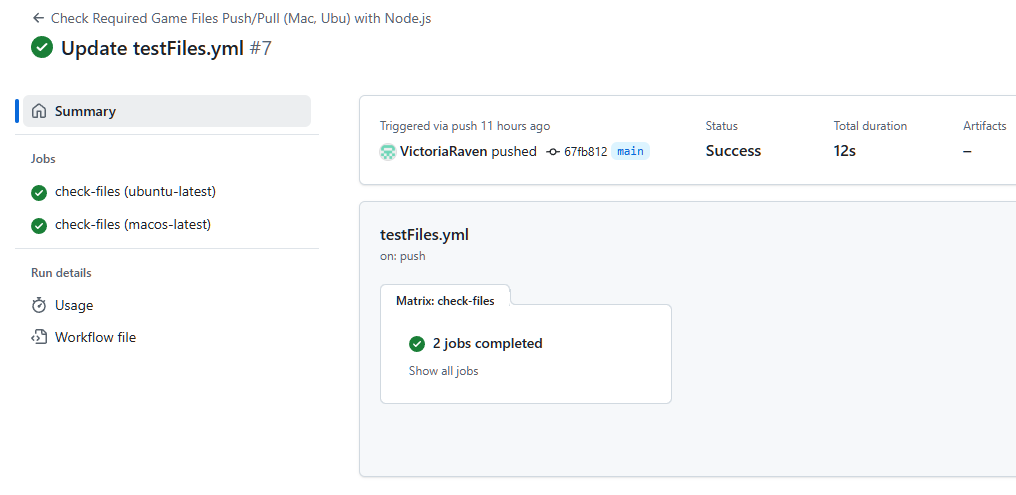
Link to testFiles.yml file:

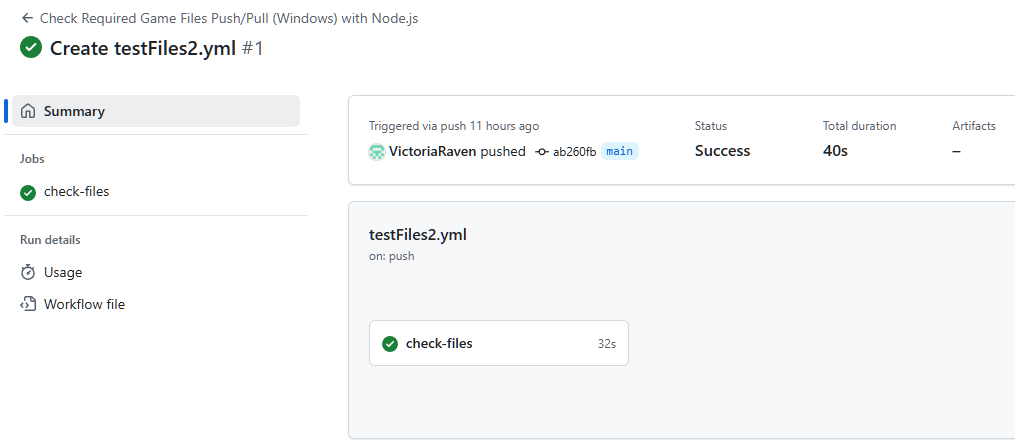
<https://gist.github.com/871efc7129b232b85c28be56d0025b5d.git>

Link to testFiles2.yml file:

<https://gist.github.com/d392c23b4b94208f4fd33d800c1ff763.git>

**ScreenShots testing:**

****

****

**Refer to** [**APPENDIX D**](#_heading=h.ptzpvhv1n8uq) **for more Screenshots of testFiles.yml and testFiles2.yml outputs.**

**Manual Testing (Excel) (Unit 5 and 6):** The manual testing plan for the Python Game Hub covers the core components of the application: the main menu interface and three mini-games—Tic Tac Toe, Trivia, and Breakout. Each section is carefully designed to ensure the functionality, responsiveness, and usability of the software from a user perspective. This also accounts for the integration testing and the user-interaction testing. The main menu serves as the central navigation point, so tests focus on verifying that each menu option leads to the appropriate game or action. Visual feedback through hover effects and background music playback are also tested to ensure they enhance user experience. We decided to implement these ones as well as adding more unit tests would be over engineering it since our testers already implemented all core parts.

In the Breakout game, the tests are centered around paddle movement, collision mechanics, scoring, and game over conditions. Manual testers are instructed to interact using both the keyboard and mouse to verify paddle responsiveness. Block collisions are tested to confirm score updates and audio feedback, while game-over conditions are validated through life depletion and proper restart behavior.

For Tic Tac Toe, the tests assess the visual rendering of the grid and the core game loop: player interaction, AI responses, and outcome detection. Scenarios for player wins, AI wins, and draws are tested to confirm that result messages are correctly displayed and that the “Play Again” feature resets the board and game state as intended.

The Trivia game is tested for the display of randomized questions, user answer selection, countdown timers, and scoring. Users are instructed to interact with both correct and incorrect answers, and to allow timers to expire to test automatic progression. The final score display and restart functionality are checked to ensure a smooth end-game experience. Overall, these tests help verify both the technical correctness and the overall playability of the Game Hub experience.

**CMSC 496 Project Test Plan.xlsx (Manual Testing):** The link below provides access to the Excel sheet used for manual testing, following the methodology outlined in the Unit Week Learning Resources. This document contains a total of 20 manual test cases, distributed evenly across the application modules: 5 for the Main Menu, 5 for Trivia, 5 for Tic-Tac-Toe, and 5 for Breakout. Each test entry includes test steps, expected outcomes, actual results, and pass/fail status, ensuring comprehensive validation of user interactions and application behavior.

**Link is below; there is no screen shots as all data is in the EXCEL:**

[CMSC 495 Project Test Plan.xlsx](https://docs.google.com/spreadsheets/d/1MhQiBjRPqWWcELhFAi9Y0fcVixZCgKGo/edit?usp=sharing&ouid=104007192992096171932&rtpof=true&sd=true)

**Refer to the Test Documentation(Excel) to see the Manual testing results and the Unit tests.**

**Test Documentation(Excel):** All testing artifacts will be documented within this link file [CMSC 495 Project Test Plan.xlsx](https://docs.google.com/spreadsheets/d/1MhQiBjRPqWWcELhFAi9Y0fcVixZCgKGo/edit?usp=sharing&ouid=104007192992096171932&rtpof=true&sd=true) containing detailed specifications for each test case. Each entry will include the expected outcome, actual result with pass/fail status, actions performed during testing, and the specific software module or component under evaluation.

**All Testing Procedures:**

**README.md:** The link below renders the code to README.md. This readme file should show how to install, run the application, and install any imports. It will also explain how to play the games and how the unit tests are done. It also teaches them unit tests /manual tests process.

<https://gist.github.com/34c40d49bcef96c70ed35a5a80c33b92.git>

**requirements.txt:** The code below renders the code of the requirements.txt. This file should allow the user to import and install dependencies into the PyCharm environment in order to run the application. If it does not work, the user has other options to install them following the directions on the README.md file. If it doesn't work the user must also install these: random; math; time; json; unittest.

**Code of the requirements.txt file:**

pygame

pyautogui

numpy

**test.py:** The link below references the new version of **test.py (james’s branch) from unit 5**. This file serves as the centralized unit testing module for the application and utilizes Python’s built-in unittest framework. It contains a total of 24 unit tests, structured as follows: 5 tests for the Main Menu, 6 for the Trivia module, 5 for Tic-Tac-Toe, and 8 for Breakout. Each test suite is implemented using the import unittest, ensuring modular and maintainable validation of core application components. This is runned through James’s branch and directly through IDE.

<https://gist.github.com/d214163413d1dc79a18379b67e661d79.git>

**Purpose:** The purpose of test.py is to automatically validate the functionality and integrity of the Python Game Hub’s key modules, including Main Menu, Tic Tac Toe, Trivia, and Breakout. The script ensures that UI behavior, game mechanics, logic processing, and data interactions function as expected under typical and edge-case conditions. These unit and integration tests are essential for maintaining code reliability during development and for verifying system consistency across updates.

**Steps/Procedures:**

GitHub Actions (24 unittest) (new test.py) (Skip if you do not have access to the Repo):

1. Go to <https://github.com/javonpayne100/CMSC495Capstone>
2. Go to Actions Tab
3. If Workflow is Disabled, Enable it
4. Click on the Workflow [1) Run Unit Tests For Mac with all push (James Branch); 2) Run Unit Tests For Windows with all push (James Branch); 3) Run Unit Tests For Ubuntu with all push (James Branch)]
5. Go to the right side of “This workflow has a workflow\_dispatch event trigger.”
6. Click on Run Workflow
7. Make sure it’s on Main branch (the yml files auto runs on separate branches even if run through main)
8. Click Run Workflow (green to confirm)
9. Refresh page to see the GitHub Action process and if green it passes
10. Environments runs (Win, Mac, Linux), but Ubuntu has glitches due to Git’s ongoing issue.
11. Now Repeat Steps (1-10) ONLY if you want to test out the old test.py with 13 unittest.
    1. On step 4 choose these workflows: [ Run Unit Tests For Mac with all push; Run Unit Tests For Windows with all push; Run Unit Tests For Ubuntu with all push ]

Command line, terminal or powershell steps:

1. Download the code from github in the “james” branch. Refer to README.md for help.
2. Navigate to the project directory
3. Ensure installation of pygame, numpy and pyautogui using “pip install pygame numpy pyautogui” or install dependencies through the requirements.txt like shown in README.md
4. Run python -m unittest test.py

Testing in IDE:

1. Download the code from github in the “james” branch.
2. Open IDE and open project from where it was saved
3. Install all needed libraries (IDE should prompt to do this automatically)
4. Select the test.py
5. Run test.py

**Test Data preparation:**

* Unit Testing will be done through the “james” branch of the github, test.py will not work properly in another branch
* Manual Testing will be done through the “james” and “main” branches.
* Questions.json data must be valid and located in the directory the Trivia module expects it
* brick.wav, wall.wav, paddle.wav, and any other required .wav files in the same directory as the game files.
* Breakout mechanics depend on default positions and attributes defined in the code—ensure these are not modified before testing.

**Test Environment Configuration:**

* Operating System: Windows, macOS, or Linux
* Python Version: 3.9 or later
* Terminal Tools: Command line, powershell, terminal or IDE
* Required Libraries: Pygame, json, math, random, sys, time, unnittest, numpy, pyautogui
* Audio: Ensure audio output is enabled to validate sound-related tests.
* Display: Use a resolution of at least 600x400 for consistent rendering.
* GitHub Actions: Required you have access to the Repository and can go to the Actions tab

**CMSC 495 Project Test Plan.xlsx (Manual Testing):** The link below renders the Excel sheet/ table for manual testing. This is 20 Manual Tests using the Excel method that was proven in the Unit Week Learning Resources.. Specifically, it will demonstrate the Main Menu(5 manual testing), Trivia(5 manual testing), Tic-Tac-Toe(5 manual testing), and Breakout(5 manual testing). This makes a total of 20 Manual Tests for this manual testing.xlsx file.

**Link is below for the CMSC 495 Project Test Plan.xlsx (Manual Testing):**

[CMSC 495 Project Test Plan.xlsx](https://docs.google.com/spreadsheets/d/1MhQiBjRPqWWcELhFAi9Y0fcVixZCgKGo/edit?usp=sharing&ouid=104007192992096171932&rtpof=true&sd=true)

**Steps/Procedures:**

Command line, terminal or powershell steps:

1. Download the code from github in the “james” branch. Refer to README.md for help.
2. Navigate to the project directory
3. Ensure installation of pygame, numpy and pyautogui using “pip install pygame numpy pyautogui” or install dependencies through the requirements.txt like shown in README.md
4. Run main menu with: python MainMenu.py
5. From there test Main Menu and each linked game (Tic Tac Toe, Trivia, Breakout)
6. Follow the steps provided from the CMSC 495 Project Test Plan.xlsx file
   1. For each test Perform the Input/Action
   2. Observe the result
   3. Compare it against the Expected Result
   4. Log pass/fail status
7. Repeat (steps 1-6) but download the code from github in the “main” branch. Refer to README.md for help. This is only for manuals, you cannot do this for unittest.

Testing in IDE:

1. Download the code from github in the “james” branch. Refer to README.md for help.
2. Open IDE and open project from where it was saved
3. Install all needed libraries (IDE should prompt to do this automatically)
4. Select and run MainMenu.py
5. From there test Main Menu and each linked game (Tic Tac Toe, Trivia, Breakout)
6. Follow the steps provided from the CMSC 495 Project Test Plan.xlsx file
   1. For each test Perform the Input/Action
   2. Observe the result
   3. Compare it against the Expected Result
   4. Log pass/fail status
7. Repeat (steps 1-6) but download the code from github in the “main” branch. Refer to README.md for help. This is only for manuals, you cannot do this for unittest.

**Test Data preparation:**

* Unit Testing will be done through the “james” branch of the github, test.py will not work properly in another branch
* Manual Testing will be done through the “james” and “main” branches.
* Questions.json data must be valid and located in the directory the Trivia module expects it
* brick.wav, wall.wav, paddle.wav, and any other required .wav files in the same directory as the game files.
* Breakout mechanics depend on default positions and attributes defined in the code—ensure these are not modified before testing.

**Test Environment Configuration:**

* Operating System: Windows, macOS, or Linux
* Python Version: 3.9 or later
* Terminal Tools: Command line, powershell, terminal or IDE
* Required Libraries: Pygame, json, math, random, sys, time, unnittest, numpy, pyautogui
* Audio: Ensure audio output is enabled to validate sound-related tests.
* Display: Use a resolution of at least 600x400 for consistent rendering.

Note: Refer to APPENDIX D and APPENDIX E for Tests and Git. The Milestones and Gnatt chart updates with the Contribution Report are in APPENDIX A and APPENDIX B and APPENDIX E (issues related to tasks/milestones). APPENDIX C is the agreement for delays.

# **Full Documentation and Code**

**Introduction / Summary (6.1-6.3):**

In this Full Documentation Code Section, the team worked on refactoring the Python Game Hub code to make it cleaner, more organized, and easier to maintain. The team changed the code from large blocks of logic into smaller, reusable components. The team removed repeated code, separated UI and game logic, and added helpful comments and docstrings to explain how everything works. All UI and game logic were decoupled for easier testing and debugging. Game logic in each mini-game—TicTacToe, Breakout, and Trivia—was cleaned up and improved to ensure smooth functionality and scalability. For example, Trivia now has a separate class to manage questions, and Breakout has better collision detection and smoother gameplay.

For **deployment**, the team made the game easy to install and run. They created a clear README file with installation steps, usage instructions, contribution guidelines, packaged dependencies in a requirements.txt, and made sure the game can be run from one main file (MainMenu.py). The game is hosted on GitHub, with the possibility of sharing executables later. Users can download the code or files, install the needed packages, and start playing right away. The team also added documentation, test plans, and diagrams to help others understand and contribute. Specifically, the GitHub repository includes helpful folders like /docs/ (for diagrams and test plans) and .github/workflows/ (for version logs), ensuring all relevant documentation is centralized and accessible.

In the **maintenance phase**, final bug fixes, polish, UI enhancements, and feature upgrades were completed before the official project end date (May 6, 2025). The code now includes detailed inline comments and docstrings (optional) for all major functions and classes. Deprecated code was removed, and consistent naming conventions were enforced throughout. Each mini-game was improved visually and functionally—for example, Trivia got a countdown timer/better scoring/questions in the json file, while TicTacToe’s AI was upgraded. Breakout became more colorful/responsive/music/block features, and the main has smoother transitions. Even though active development is ending, the GitHub repo is well-documented and ready for future updates, feature additions, or community contributions.

**Full Documentation Requirement:** The project satisfies the rubric’s documentation criteria. Every file contains clear inline comments and docstrings to explain logic, input/output, and purpose. A well-organized README.md provides users with setup instructions, gameplay details, and developer guidelines. External documentation files—including UML diagrams, test plans, and version histories—are stored in the /docs/ and .github/workflows/ folders. Together, these resources ensure the program is easy to understand, maintain, and extend—both now and in the future.

**Note: The detailed breakdown of these concepts (6.1-6.3) are below.**

# **6.1. Refactor the Software**

To refactor the Python Game Hub codebase, the focus would be on enhancing modularity, reducing redundancy, and improving clarity across all game files. For instance, in MainMenu.py, repeated code for rendering buttons and handling clicks could be abstracted into reusable helper functions or classes, making the menu system more scalable and easier to manage. We also used the button functions ot include a hyperlink to the github repo as an information page that the user can access. In Breakout.py, logic for collision detection, block generation, and object movement could be separated into dedicated modules or classes with clearly defined responsibilities, improving testability and reducing coupling. Similarly, Trivia.py can be refactored by creating a question handler class that manages loading, validating, and retrieving questions from the JSON file, rather than embedding this logic directly into the game loop. Across all modules, adopting consistent naming conventions, adding docstrings, and separating UI logic from game logic will significantly improve maintainability. In Trivia.py, the current logic for loading and selecting questions from the JSON file is tightly coupled with the game loop. By refactoring this into a dedicated QuestionManager class, the code becomes cleaner and easier to extend—for example, adding new question types or tracking question difficulty would be straightforward. The TicTacToe.py file could benefit from separating game logic (such as win condition checks) from UI rendering, allowing each to be tested and maintained independently. Additionally, across all files, introducing consistent naming conventions, adding inline comments and docstrings, would enhance readability and collaboration. Finally, implementing logging for key game events and errors would improve traceability, especially for debugging and future development. Through these changes, the codebase would not only become more elegant and professional but also far more robust and scalable.

# **6.2. Deploy the Software**

### **Deployment Strategy Overview:**

The deployment of the **Python Game Hub** marks the final stage of the development process. The goal is to make the software easily accessible to users across different platforms while ensuring stability, performance, and ease of use.

**Final Code Documentation and Readability Enhancements:**

* **Inline Comments:** Every major function, class, and logic block in each Python file (MainMenu.py, TicTacToe.py, Trivia.py, Breakout.py, etc.) includes clear, concise comments to explain its purpose and logic.
* **Docstrings(optional):** All functions and classes contain detailed Python docstrings explaining input parameters, return values, and side effects.
* **README.md:** A comprehensive README.md is provided at the root of the repository, containing:
  + Installation instructions
  + Game instructions and controls
  + Testing instructions
  + Contribution and licensing details
  + Details on Units 1-8 and doc locations
* **External Documentation Files:**
  + /.github/workflow/: Contains the version logs of the GitHub Actions Code.
  + /docs/: Contains architecture diagrams (UML, use case, activity), test plans, and version logs.
  + /requirements.txt: Lists all required Python packages for environment setup.
  + /docs/Unit5/CMSC 495 Project Test Plan.xlsx: Provides manual testing steps, inputs, and expected results (copy of it in the docs folder, else it’s stored through the cloud on google drive link)

### **Packaging the Application for Users:**

**Dependencies** are bundled using a requirements.txt file. Users can run:  
 pip install -r requirements.txt

**Single Entry Point:** The user starts the Game Hub by running:  
 python MainMenu.py

OR the alternative methods are in the README.md file.

Link to the README.md: <https://gist.github.com/34c40d49bcef96c70ed35a5a80c33b92.git>

### **Cross-Platform Distribution:**

### **Option A: Source Code Deployment (GitHub)**

* Hosted on GitHub at<https://github.com/javonpayne100/CMSC495Capstone>
* For now it is a private repository unless Unit 8 requires us to make it public, we will.
* Users can clone the repository or download the ZIP and follow the instructions in the README.md file to install dependencies and launch the game.

#### **Option B (Not fully tested to work; Just a hypothesis strategy): Executable Packaging**

Created using pyinstaller to generate a standalone executable file for Windows/macOS/Linux:

pyinstaller --onefile MainMenu.py

* Resulting executable can be distributed as:
  + PythonGameHub.exe (Windows)
  + PythonGameHub.app (macOS)
  + PythonGameHub (Linux)
* All necessary assets (images, sounds, questions.json, etc.) are bundled using the --add-data flag in pyinstaller.
* **Note:** We may not do this version as our team looked into it online and noticed that it gets really complex for Python applications vs C++ or Java ones where it is easily able to make a .exe file without complications.
  + In other words, we may opt when deployment for others to access it directly through the **GitHub repository** once it is published as public.

### **Hosting Options:**

* **Local/Offline Distribution:**
  + After the user downloads all the source files from the repository:
  + The user can share or use it via USB drive, Dropbox, Google Drive, or GitHub release ZIP files.
* **GitHub Releases Page:**
  + Upload compiled executables and update changelogs in the GitHub “Releases” section for each version.
  + This is optional and we may include this if our team wants to otherwise it would just be the repository link.

**Post-Deployment Considerations:**

* **Bug Reporting:** Users are encouraged to report bugs or issues via the GitHub Issues tab.
* **Future Updates:** New games or features will be developed in separate branches and merged via Pull Requests after thorough testing.
* **User Feedback Loop:** Collect feedback through surveys or GitHub discussions for continuous improvement.
* Include a **Release/Package** version on the repository only if our team has time.

With these steps, the Python Game Hub is fully prepared for deployment. With clearly documented code, robust testing, and cross-platform packaging options, users can download, install, and run the game hub with minimal setup. GitHub serves as the central platform for distribution, issue tracking, and future development updates.

# **6.3. Maintain the Software**

**Maintenance Overview:** The maintenance phase of the Python Game Hub focuses on ongoing updates, feature enhancements, and bug fixes after initial development has concluded. Since the project will officially conclude on May 6, 2025, all final maintenance efforts—including refactoring, documentation cleanup, and feature polishing—have been completed during the final phase of this course and our team may decided to turn it in early on May 3, 2025 depending if we catch bugs/issues for resubmission before May 6, 2025.

**Refactoring Summary:** Throughout Phase 2, our team made significant improvements to the codebase to enhance readability, maintainability, and modularity. We added detailed in-line comments across all Python files to clarify the logic and flow of each section. Larger functions were broken down into smaller, reusable components to reduce repetition and improve overall structure. We also refactored game logic in each of the three mini-games to streamline conditionals and input handling. Variables and function names were revised to follow consistent and descriptive naming conventions, improving clarity for future contributors. Deprecated or redundant sections of code leftover from early prototypes in Phase 1 were removed to maintain a clean and efficient codebase.

**Feature Changes: Phase 1 vs Phase 2**

In the **MainMenu.py** module, the original version featured only a basic, functional main menu. In Phase 2, the main menu was transformed into a more engaging and interactive experience. Enhancements included hover-sensitive buttons, a smooth gradient background, and continuous background music to create a polished first impression. A centralized navigate() function was introduced to improve game transitions and menu responsiveness, enhancing the user experience significantly. It should beagle to transition between the 3 games. It also has an information hyperlink (using import webbrowser) to the GitHub Repo Hyperlink of the README.md so that the user can access the information during the game as well through the button. This modification also passed the GitHub Actions tests.

**Trivia.py** in Phase 1 offered limited UI feedback and a basic quiz loop. For Phase 2, the game was redesigned to include randomized question display from a JSON file, improved score tracking, and both 1-player and 2-player modes. A circular donut-style countdown timer was added to visually represent remaining time using trigonometric functions. The answer validation logic was refined to ensure fairness and accuracy, while bugs related to question repetition and incorrect scoring were fixed.

The **TicTacToe.py** file initially included basic two-player gameplay and AI logic. In the updated version, the AI logic using the minimax algorithm was improved for better decision-making. The game’s visual design was enhanced with clear win/draw messages and a custom “Play Again” button that resets the game seamlessly. Improvements were also made to edge case handling, especially in win/draw detection, and the code structure was cleaned up for readability.

**Breakout.py** experienced one of the most significant upgrades. In Phase 1, it had essential paddle and ball mechanics, block collisions, and a simple scoring system. In Phase 2, the game saw the addition of block health systems that vary by color, improved sound effects, and more precise collision detection logic. Notably, the pyautogui.alert() popup used for Game Over messages was removed and replaced with a custom draw\_message() function. This change resolved compatibility issues on macOS systems and ensured smoother cross-platform performance. The updated version is more stable, efficient, and visually satisfying, while remaining scalable for future enhancements.

**Plan for Future Maintenance:** Although the official end of this project is May 6, 2025, the code has been structured in a way that makes future updates and maintenance straightforward. Our team may decide to turn it in early on May 3, 2025 depending if we catch other bugs/issues for resubmission before May 6, 2025. The project repository on GitHub remains active and documented, allowing team members or external contributors to submit pull requests and track issues. The modular codebase ensures that new games or features, such as user profiles or online multiplayer modes, can be easily added. All core libraries and dependencies are listed in a requirements.txt file, and comprehensive setup instructions are provided in the README.md, making it simple for future users or developers to get started. In addition, the use of Python’s unittest module, along with optional GitHub Actions for CI/CD testing, ensures that core features remain stable with every update. These steps make the Python Game Hub a strong foundation for long-term expansion, experimentation, or inclusion in a professional portfolio.

# **7. Source Code (Phase 2)**

**README.md:** The link below renders the code to README.md. This readme file should show how to install, run the application, and install any imports. It will also explain how to play the games and how the unit tests / manual tests are done. It also teaches the user how the unit test / manual tests are processed.

<https://gist.github.com/34c40d49bcef96c70ed35a5a80c33b92.git>

**requirements.txt:** The code below is the requirements.txt dependencies that the user must install through the README.md file. This file should allow the user to import and install dependencies into the PyCharm environment in order to run the application. If it does not work, the user has other options to install them following the directions on the README.md file. If it doesn’t work, the user must include these: random; math; time; json; unittest; sys

**requirements.txt Code:**

pygame

pyautogui

numpy

**MainMenu.py:** The link below renders the code to MainMenu.py. The main menu is the starting and central point of the Game Hub. This is where players can choose out of the three games to play.

<https://gist.github.com/6c7571d895fb7534433a3f99295eddd1.git>

**Trivia.py:** The link below renders the code to Trivia.py. The trivia file is where players can play the trivia game. There are several different features to this game such as 1-player or 2-player mode and a question category selection.

<https://gist.github.com/564baecb625c8720fcf7ed4f02aeb257.git>

**Questions.json:** The link below renders the code to Questions.json. This file is a container for all trivia questions. The trivia.py file fetches all questions from this file. This format makes the code look seamless and also makes it easier to add or delete questions to the game making it dynamic.

<https://gist.github.com/6fc19afdb1c9da9a69329d97b1ab0bcb.git>

**TicTacToe.py:** The link below renders the code to TicTacToe.py. The TicTacToe file is where players can play the Tic-Tac-Toe game. There are several different features to this game such as an AI mode and other graphic features.

<https://gist.github.com/1f6e129b26d35851bd899922765bdb61.git>

**Breakout.py:** The link below renders the code to Breakout.py. The Breakout file is where players can play the Breakout game. There are several different features to this game which includes counting scores, lives, if you hit more than one block it will make it disappear, and other graphic features.

<https://gist.github.com/bf5c8328950daaf7abe96bac46e9c8ad.git>

Note: All audio files(.wav, .mp3) or docs/images (.png,.jpeg, docx) are not included in the code as it is not counted as “code files” and is in the media folder, not included in this submission as they are not code files.

**Other Source Codes for GitHub Actions, Unit Testing, and Manual Testing:**

**GitHub Actions Testing with .yml files (6 of them):** This link below renders the code to the updated **.yml** files to automate the unit tests for our application. It is highly recommended that you run the test.py following the instructions of the IDE, but this method will also confirm it as a double checker to make sure it works through Git’s (CI/CD) pipeline. GitHub Actions Automated testing was tested and created by Victoria to confirm and validate the unittest of James’s test.py files. Victoria will create and implement 3 GitHub Actions for the old test.py file in the main branch and 3 GitHub Actions for the new test.py in the james branch (test.yml, test2.yml, test3.yml, testJames.yml, testJames2.yml, testJames3.yml). The first three without James label should test the old test.py on the main branch with 13 unittest passed. The next three with the James label should test the new test.py on the James branch with 24 unittest passed. This should allow our team to double check with Jame’s manual IDE python runs for the test.py files and make sure it is working. The test.py (old), test.py (new), and CMSC 495 Project Test Plan.xlsx were created and tested by James and the GitHub Actions was made by Victoria while integrating it to the branches and test.py files.

Link to the test.yml file:

<https://gist.github.com/0caa921374cb8b454b3e0519fdbe7267.git>

Link to the test2.yml file:

<https://gist.github.com/828662e48bdb01e9c6dfeaa800fdbc51.git>

Link to the test3.yml file:

<https://gist.github.com/6bc3545fec63e50b31b83bc587992c54.git>

Link to the testJames.yml file:

<https://gist.github.com/83ddfc5588f6dc35678e9f3f61e30662.git>

Link to the testJames2.yml file:

<https://gist.github.com/7ea415e4d4c26de0378e084cb7392390.git>

Link to the testJames3.yml file:

<https://gist.github.com/9ebc800353d91b027d9912409596f271.git>

Link to the old test.py file: <https://gist.github.com/VictoriaRaven/e1c8523412d6f895a7a91fc55c90b8df>

Link to the new test.py file:

<https://gist.github.com/VictoriaRaven/454a8b7026c67efb9e07d562688357fd>

**Git (Version Control) (main) branch only:** This test should be in the .github/workflows of files: **testFiles.yml** and **testFiles2.yml**. The first one tests if the main branch has all the core game files (detects it’s not missing/exists) for Mac and Ubuntu. The second one tests if the main branch has all the game files (detects it’s not missing/exists) for Windows. It uses Node.js and Git Actions commands to check if the files exists.These two tests ensure that our members when merging did not change the file names or delete/remove it from the main. This ensures that there are no core files missing when prepping for pull/push/merge, and deployment for Unit 8.

Link to testFiles.yml file:

<https://gist.github.com/871efc7129b232b85c28be56d0025b5d.git>

Link to testFiles2.yml file:

<https://gist.github.com/d392c23b4b94208f4fd33d800c1ff763.git>

**CMSC 495 Project Test Plan.xlsx (Manual Testing):** The link below renders the Excel sheet/ table for manual testing. This is 20 Manual Tests using the Excel method that was proven in the Unit Week Learning Resources.. Specifically, it will demonstrate the Main Menu(5 manual testing), Trivia(5 manual testing), Tic-Tac-Toe(5 manual testing), and Breakout(5 manual testing). This makes a total of 20 Manual Tests for this manual testing.xlsx file.

**Link is below:**

[CMSC 495 Project Test Plan.xlsx](https://docs.google.com/spreadsheets/d/1MhQiBjRPqWWcELhFAi9Y0fcVixZCgKGo/edit?usp=sharing&ouid=104007192992096171932&rtpof=true&sd=true)

# **8. Conclusion**

Phase II marks a pivotal stage in the development of the Python Game Hub, demonstrating our team’s progress from functional prototypes to a fully tested, deployable software product. Through collaboration, structured planning, and iterative development, we refined each game: Tic Tac Toe, Trivia, and Breakout, into polished modules integrated through a central Main Menu interface.

This phase emphasized code quality, test coverage, documentation, and readiness for deployment. We finalized all core functionalities, applied best practices in refactoring, and ensured thorough testing through both unit and manual test strategies. Git and GitHub Actions were instrumental in managing our codebase and automating tests across environments, contributing to the application’s robustness.

As a team, we learned the importance of consistent communication, modular design, and the value of a well-maintained version control workflow. We gained deeper experience with debugging, user experience refinement, and integrating code from multiple contributors into a unified system. Perhaps most importantly, we learned how to collaboratively plan, implement, and test real software features while adapting to challenges along the way.

Our development environment is now complete, our games are fully playable with user-friendly design, and our codebase is well-documented and maintainable. With the successful implementation of this phase, the project is now ready for the final phase: User Guide documentation and deployment. We are confident that the Python Game Hub meets all project objectives and serves as a strong foundation for future updates, enhancements, and contributions.

# **9. APPENDIX A:**

Notes: This includes Overall tasks, milestones, hours, and cost (for unit 1-8) and progression.

Refer to APPENDIX B for a contribution report for specific Tasks, Milestones, Issues Completed.

Refer to APPENDIX E for Git Tasks, Milestones, Issues Completed.

A Large Table Gantt Chart repeat of Project Plan in Unit 4:

| Week | Dates | Lead | Topic | Description | Due Date | Assignments Due |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 3/10-3/14 | Everyone  And Project Manager | Team Formation  And staring on the Project Plan | Get to know team members & pick a project  -Outline Milestones  -Delegate responsibility  -Describe project's purpose | 3/14 | N/A |
| 2 | 3/15-3/18 | Documentation  And Requirements | Project Plan | -Create a formal Project Plan   * Testing Units   Updating Git | 3/25 | Project Plan |
| 3 | 3/18-3/28 | Documentation  And Developers | Design | -Create application structure  - Develop user interface/functionality  -Create UML diagrams   * Testing Units   Updating Git | 4/1 | Project Design |
| 4 | 3/28-4/4 | Everyone | Phase 1 Source | Software Development   * Testing Units   Updating Git | 4/8 | Phase 1 Source  Peer Review 1 |
| 5 | 4/4-4/11 | Everyone | Testing | * Software Development | 4/15 | Test Plan |
| 6 | 4/11-4/18 | Everyone | Phase 2 Source | * Software Development * Testing Units * Updating Git | 4/27 | Phase 2 Source  (Prepare for Peer Review 2 on week 8) |
| 7 | 4/18-4/25 | Everyone | User Guide and Phase 2 | * Software Development * Testing Units * Updating Git * User Guide documentation and submission | 4/27 | User Guide  and Phase 2 Source (Prepare for Peer Review 2 on week 8) |
| 8 | 4/25-5/2 | Everyone | Final Report | * Compile all Topics into a single document * Testing Units * Updating Git | 5/4 | Final  Peer Review 2 |

Below is a bigger picture version of this:

A screenshot of a computer

AI-generated content may be incorrect.

A white background with black text

AI-generated content may be incorrect.

### **Overall Project Status and Progress Update:**

This report provides an updated analysis of our project’s progress, highlighting the completion status of key project phases, including the project plan, design, testing, and upcoming milestones. Based on the current progress, we estimate the project is **80% complete**, with testing being a key focus in the current week.

We have made substantial progress so far. Here’s an overview of what has been completed, the current phase, and what is left to finish:

#### **Completed Phases (80% complete):**

* **Project Plan**: Fully completed, outlining key tasks, milestones, and deadlines.
* **Project Design**: All design elements and documentation are complete.
* **Phase 1**: The development phase, where all initial features and functionalities were implemented, including core game functionality.
* **Testing:** Fully completed manual and unit testing of the application and the implemented features. The tests are being executed on the project. WE also completed **Trivia.py and Questions.json**, which is part of the game suite for Unit 5.
* **Phase 2 (Currently this week):** The last development phase, where all new features and functionalities were implemented, including core game functionality. And all the tests are implemented. And all the documentation best practices are fulfilled. (Code comments, Readme.md file, documentations, etc.)

#### **Upcoming Tasks (Remaining 20%):**

1. **User Guide Documentation**: The user guide is yet to be completed, which will provide comprehensive instructions for using the software.
2. **Deployment**: The project will be deployed in the final phase. This will include submitting the **User Guide** and the final source code (all implemented game files, tests, etc.). This includes finishing and completing the peer review 2 for Unit 8.

### **Feature Completion**

Here is a breakdown of the current status of the project features and the remaining work needed:

Note: We will not implement a score saver as doing it in real time, updating it through the leaderboard was complicated, as most of the servers that needed to store it online require premium access, which we preferred to be free only (no costs). We also did not have enough time to implement due to the (2 weeks left), and the additional features were more significant than storing real-time-updated data scores on a hosted server. Thus, that scoring idea on a hosted real-time-updated server is removed due to the time constraints for the last two weeks, and would rather be spent on implementing/completing the additional features.

| **Feature** | **Current Status** | **Remaining Tasks** | **Progress** |
| --- | --- | --- | --- |
| **Mainmenu.py** | Completed and fully implemented. | No remaining tasks.  -Updated buttons, UI, updated look, and transitions.  - Updated information button to allow user a hyperlink to Github repository for the README.md page for information/history. | 100% |
| **TicTacToe.py** | Completed and fully implemented. | No remaining tasks.  -Updated AI/CPU/transition, music | 100% |
| **Breakout.py** | Completed and fully implemented. | No remaining tasks.  -Updated scores/lives, including transition, music | 100% |
| **Trivia.py** | Completed and fully implemented. | Finish the implementation for **Unit 5 and 6** and include tests.  -Updates 1-2 player mode and also scores, music, and subject questions.  -Wrap the text around the screen and fix questions.json for symbols issues. | 100% |
| **Unit Testing (all games)** | Comprehensive testing of all games with both manual and unit tests.  Completed and fully implemented | Complete tests for **Trivia.py** and implement all tests (unittest and manual tests based on the Excel sample) | 100% |
| **Phase 2 (all parts from unit 1-5)** | Completed and fully implemented | Complete Phase 2  Completed and fully implemented for Trivia.py Questions.json and other minor issues. | 100% |

### **Current Phase 2 Week and Plan:**

As we are currently in **Phase 2 Week**, the main focus is on ensuring the correctness and reliability of the implemented features, testing, documented, installation, dependencies, and all three systems work:

1. **Phase 2:**
   * During week 6, our team will collaborate to write the document and finish up any other tests with all the Python files to ensure it works.
   * Team will modify minor changes for the UI visual look and complete rerun tests.
   * The team will modify the README.md files, use git to track progress, and update the status. No more code is implemented as we all agreed that we already implemented all features. Our team will fix minor issues on all games for the screen size adjustments, captions, Trivia.py text wrappings, Questions.json symbols issues, and an information hyperlink button on the main menu to the github repo so user can access the readme.md file for information about the game.
2. **Test Completion and Return**:
   * The testing team will finalize the unit tests and manual tests, ensuring that all functions are working as expected before moving to documentation and deployment.

### 

### **Next Phases: Weeks 7 and 8 – User Guide and Finalization/Deployment**

Once the code for Trivia.py is completed and all tests have been confirmed, the team will focus on finalizing the project documentation:

* **User Guide**: During Week 7, our team will work together since there is no more code to be implemented and complete the user guide. This document will cover how to set up and play the games, as well as detailed descriptions of each feature. There is no more code developing or testing this week, but documentation as a whole team.
* **Project Deployment**: During Week 8, once the documentation is complete from Week 7, the final source code, along with the user guide and other documentation, and the peer review 2 will be submitted for deployment. This week, our team will work together to ensure that the documentation and coding files are working and complete each rubric. There is no more code developing or testing this week, but documentation and submitting the final project as a whole team.

### **Project Completion Timeline**

| **Week** | **Task** | **Team Completion Percentage** |
| --- | --- | --- |
| **Week 1** | Create and Form Teams (Brainstorm) | 100% |
| **Week 2** | Project Plan completed (Start Base Code for Python) | 100% |
| **Week 3** | Project Design completed (Continue Developing Mainmenu.py, TicTacToe.py, Breakout.py, Trivia.py) (Brainstorm a README.md and implement Git Version Control and Brainstorm Unit Tests) | 100% |
| **Week 4** | Phase 1 (Implementing features of Mainmenu.py, TicTacToe.py, Breakout.py, Trivia.py; Implement Git Version Control, and start out Unit Tests; implement README.md to prep install and run) | 100% |
| **Week 5** | Testing Week (Finish Unit testing and Manual Testing based on the Excel sample and also finish implemented features (Breakout.py; TicTacToe.py; and MainMenu.py features are already finished, waiting on Trivia.py)) | 100% |
| **Week 6** | Phase 2 (Testing and working on Documentation: Making sure installation works, etc) (Mainmenu.py, TicTacToe.py, Breakout.py, Trivia.py) | 100 % |
| **Week 7** | User Guide Documentation and Start Deployment | N/A % |
| **Week 8** | Deployment: Finish Docs, Code, Finalize Application | N/A % |
| **Overall Progress** |  | **80 % Complete** |

### **Total Progression**

The project is currently **80% complete**, with significant milestones already reached. Most of the core functionality, including the main games and their associated files, has been implemented and is undergoing testing. The remaining tasks involve finalizing and ensuring all **Tests and documentation for Phase 2 (Unit 6)** are completed, and producing the **User Guide (Unit 7)** and **final deployment documentation (Unit 8)**. By working together during weeks 7 and 8, our team will complete the remaining documentation and prepare the project for submission. All code will be reviewed, and the project will be thoroughly tested to ensure that it meets all requirements.

# **10. APPENDIX B:**

CONTRIBUTION REPORT

===============================================================

Project: Python Game Hub

Notes: This includes tasks, milestones, hours, and cost.

Refer to APPENDIX E for Git Tasks, Milestones, Issues Completed.

| TASKS COMPLETED / CONTRIBUTION LOG | | | |  |
| --- | --- | --- | --- | --- |
| DATE | TASKS/MILESTONES | NAME OF VOLUNTEER | HOURS | COST |
| 3/10/2025 to 3/14/2025 | Created at team for project | Lee, Victoria;  Mutry, James;  Foster, Todasha;  Chung, Dajin;  Payne, Javon;  Ipaye, Oluwatumininu | ~5 hour | $0 |
| 3/15-3/17 and 3/18/2025 | Worked and completed Unit 1 and 2 Discussion and Replies | Lee, Victoria;  Mutry, James;  Foster, Todasha;  Chung, Dajin;  Payne, Javon;  Ipaye, Oluwatumininu | ~ 3 hours per person | $0 |
| 3/15-3/17 and 3/18/2025 | Worked on Project Plan (3/11-3/17);  Turned in Project Plan (3/18) | Lee, Victoria;  Mutry, James  Chung, Dajin | ~6-8 hour | $0 |
| 3/18/2025 to 3/26/2025 | Worked on Project Design Plan (3/18-3/26) (delayed by 2-4 days for not rushing and lots of time);  Turned in Project Design Plan (3/27/2025) | Lee, Victoria;  Mutry, James | ~6-8 hour | $0 |
| 3/16/2025 to 3/22/2025 | **Contribution (Dajin Chung):** Created the foundational code for a playable 1v1 Tic Tac Toe, Assisted with Breakout Game background display **Milestones:** Base grid display, turn based logic **Challenges:** Grid refresh bugs on re-click; addressed with a redraw method, Breakout game background flickering **Resolution:** Bug fixed and committed to Git.  Contribution (Todasha Foster): Spent 1 hour coding click sounds for player interactions with Xs and Os | Chung, Dajin  Foster, Todasha | ~3 hours  1 hour |  |
| 3/16/2025 to 3/22/2025 | Worked on the Breakout portion of the game hub. Contribution (Todasha Foster): Spent one hour coding game sound effects for Breakout. Implemented audio for the ball hitting blocks, the ball hitting the paddle, and the player losing the game. Implemented a condition to loop the game back to the main menu.  **Contribution (Dajin Chung):** Implemented player vs AI turn structure, Added basic win condition checks, used pygame.draw methods to render player marks of X and O dynamically **Milestones:** Turn management and player input handling, Drawing X/O marks in correct grid square, board matrix integration for logic evaluation, initial win/draw detection functionality  **Challenges/Issues:**Clicks on board did not register correctly, early bug where could overwrite an existing move **Resolution:** added bound checks to prevent overwrites, tested and verified board accuracy after each turn | Ipaye, Oluwatumininu;  Foster, Todasha.  Chung, Dajin | ~7 hour ~1 hour  ~2 hour | $0 |
| 3/16/2025 to 3/26/2025 | Worked on the Main Menu portion and Contribution (Todasha Foster): Spent 1 hour coding the main menu sound to play when the screen loads and stops once the player navigates away.  Worked on base game, MenuGame Hub, and combine the games into the main through Git pull/push (~ 7 hour) | Foster, Todasha;  Payne, Javon; | ~1 hour  ~ 7 hours | $0 |
| 3/16/2025 to 3/22/2025 | Worked on the Trivia portion. Contribution (Todasha Foster): Expanded the background graphics to enhance the visual appeal of the gameplay environment. Spent 1 hour coding background music to automatically play during gameplay and stop when the game ends. Also, added crowd reaction sound effects, cheering sound to trigger upon a win and an "aww" sound to play after a loss.  Worked on tic tac toe functions/logic 1hr | Foster, Todasha;  Chung, Dajin;  Javon | ~1 hour | $0 |
| 3/20/2025 to 3/26/2025 | Debugged other mini games to make game hub work when combining to the menu (~1 hour) | Payne, Javon;  Lee, Victoria; | ~1 hour | $0 |
| 3/16/2025 to 3/22/2025 | Testing Games work/debugging for help (~1 hour) | Lee, Victoria;  Mutry, James;  Foster, Todasha;  Chung, Dajin;  Payne, Javon;  Ipaye, Oluwatumininu | ~ 1 hour | $0 |
| 3/26/2025 to 3/31/2025 | Worked and completed Unit 3 Discussion and Replies  **Contribution(Dajin Chung):** Assisted with Trivia game logic  **Milestones:** Trivia menu buttons, return to menu logic  **Challenges:** Trivia game ends abruptly; addressed with refactoring method logic  **Resolution:** Bug fixed | Lee, Victoria;  Mutry, James;  Foster, Todasha;  Chung, Dajin;  Payne, Javon;  Ipaye, Oluwatumininu | ~ 3 hours per person | $0 |
| 3/29/2025 to 4/5/2025 | Worked on Phase 1 Source Reporting and submitting it on 4/4/2025 | Lee, Victoria;  Mutry, James; | ~8 hours | $0 |
| 3/29/2025 to 4/5/2025 | Worked on Peer Review Evaluations, completed, and turned it in before or on Tues. | Lee, Victoria;  Mutry, James;  Foster, Todasha;  Chung, Dajin;  Payne, Javon;  Ipaye, Oluwatumininu | ~ 3-4 hours per person | $0 |
| 3/29/2025 to 4/5/2025 | Worked on Unit 4 discussions, due Tuesday | Lee, Victoria;  Mutry, James;  Foster, Todasha;  Chung, Dajin;  Payne, Javon;  Ipaye, Oluwatumininu | ~ 3-4 hours per person | $0 |
| 3/29/2025 to 4/5/2025 | Worked on Additional Features to the Three Games and testing them out  Worked on Making the games transition back to the main menu hub (Not finished for some and will continue this for unit 5). [If the game ends, make it go back to the main menu instead of exiting.] | Payne, Javon;  Ipaye, Oluwatumininu  Foster, Todasha;  Chung, Dajin; | ~ 7 hours | $0 |
| 3/29/2025 to 4/5/2025 | Working on creating and tested out the Unit Test in the test.py file with comments. It has the code for the unit testing. (~ 7 hours)  Tested out the Unit Testing. (~1 hour) | Mutry, James;  Lee, Victoria; | ~ 7 hours  ~ 1 hour | $0 |
| 3/29/2025 to 4/5/2025 | Worked on creating and testing out the README.md file based on the requirements and meeting the professor's feedback (email).  [Explains how to set up; how to run; how to play; how to do the unit test; project’s progress; git;]  Worked on creating the media folder and adjusting the other files to move the sounds/music. | Lee, Victoria; | ~ 7 hours | $0 |
| 3/29/2025 to 4/5/2025 | Worked on adding the GitHub Code Links to the document with a short description. Worked on fixing the Git Version control on the documentation (~2 hours)  Tested Links, updated it, and checked documentation updates (~ 1 hour) | Payne, Javon;  Lee, Victoria; | ~ 3 hours  ~ 1 hour | $0 |
| 4/5/2025 to 4/11/2025 | Worked on Test Plan and submitting it on 4/12/2025 | Lee, Victoria;  Mutry, James; | ~8 hours | $0 |
| 4/5/2025 to 4/11/2025 | Worked on the Unit Tests on new test.py, making changes to the files on james branch, testing the new unit tests, creating manual tests, and updating the excel sheet.  Write on parts 5, 6, 9 of the Test Plan | Mutry, James; | ~ 16 hours | $0 |
| 4/5/2025 to 4/11/2025 | Worked on the GitHub Actions for all three systems (Mac, Win, Linux) with the old test.py and new test.py  Worked on writing / completing all Test Plan sections, except 5, 6, 9.  Worked on updating the README.md file | Lee, Victoria; | ~ 16 hours | $0 |
|  | Assisted Victoria to test out the GitHub Actions on their ends even though Victoria tested it out on her side and it works. This is for all 6 yml files (old test.py and new test.py) | Ipaye, Oluwatumininu  Mutry, James; | ~ 1 hour | $0 |
| 4/5/2025 to 4/11/2025 | Worked on Unit 5 discussions, due Tuesday | Lee, Victoria;  Mutry, James;  Foster, Todasha;  Chung, Dajin;  Payne, Javon;  Ipaye, Oluwatumininu | ~ 3-4 hours per person | $0 |
| 4/5/2025 to 4/11/2025 | Worked on completing Trivia.py, and Questions.json and including testing the features. All games already completed 100%.  Merged Questions.json into main through upload. | Payne, Javon; | ~8 hours | $0 |
| 4/5/2025 to 4/11/2025 | Worked on helping expand the Tasks, Milestones, and Job Contribution on the Test Plan  Worked on helping Javon and Jame’s fix Mac issues with breakout.py | Mutry, James;  Foster, Todasha;  Chung, Dajin;  Payne, Javon;  Ipaye, Oluwatumininu | ~ 8 hours | $0 |
| 4/11/2025 to 4/18/2025 | Worked on Phase 2 Source Code Report for Unit 6 and submitting it on 4/19/2025 | Lee, Victoria;  Mutry, James;  Foster, Todasha;  Chung, Dajin;  Payne, Javon;  Ipaye, Oluwatumininu | ~8 hours per person | $0 |
| 4/11/2025 to 4/18/2025 | Worked on Unit 6 discussions, due Tuesday | Lee, Victoria;  Mutry, James;  Foster, Todasha;  Chung, Dajin;  Payne, Javon;  Ipaye, Oluwatumininu | ~ 3-4 hours per person | $0 |
| 4/11/2025 to 4/18/2025 | Worked on debugging the MAC os issue on Breakout.py  Javon and James helped detect the issue which was the pyautogui.alert issue before breakout.py starts.  Todasha helped rewrite the code for MAC | Mutry, James;  Payne, Javon;    Foster, Todasha; | ~8 hours per person | $0 |
| 4/11/2025 to 4/18/2025 | Worked on fixing (changing minor code for) the text wrap issue for Triva.py and the math symbols not showing up through the Questions.json file. (UTF-8 or some symbols unsupported through pygame.)  This peron also tested it out so it works. | Payne, Javon; | ~8 hours | $0 |
| 4/11/2025 to 4/18/2025 | Worked on fixing README.md file and making sure it is easy to intall and run. Makes sure it breaks down both old and new test.py unittests and manual tests onto the readme.md file.  Created the Information hyperlink button on the MainMenu.py and fixed screen sizes and captions on the other .py games. | Lee, Victoria | ~16 hours | $0 |
| 4/18/2025 to 4/25/2025 | Worked on User Guide for Unit 7 and submitting it on 4/26/2025 | Lee, Victoria;  Mutry, James;  Foster, Todasha;  Chung, Dajin;  Payne, Javon;  Ipaye, Oluwatumininu | ~16 hours per person | $0 |
| 4/18/2025 to 4/25/2025 | Worked on Unit 6 discussions, due Tuesday | Lee, Victoria;  Mutry, James;  Foster, Todasha;  Chung, Dajin;  Payne, Javon;  Ipaye, Oluwatumininu | ~ 3-4 hours per person | $0 |
| 4/25/2025 to 5/2/2025 | Worked on Deployment for Unit 8 and submitting it on 5/3/2025 | Lee, Victoria;  Mutry, James;  Foster, Todasha;  Chung, Dajin;  Payne, Javon;  Ipaye, Oluwatumininu | ~16 hours per person | $0 |
| 4/25/2025 to 5/2/2025 | Worked on Unit 8 discussions, due Tuesday  Worked on Peer Review 2, due Tuesday | Lee, Victoria;  Mutry, James;  Foster, Todasha;  Chung, Dajin;  Payne, Javon;  Ipaye, Oluwatumininu | ~ 6 hours per person | $0 |

**Expansion on the Contribution Report for the tasks, milestones, not completed, completed, hours, cost, etc:**

**Refer to Git for more proof of milestones, tasks, and issues on APPENDIX E.**

3/10/25 – 3/14/25: Team created for final project. Project ideas shared and discussed, decisions made for language to be used and meeting times, and tasks delegated to each member. Time spent totaled roughly five hours, costing zero dollars.

3/15/25 – 3/18/25: Unit 1 and 2 discussions completed. Replies made to other classmates to collaborate and encourage success on their respective projects. Time spent totaled roughly three hours per member, costing zero dollars.

3/15/25 – 3/18/25: Project Plan created and edited to meet design rubrics. The Project Plan gives an overview of the chosen project, and goes into detail on the goals, objectives, and deliverables. Time spent totaled roughly six to eight hours, costing zero dollars.

3/18/25 – 3/26/25: Project Design Plan created and edited to meet design rubrics. The Project Design Plan is similar to the Project Plan and also includes risks associated with the project, as well as an evaluation plan and the schedule showing the timeline and/or milestones. Time spent totaled roughly six to eight hours, costing zero dollars.

3/16/25 – 3/22/25: Dajin created base code to be used for the Tic-Tac-Toe game. Base code to be expanded on with functions and refinements. Time spent totaled roughly two hours, costing zero dollars.

3/16/25 – 3/22/25: Oluwatumininu created the base code for Breakout and added functionality to the code. Todasha created and implemented sound effects for various actions to be performed when running the code. Time spent totaled roughly eight hours, costing zero dollars.

3/16/25 – 3/22/25: Dajin worked further on the Tic-Tac-Toe code functionality and logic. Javon created base code for the Trivia game. Todasha created and implemented sound effects for actions performed when running the code. Time spent totaled roughly one hour, costing zero dollars.

3/16/25 – 3/26/25: Javon created base code for the main menu hub that ties in all three games to be created. Games were combined into the main menu through Git pull/push. Todasha created and implemented sound effects for various actions performed when running the code. Time spent totaled roughly eight hours, costing zero dollars.

3/20/25 – 3/26/25: Victoria and Javon worked on debugging the mini games to ensure proper functionality when they are combined with the main menu. Time spent totaled roughly one hour, costing zero dollars.

3/16/25 – 3/22/25: The entire team worked on testing out the games and their features, as well as debugging issues. Time spent totaled roughly one hour, costing zero dollars.

3/26/25 -3/31/25: Unit 3 discussion and replies to other classmates completed. Time spent totaled roughly three hours per person, costing zero dollars.

3/29/25 – 4/5/25: Victoria and James worked on the Phase 1 Source Code and edited the file to meet the design rubrics. The Source Code entails the project’s development environment, functionalities and algorithms used, unit tests to verify correctness of the code modules, and documentation to enhance readability. Time spent totaled roughly eight hours, costing zero dollars.

3/29/25 – 4/5/25: Peer Review Evaluation forms filled out and turned in by due date. Time spent totaled roughly three to four hours per person, costing zero dollars.

3/29/25 – 4/5/25: Unit 4 discussion and replies to other classmates completed. Time spent totaled roughly three to four hours per person, costing zero dollars.

3/29/25 – 4/5/25: Additional features (if feasible) are created for the mini games and tested out. Function to return back to the main menu from any game added to enhance user experience. Time spent totaled roughly seven hours, costing zero dollars.

3/29-25 – 4/5/25: Victoria worked on the README.md file to meet design rubrics and feedback received from the professor. The README.md file is crucial as it documents the contents and structure of our project in the scenario that certain information needs to be located by a researcher. Media folder was also created to easily locate files for sounds or music for in-game actions. Time spent totaled roughly seven hours, costing zero dollars.

3/29/25 – 4/5/25: Javon and Victoria worked on Git version control to be added to the Phase 1 Source Code documentation. GitHub code links were also added to the document with short descriptions for easier access to the repository if needed. Links were tested and updated to ensure correctness. Documentation updates checked out for accuracy. Time spent totaled roughly four hours, costing zero dollars.

4/5/2025 – 4/11/2025: Victoria Lee worked on the Test Plan and submitted it on 4/11/2025.

4/5/2025 – 4/11/2025: James Mutry worked on the Unit Tests in the new test.py, made changes to files on James' branch, tested new unit tests, created manual tests, updated the Excel sheet, and wrote parts 5, 6, and 9 of the Test Plan (~8 hours, $0).

4/5/2025 – 4/11/2025: James Mutry worked on GitHub Actions for Mac, Windows, and Linux using both the old and new test.py, completed all sections of the Test Plan except parts 5, 6, and 9, and updated the README.md (~16 hours, $0).

4/5/2025 – 4/11/2025: Victoria Lee assisted in testing GitHub Actions across all six .yml files, even though her local tests passed (~16 hours, $0).

4/5/2025 – 4/11/2025: Oluwatumininu Ipaye and James Mutry assisted Victoria with testing GitHub Actions across all systems (~1 hour, $0).

4/5/2025 – 4/11/2025: Victoria Lee, James Mutry, Todasha Foster, Dajin Chung, Javon Payne, and Oluwatumininu Ipaye worked on Unit 5 discussions, due Tuesday (~3–4 hours per person, $0).

4/5/2025 – 4/11/2025: Javon Payne completed work on Trivia.py and Questions.json, tested all features, and confirmed all games were completed 100%; he also merged Questions.json into the main branch (~8 hours, $0).

4/5/2025 – 4/11/2025: James Mutry, Todasha Foster, Dajin Chung, Javon Payne, and Oluwatumininu Ipaye contributed to expanding the Tasks, Milestones, and Job Contributions in the Test Plan, and helped resolve Mac issues with breakout.py (~8 hours, $0).

4/11/2025 – 4/18/2025: Victoria Lee, James Mutry, Todasha Foster, Dajin Chung, Javon Payne, and Oluwatumininu Ipaye Worked on Phase 2 Source Code Report for Unit 6 and submitted it on 4/19/2025. (~8 hours per person, $0)

4/11/2025 – 4/18/2025: Victoria Lee, James Mutry, Todasha Foster, Dajin Chung, Javon Payne, and Oluwatumininu Ipaye Worked on Unit 6 discussions, due Tuesday. (~3–4 hours per person, $0)

4/11/2025 – 4/18/2025 James Mutry, Javon Payne, and Todasha Foster Worked on debugging the Mac OS issue on breakout.py. Javon and James detected the issue, which was related to pyautogui.alert before breakout.py starts. Todasha helped rewrite the code for Mac compatibility. (~8 hours per person, $0)

4/11/2025 – 4/18/2025: Javon Payne Worked on fixing the text wrap issue in Trivia.py and resolving math symbols not displaying correctly from Questions.json (UTF-8 or Pygame symbol support issue). The fix was tested successfully. (~8 hours, $0)

4/11/2025 – 4/18/2025: Victoria Lee Updated README.md to ensure it is clear and easy to follow. Added breakdowns for both old and new test.py unit tests and manual test instructions. Created the Information hyperlink button on the MainMenu.py and fixed screen sizes and captions on the other .py games. (~16 hours, $0)

4/18/2025 – 4/25/2025: Victoria Lee, James Mutry, Todasha Foster, Dajin Chung, Javon Payne, and Oluwatumininu Ipaye Worked on User Guide for Unit 7 and submitted it on 4/26/2025. (~16 hours per person, $0)

4/18/2025 – 4/25/2025: Victoria Lee, James Mutry, Todasha Foster, Dajin Chung, Javon Payne, and Oluwatumininu Ipaye Worked on Unit 7 discussions, due Tuesday. (~3–4 hours per person, $0)

4/25/2025 – 5/2/2025: Victoria Lee, James Mutry, Todasha Foster, Dajin Chung, Javon Payne, Oluwatumininu Ipaye Worked on Deployment for Unit 8 and submitted it on 5/3/2025. (~16 hours per person, $0)

4/25/2025 – 5/2/2025: Victoria Lee, James Mutry, Todasha Foster, Dajin Chung, Javon Payne, and Oluwatumininu Ipaye Worked on Unit 8 discussions (due Tuesday) and Peer Review 2 (due Tuesday). (~6 hours per person, $0)

**Signed: Lee, Victoria; Mutry, James; Foster, Todasha; Chung, Dajin; Payne, Javon; Ipaye, Oluwatumininu**

**Date: 3/10/2025 - 5/3/2025**

# **11. APPENDIX C:**

DELAY AGREEMENT / EMERGENCIES (PUSH BACK MILESTONES):

===============================================================

Project: Python Game Hub

Agreement:

All team members agree that if any project or weekly milestones/tasks are not completed as scheduled, they will be pushed back by a period of 1-2 days. This delay will apply unless otherwise agreed upon by the team, with consideration for any unforeseen circumstances. During this period of delay, all members can also swap roles to complete the tasks within the delay time period. After the extra period, the tasks must be completed with no excuses.

Emergencies and Exceptions:

In cases of emergencies or other excusable events, the delay in completion of milestones/tasks may be longer. Each team member is expected to communicate any such issues in advance to ensure that an appropriate delay period is decided upon. Examples of acceptable delays may include, but are not limited to: personal emergencies, technical difficulties, illness, or unexpected external factors that impede progress.

Weekly Meetings:

To mitigate delays and issues, weekly meetings will be held to address any ongoing challenges, ensure clear communication, and provide an opportunity for team members to discuss progress and potential risks. These meetings will serve as a platform for verbal communication, enabling the team to stay aligned and address any emergent issues quickly, preventing further delays. Discord and Git Control will also help mitigate these issues as well.

Acknowledgment:

As of **4/18/2025**, our team has not made any delays and completed each unit within a week.

The following team members acknowledge and agree to the delay terms as outlined above:

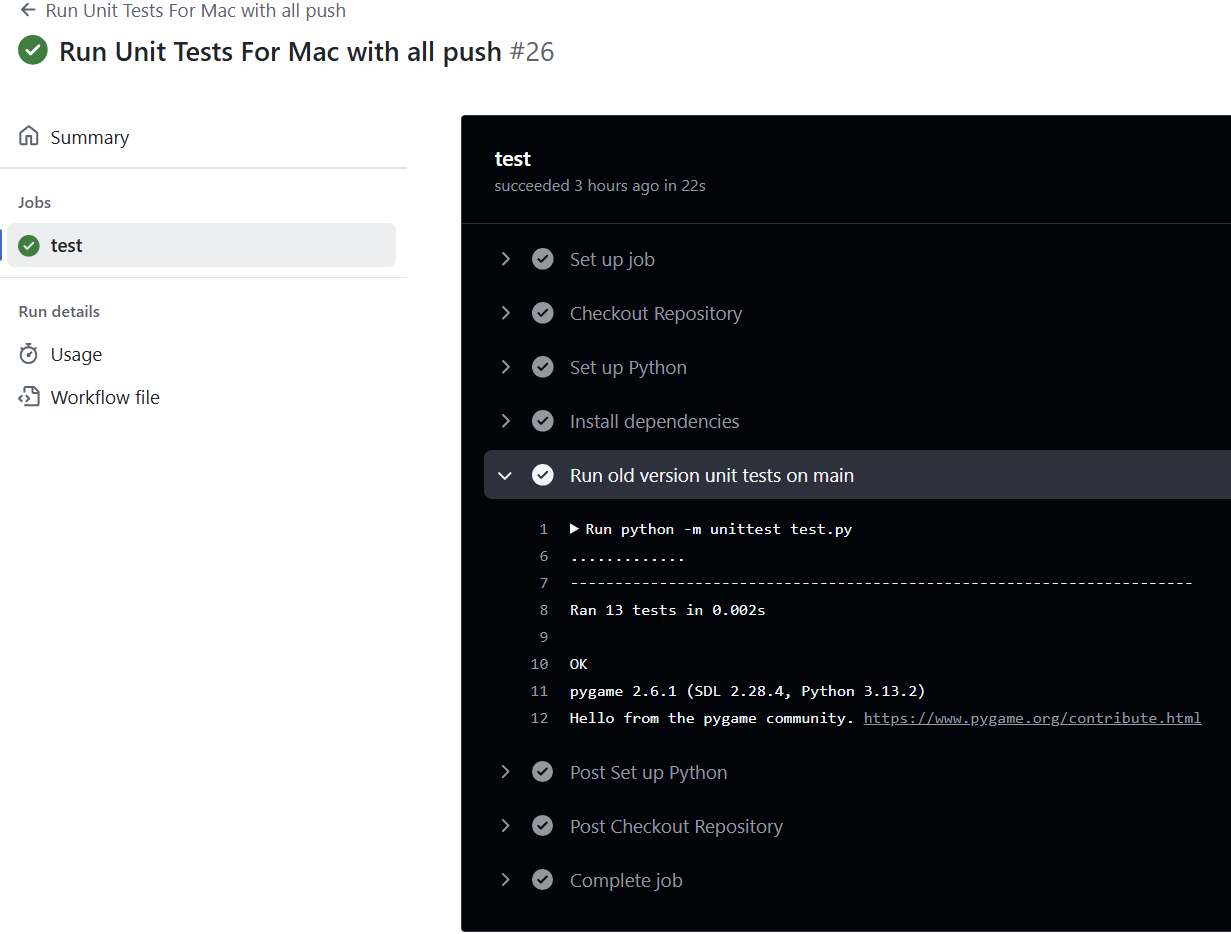
**Signed:  
Lee, Victoria; Mutry, James; Foster, Todasha; Chung, Dajin; Payne, Javon; Ipaye, Oluwatumininu**

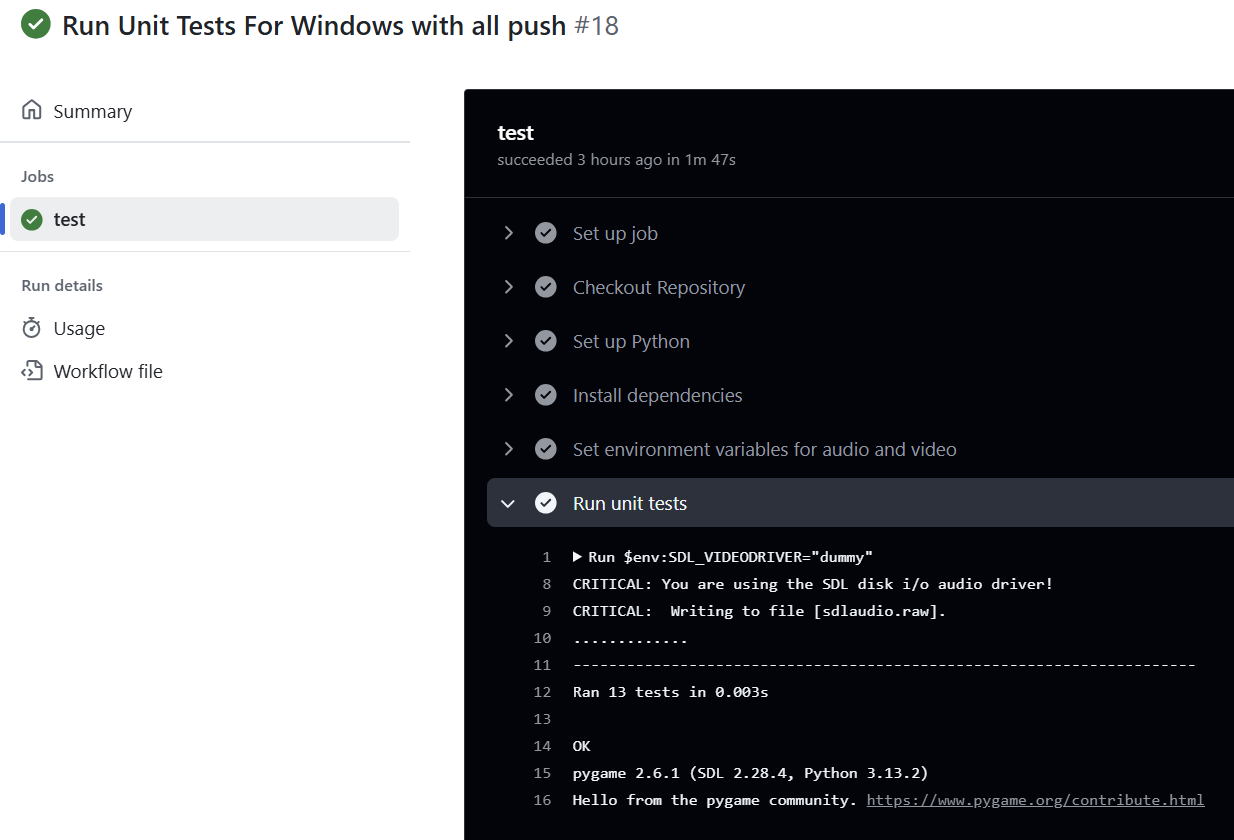
**Date: 3/18/2025 - 5/3/2025**

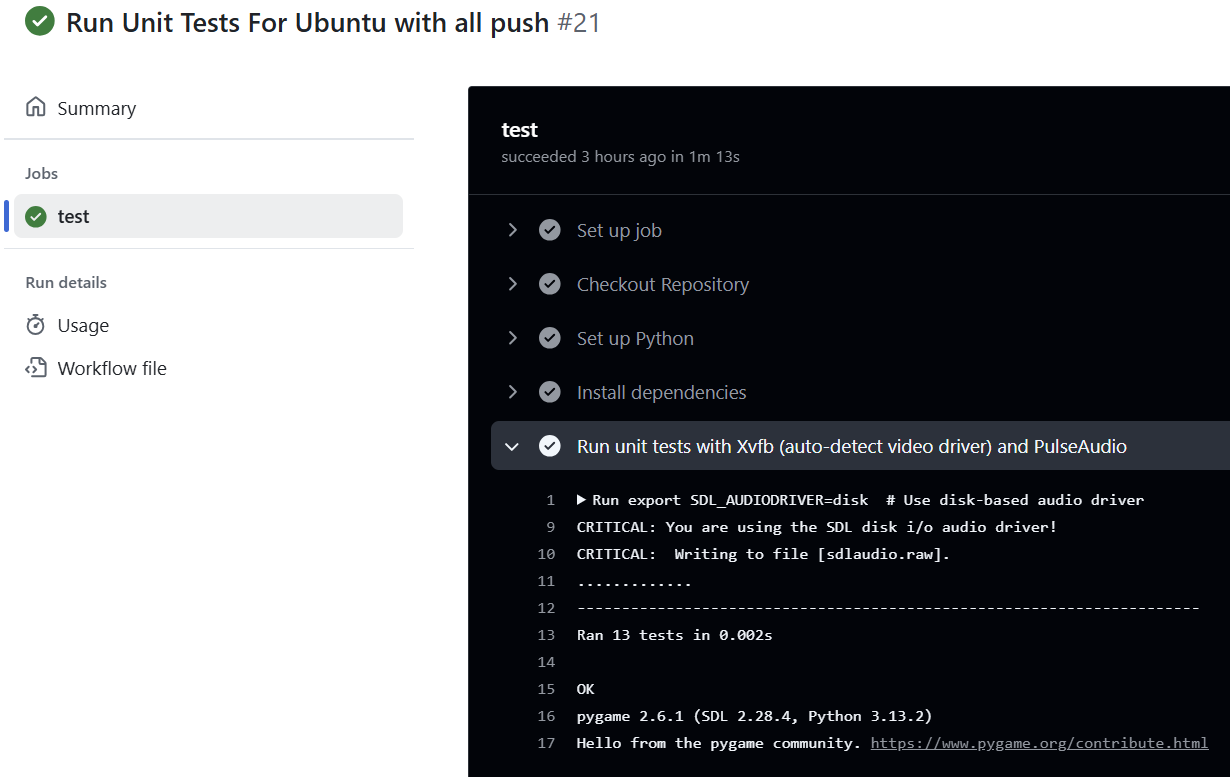
# **10. APPENDIX D:**

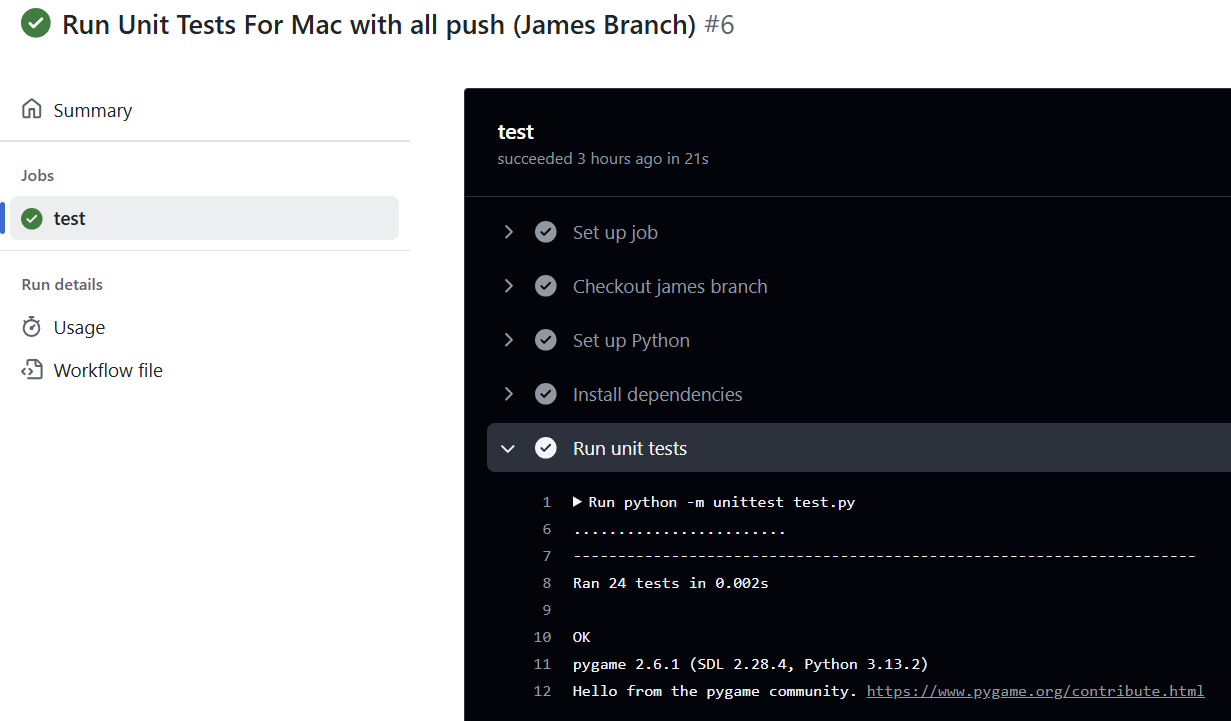
**Continued Screenshots for GitHub Actions working for all 6 unit tests:**

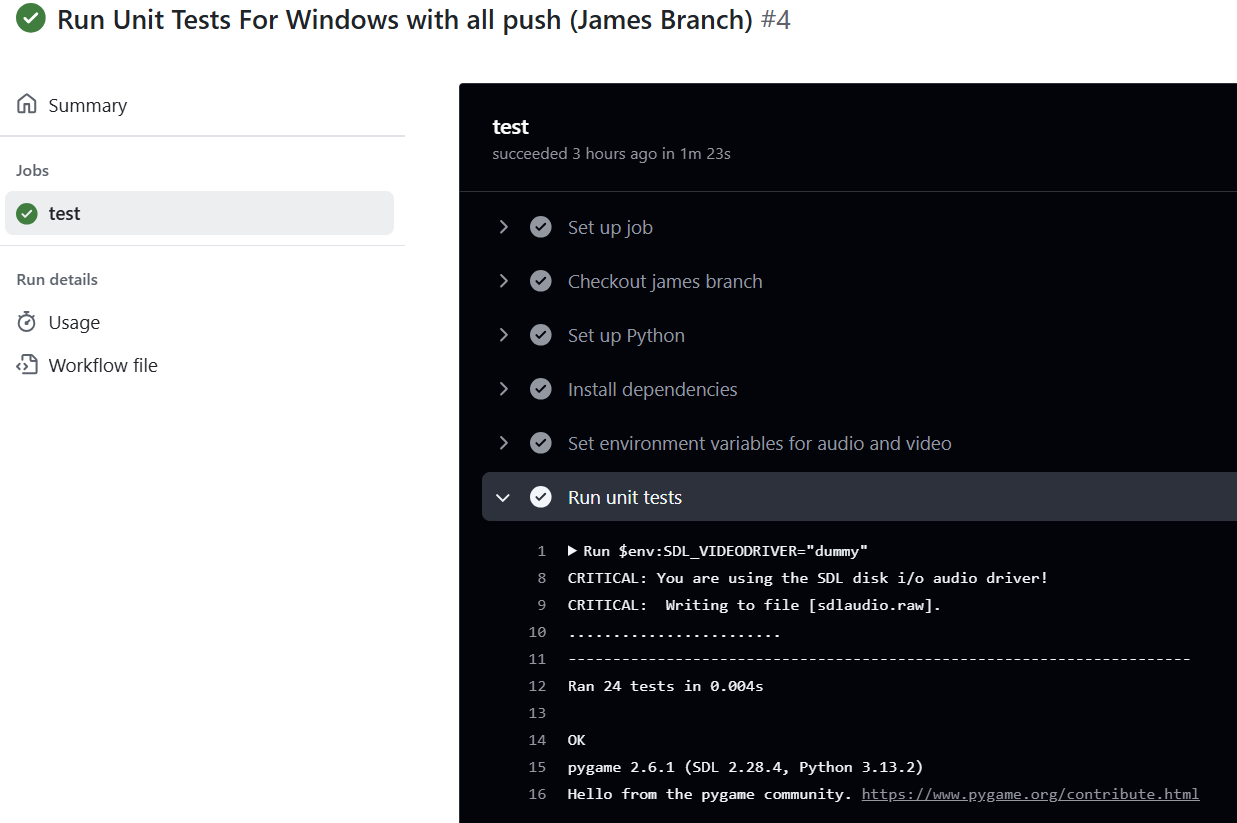
**(I placed it here as the document was becoming long if I didn’t put it in the APPENDIX)**

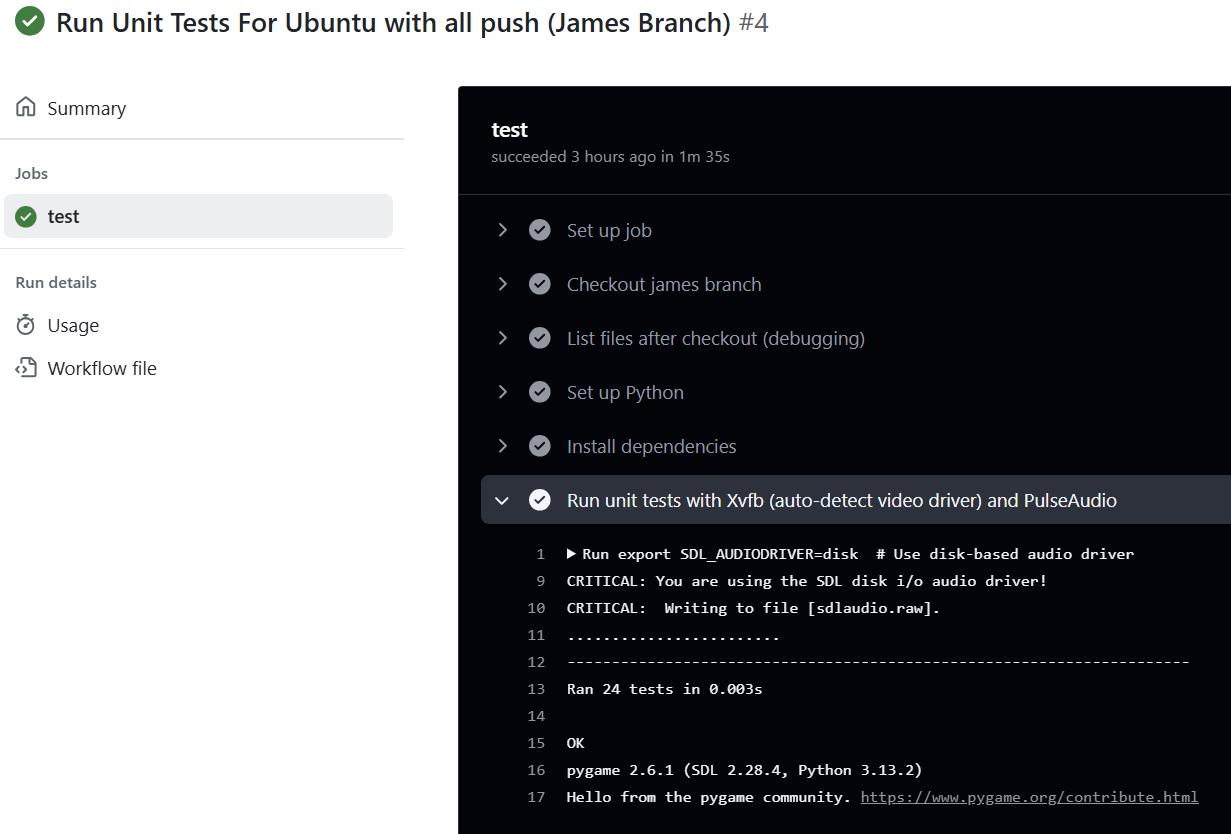
****

****

****

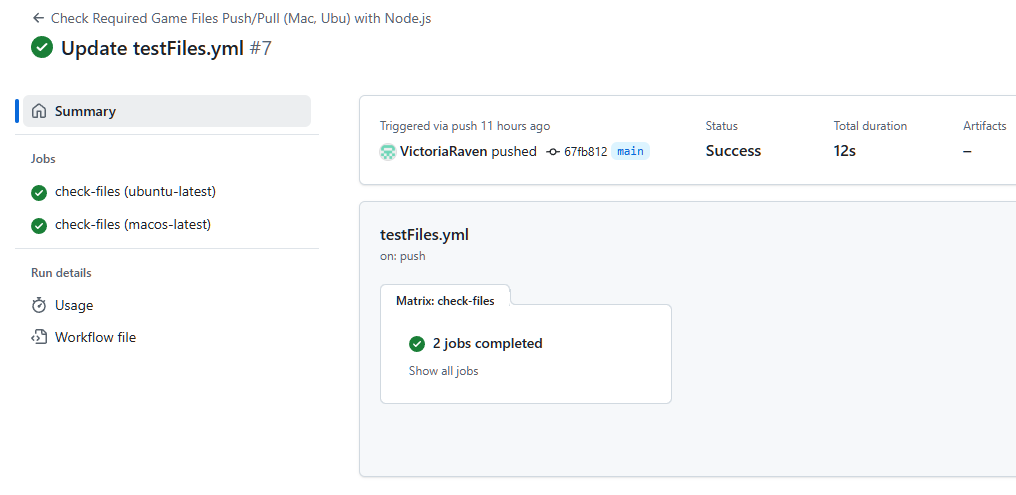
****

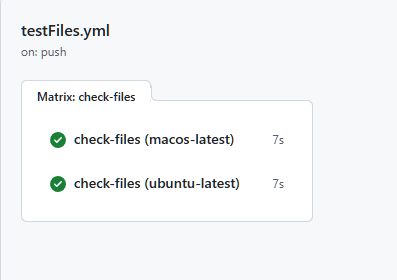
****

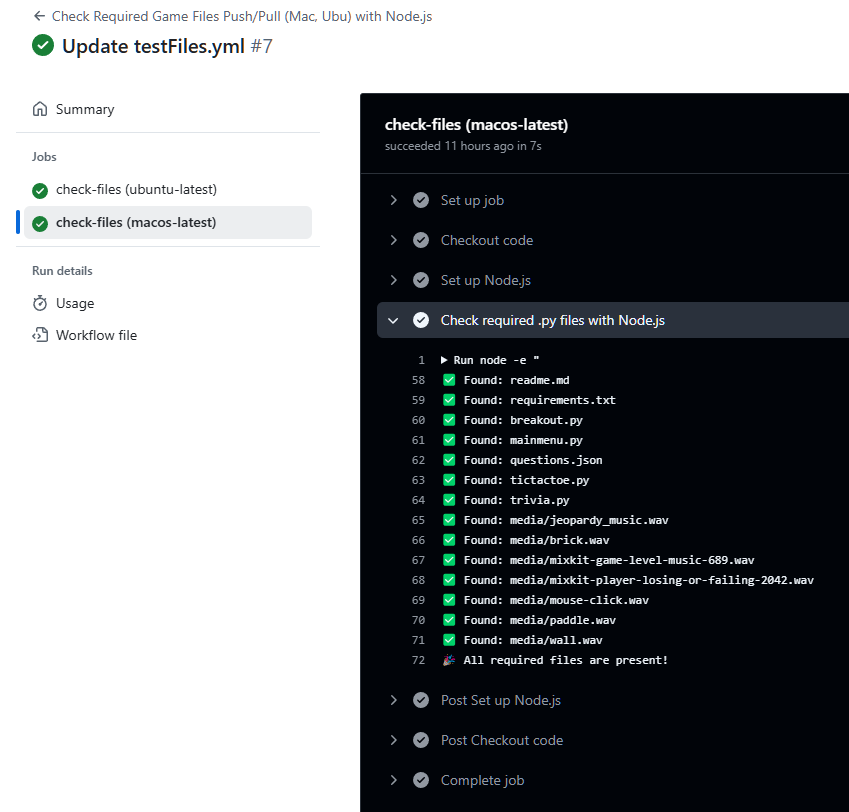
****

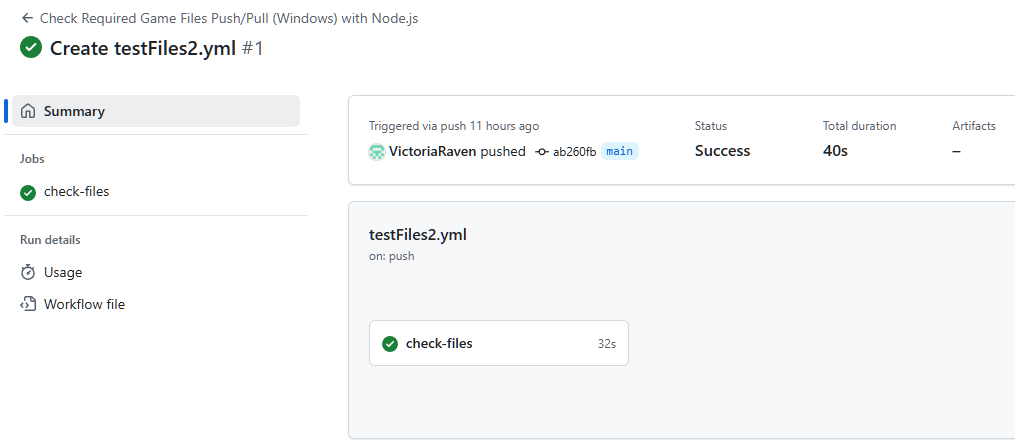
**Continued Screenshots for GitHub Actions working for all 2 .github/workflows testFiles:**

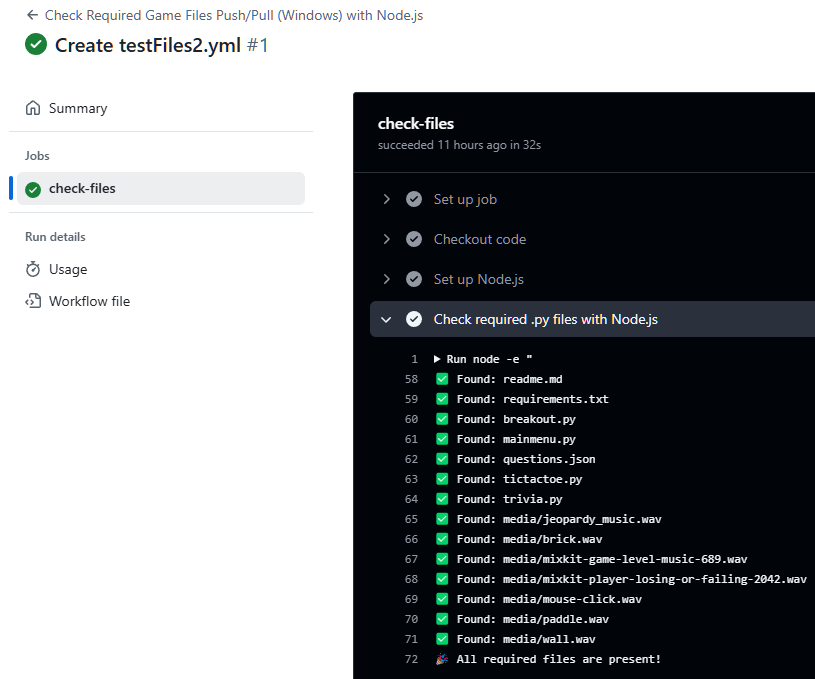
**(I placed it here as the document was becoming long if I didn’t put it in the APPENDIX)**

****

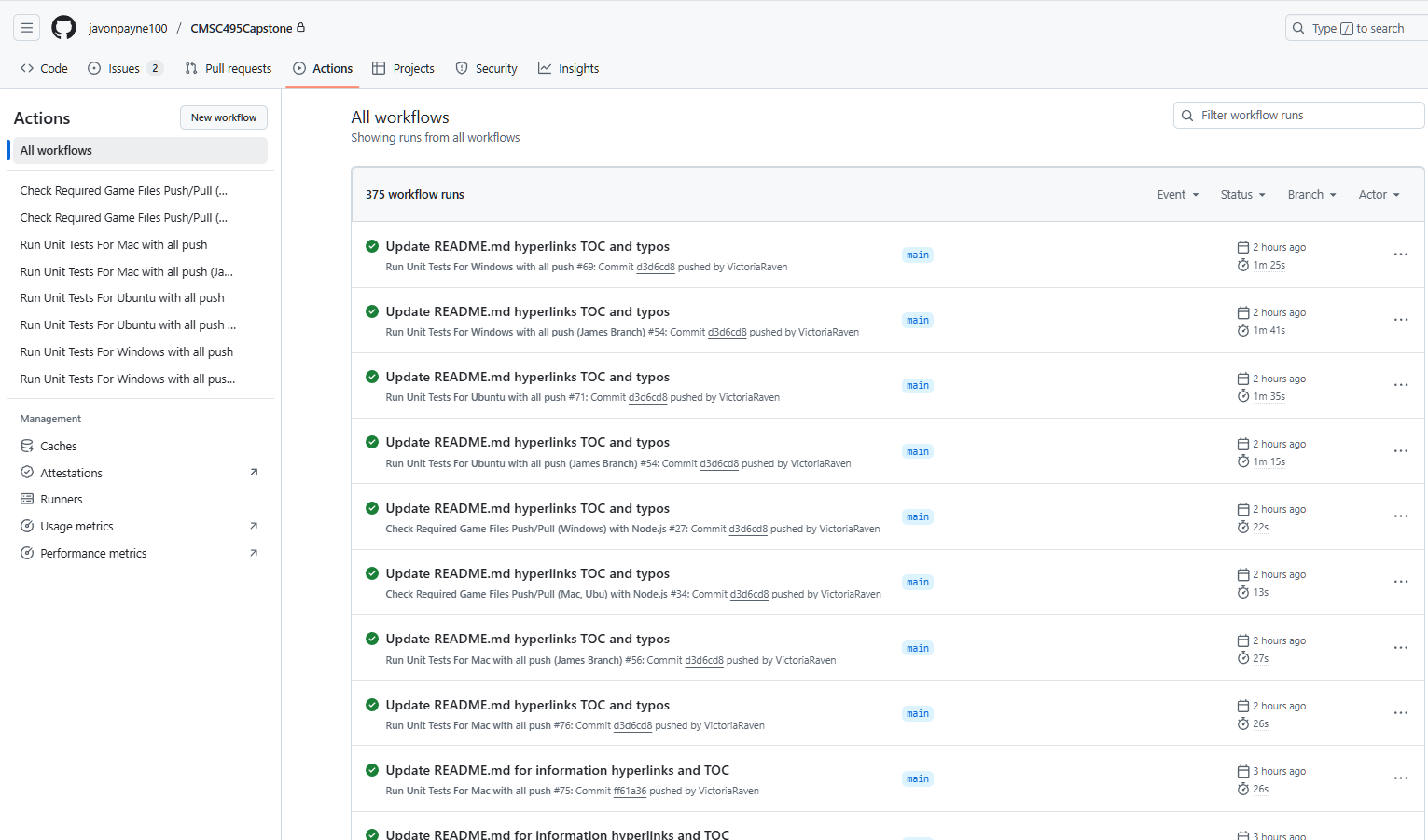
****

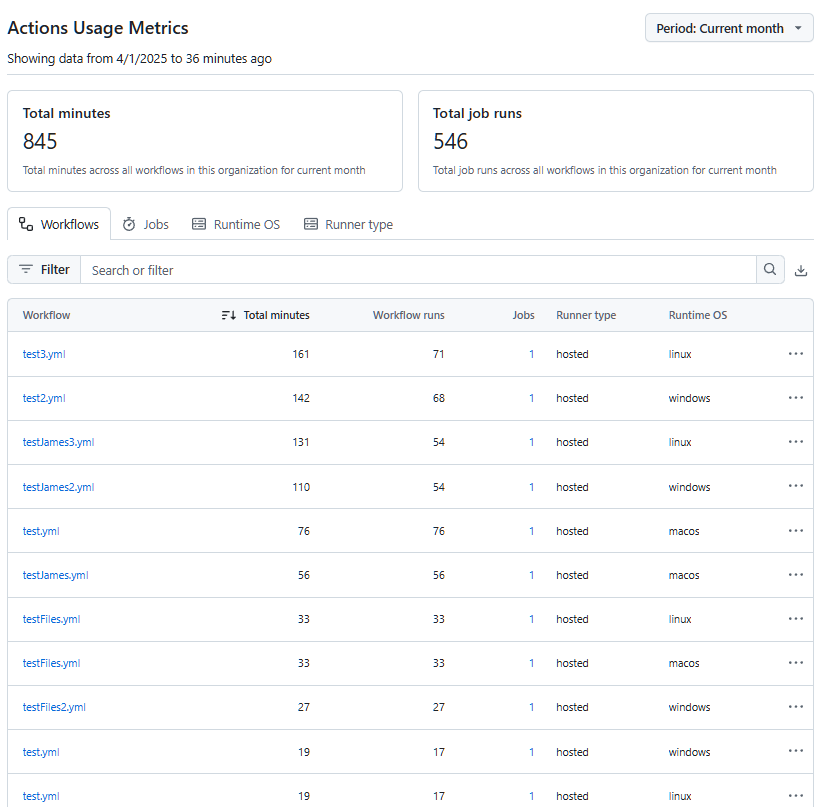
****

****

****

**Our GitHub Actions Workflow Overall (4/18/2025) Screenshots:**

****

****

**Unit 4’s Unit Testing Process Screenshots and basis for Unit 5 and Unit 6:**

This covers all parts of the Unit Testing, Debugging, and the relevant Software Engineering Techniques for each Break-Out, Tic-Tac-Toe, and Trivia. This table outlines the steps, questions to ask, and techniques that align with the software engineering process. I am placing this here so that you know the transition from Unit 4 to Unit 5 to Unit 6. This is based on GeeksforGeeks (n.d.) and Tsui, F., Karam, O., & Bernal, B. (2014). Below is the Chart diagram of this (Simplified):

A close-up of a grid

AI-generated content may be incorrect.

| **Category** | **Game/Menu** | **Meets Requirements** |
| --- | --- | --- |
| **Unit Testing/ Manual Testing:** | Break-Out | Yes; Completed; Closed |
|  | Tic-Tac-Toe | Yes; Completed; Closed |
| **Unit Testing/Manual Testing (Cont.)** | Trivia  Main Menu | Yes; Completed; Closed  Yes; Completed; Closed |
| **Unit Testing** | Break-Out | Yes; Completed; Closed |
|  | Tic-Tac-Toe | Yes; Completed; Closed |
|  | Trivia  Main Menu | Yes; Completed; Closed  Yes; Completed; Closed |
| **Manual Testing** | Break-Out | Yes; Completed; Closed |
|  | Tic-Tac-Toe | Yes; Completed; Closed |
|  | Trivia  Main Menu | Yes; Completed; Closed  Yes; Completed; Closed |
| **Debugging Process** | Break-Out | Yes; Completed; Closed |
|  | Tic-Tac-Toe | Yes; Completed; Closed |
|  | Trivia  Main Menu | Yes; Completed; Closed  Yes; Completed; Closed |
| **Performance Testing** | Break-Out | Yes; Completed; Closed |
|  | Tic-Tac-Toe | Yes; Completed; Closed |
|  | Trivia  Main Menu | Yes; Completed; Closed  Yes; Completed; Closed |
| **Cross-Platform Testing** | Break-Out | Yes; Completed; Closed |
| (Windows, Mac, Linux, IDE, etc) | Tic-Tac-Toe | Yes; Completed; Closed |
|  | Trivia  Main Menu | Yes; Completed; Closed  Yes; Completed; Closed |
| **Additional Features & Fixes** | Break-Out | Yes; Completed; Closed |
|  | Tic-Tac-Toe | Yes; Completed; Closed |
|  | Trivia | Yes; Completed; Closed |
|  | Main Menu | Yes; Completed; Closed |
| **Fixing 1 Issue in Unit 6** | Trivia and Questions | Yes; Completed; Closed |

**Note: Refer to APPENDIX E for the Git Version Control Process**

# **12. APPENDIX E:**

**Git (GitHub Version Control):**

**Testing/Debugging Requirements:**

* To track progress on each coding section, we will use Git for Version Control.
  + Each process will include a Git commit section
  + Each readme file will ensure that the user knows how to set the requirements of the software and run the application to play the games.
  + Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process

**Version Control with Git:**

* **Git as Version Control:**
  + Git will be used to track the progress of the project, manage code changes, and ensure collaboration between developers. Every change made to the codebase will be tracked with commit messages, providing a clear history of the project's development.
  + Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process
* **Branching and Commit Process:**
  + **Create a Separate Branch for Each Feature or Update:**
    - Each developer will create a separate feature branch to work on a specific task or update. This ensures that changes are isolated and don’t interfere with the main codebase (usually the main or develop branch).
    - Example: A developer working on the Tic-Tac-Toe game logic would create a branch named feature/tic-tac-toe-game-logic.
  + **Make Changes and Commit (Push/Pull/Directly through Main):**
    - Developers will work on their assigned feature in their own branches, making incremental changes as needed. For each change, they will commit with clear, descriptive commit messages explaining what was added or modified in the code.
    - After pushing the changes, the developer will create a Pull Request (PR) in GitHub to merge their feature branch into the main branch.
    - Or the developer can just upload it or change it directly through the main after asking permission form the other team members.
    - This ensures that the main branch always contains the most up-to-date, stable version of the code.
* **Tracking Code Changes and Progress:**
  + Each commit and PR will serve as a record of progress in the project. This enables easy tracking of features and bug fixes over time.
  + Developers can refer back to specific commits and PRs to understand when and why certain changes were made, ensuring transparency and accountability throughout the development process.
* **Collaboration and Coordination:**
  + Developers will regularly pull the latest changes from the main branch to keep their feature branches up to date and avoid conflicts.
  + If any merge conflicts arise, the developer will resolve them before pushing their changes, ensuring that the codebase remains functional and consistent.

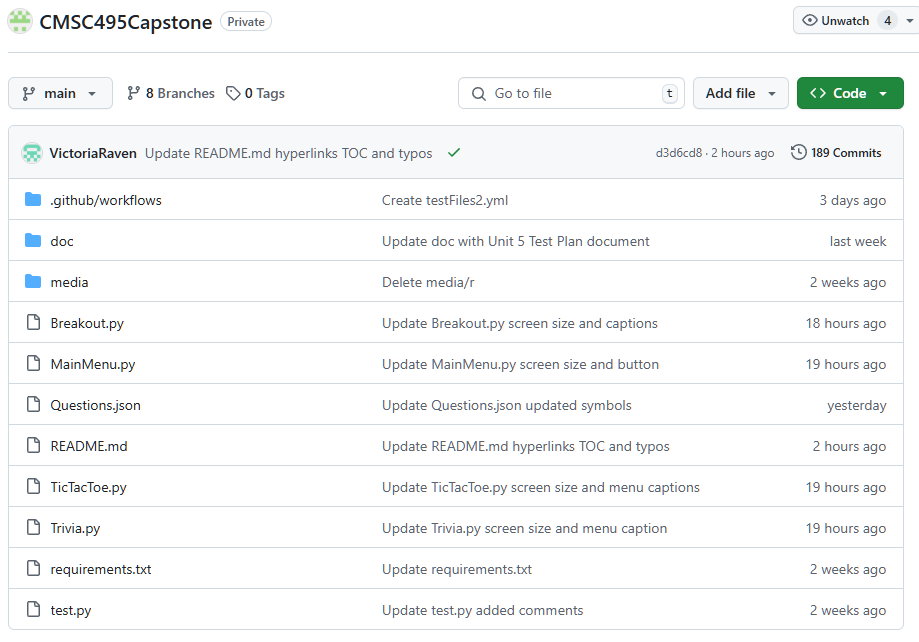
**Explanation:**

Our team uses GitHub to collaborate and track progress throughout the duration of the project. We maintained a shared repository on GitHub where each team member cloned the main branch into their own branch. Before each member was able to clone from the main branch, there had to be an established workflow. This consisted of four separate .py files in which they all connected to the MainMenu.py file, where the program starts. These steps assisted the team tremendously in the long run by making it easy to merge code seamlessly.

**To ensure a smooth collaboration, we followed a structured process:**

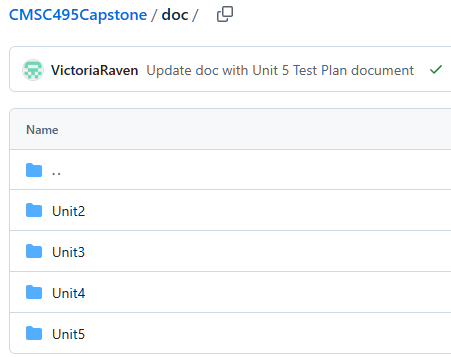
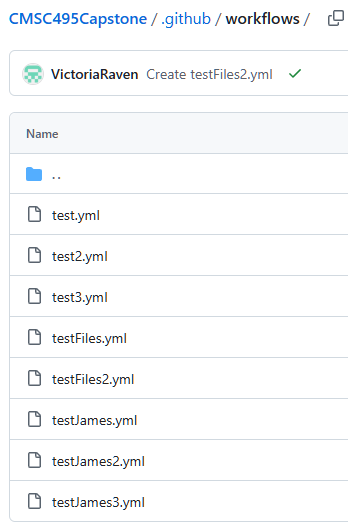
1. Feature Development: Each team member worked on a separate feature within their branches
2. Review and Commit: Before commits are made, each team member uploads their changes to their branch. The Git Lead then pulls each member's changes and tests them on the local environment. Once all code is confirmed to operate as expected it is uploaded to the main branch on GitHub. After it has been uploaded, each member pulls the code from the main and confirms that it is working. If the commit ever causes complications, we will simply revert the main branch back to its previous version.
3. Conflict resolution: We occasionally encounter merge conflicts, in which we discuss and collaborate using Git’s merging tools. If there are any bugs or glitches found, we simply record it in the issues section and begin repairing.
4. Tracking progress: To track changes we use GitHub commit history and issue tracking to monitor progress and assign tasks.

**Main branch:**

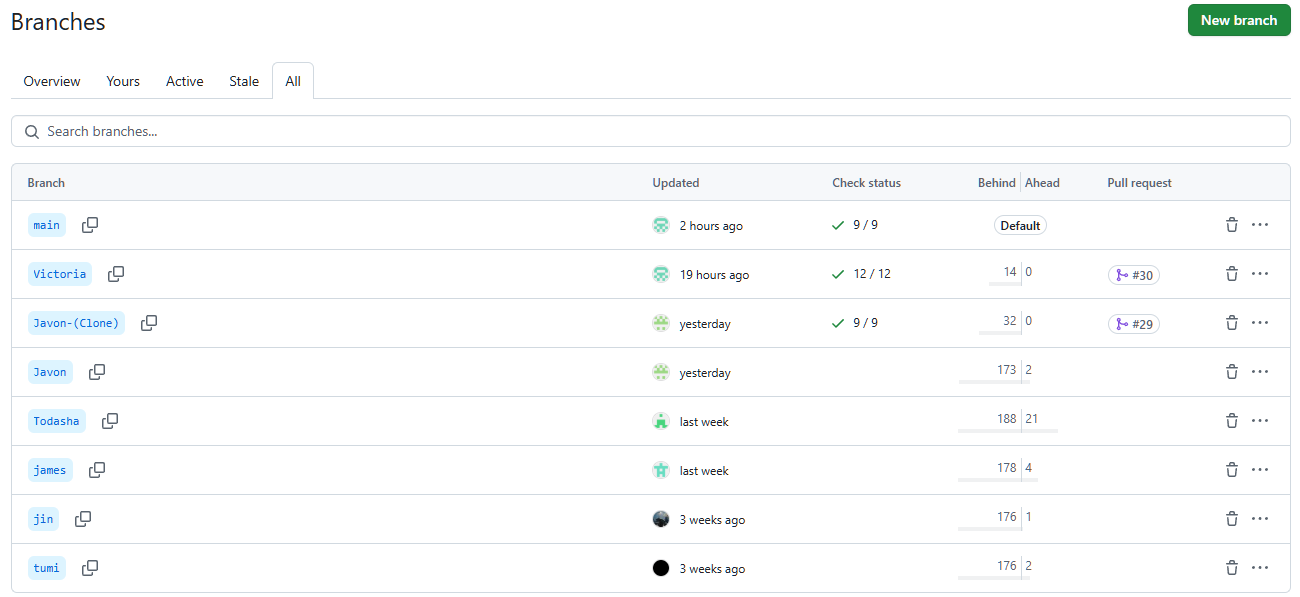


Media files and doc files and .github/workflow files:

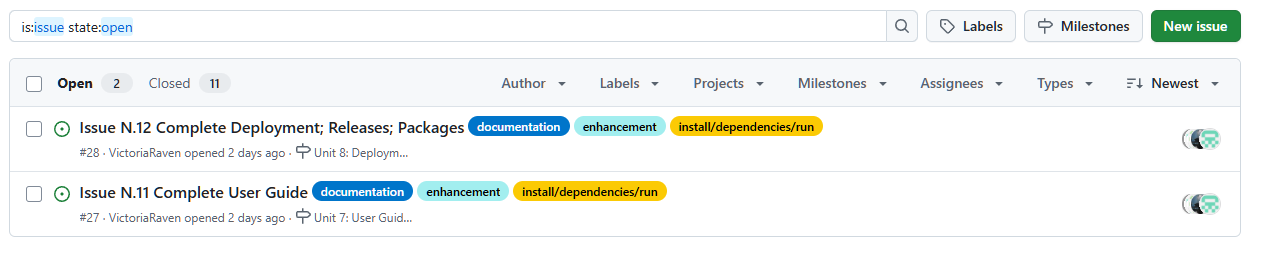
A screenshot of a computer

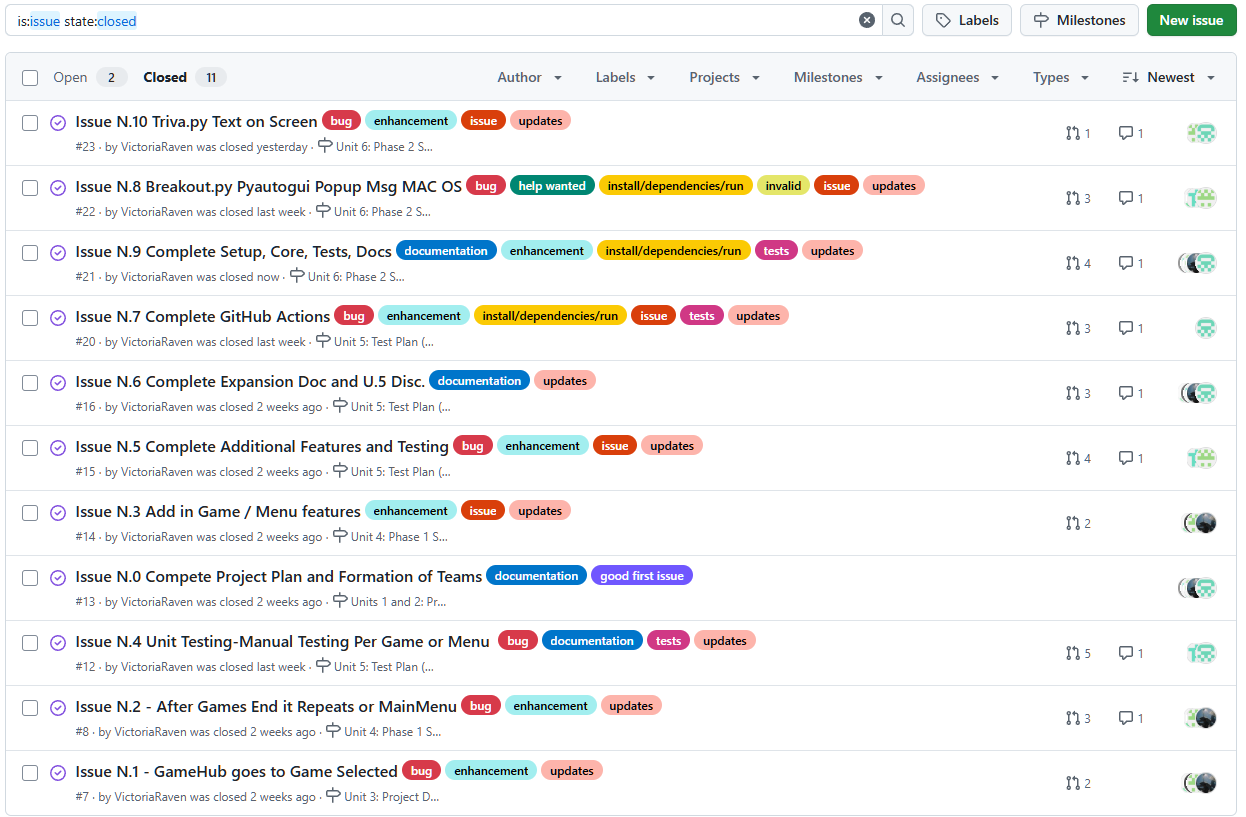
AI-generated content may be incorrect.  

We have a total of 8 branches:

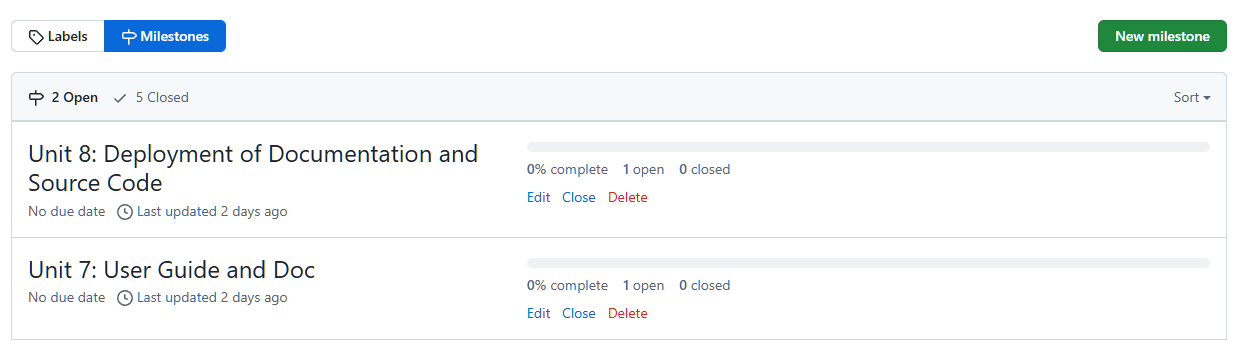


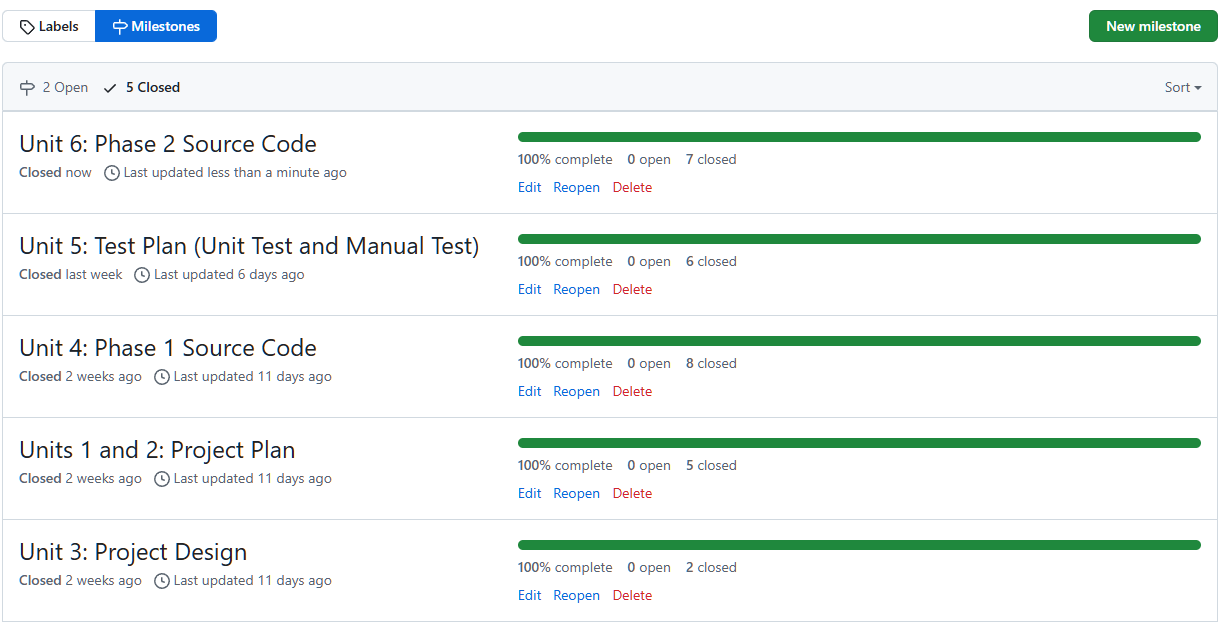
These are All our issues both Open and Closed:



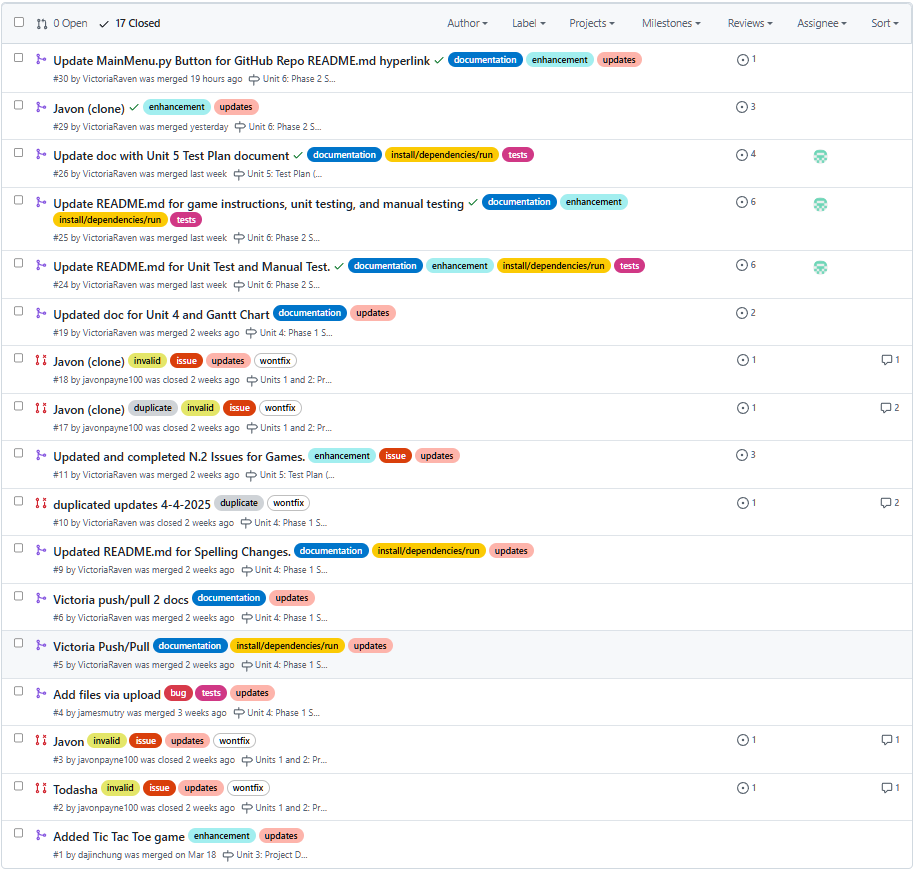


These are our milestones (open and closed) (this was updated on 4/18/2025):

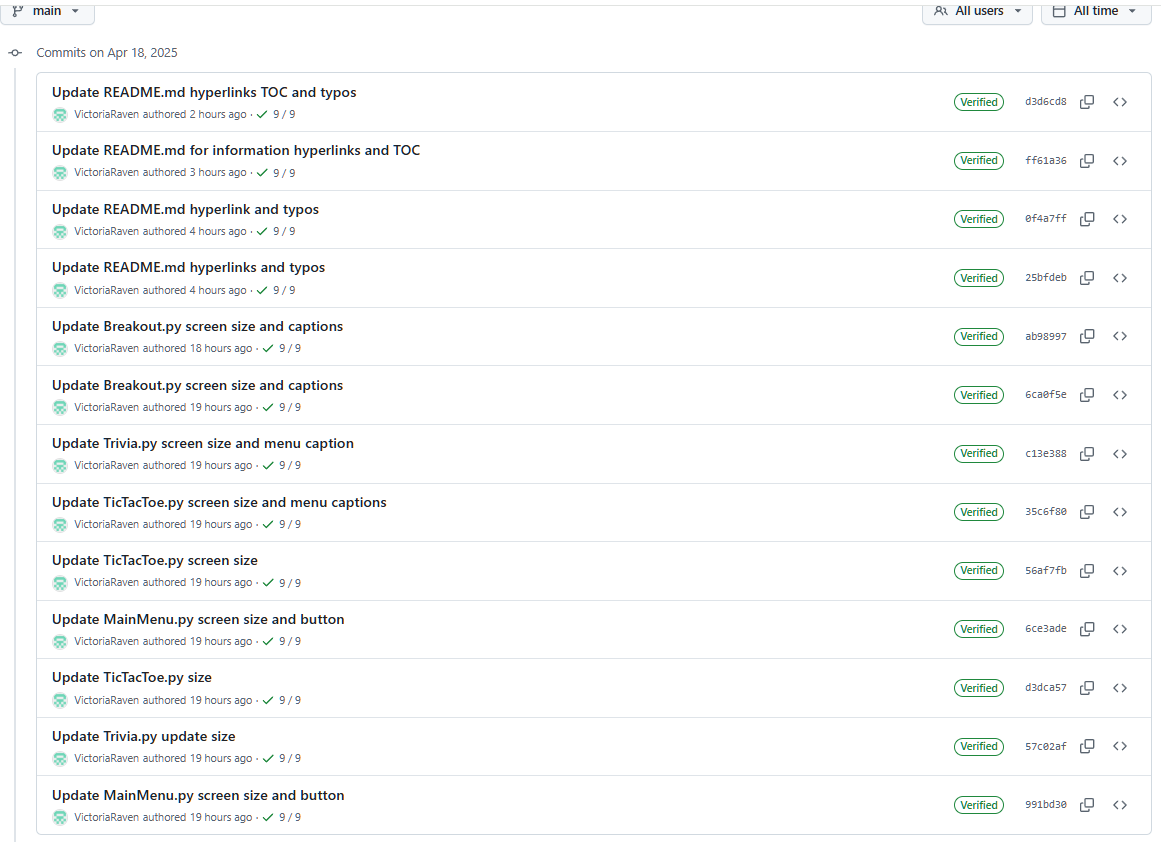


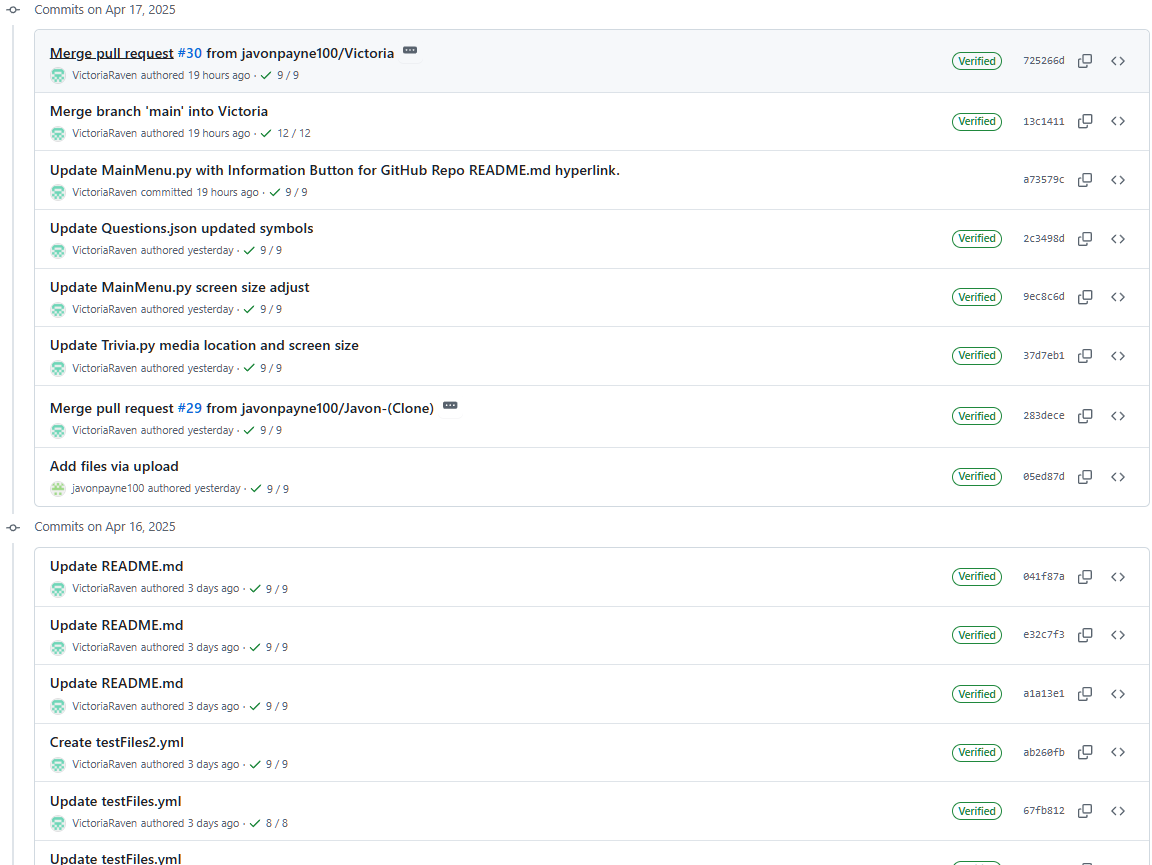


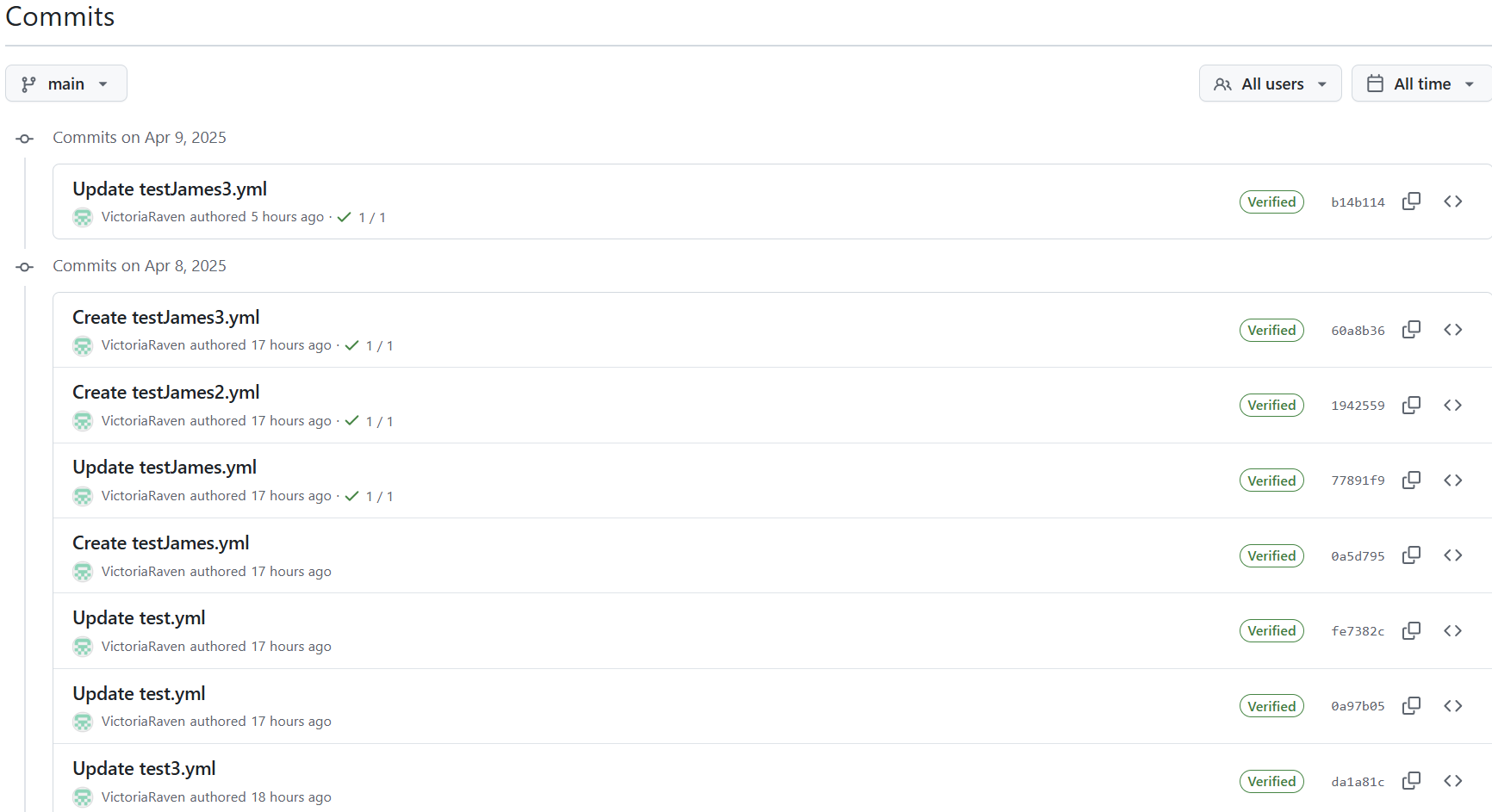
Here is our pull requests to merge onto the main using the branches:

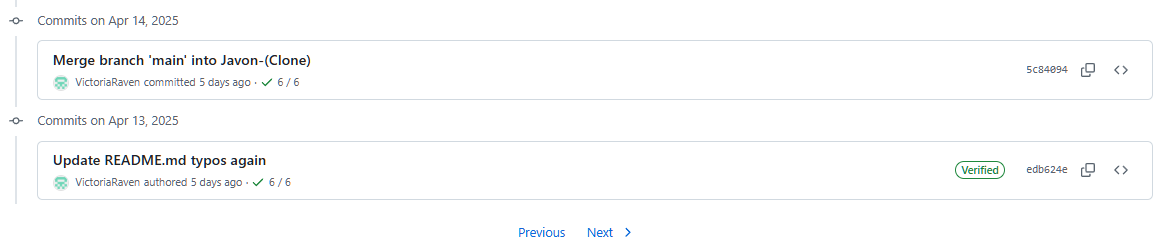


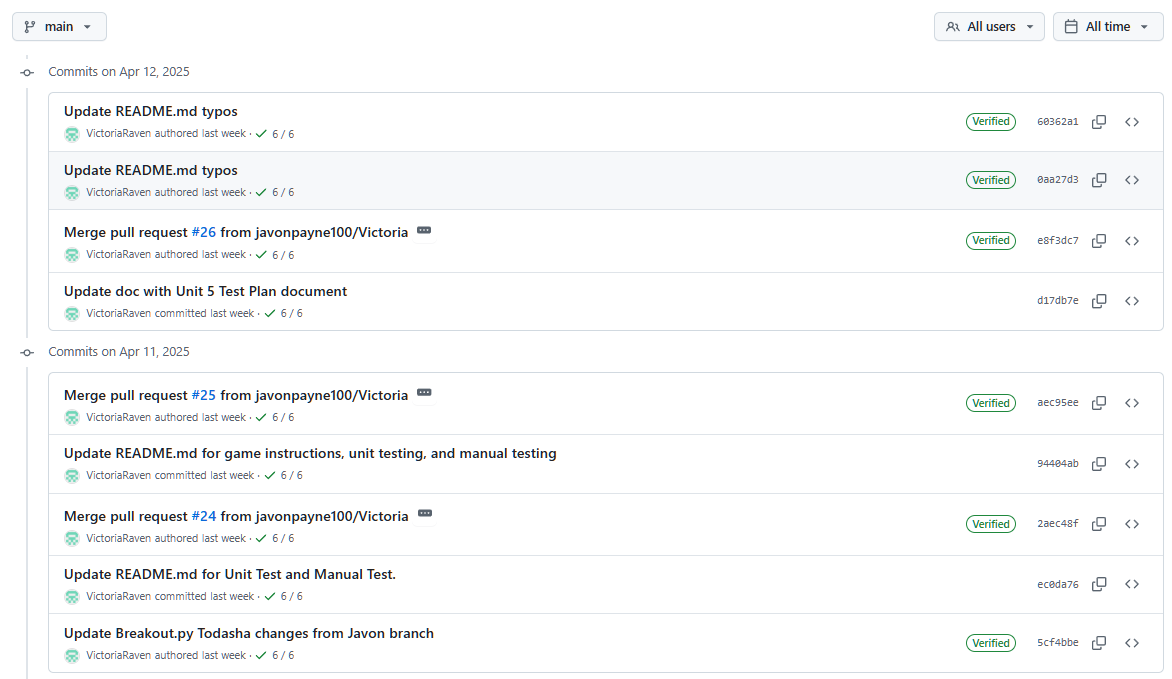
We have a total of 189 Commits (I will not show all screenshots as it is quite long; only main parts):

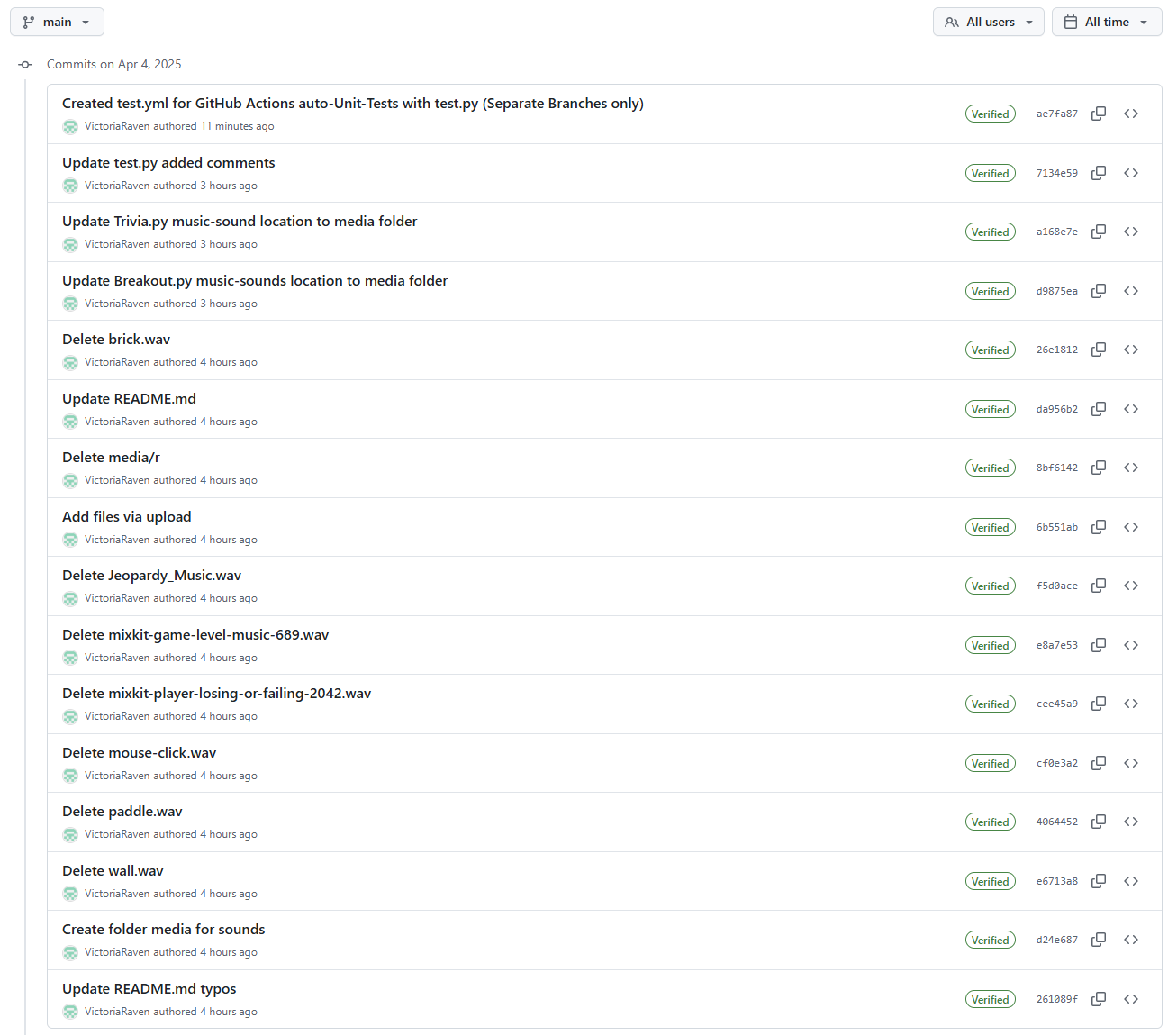


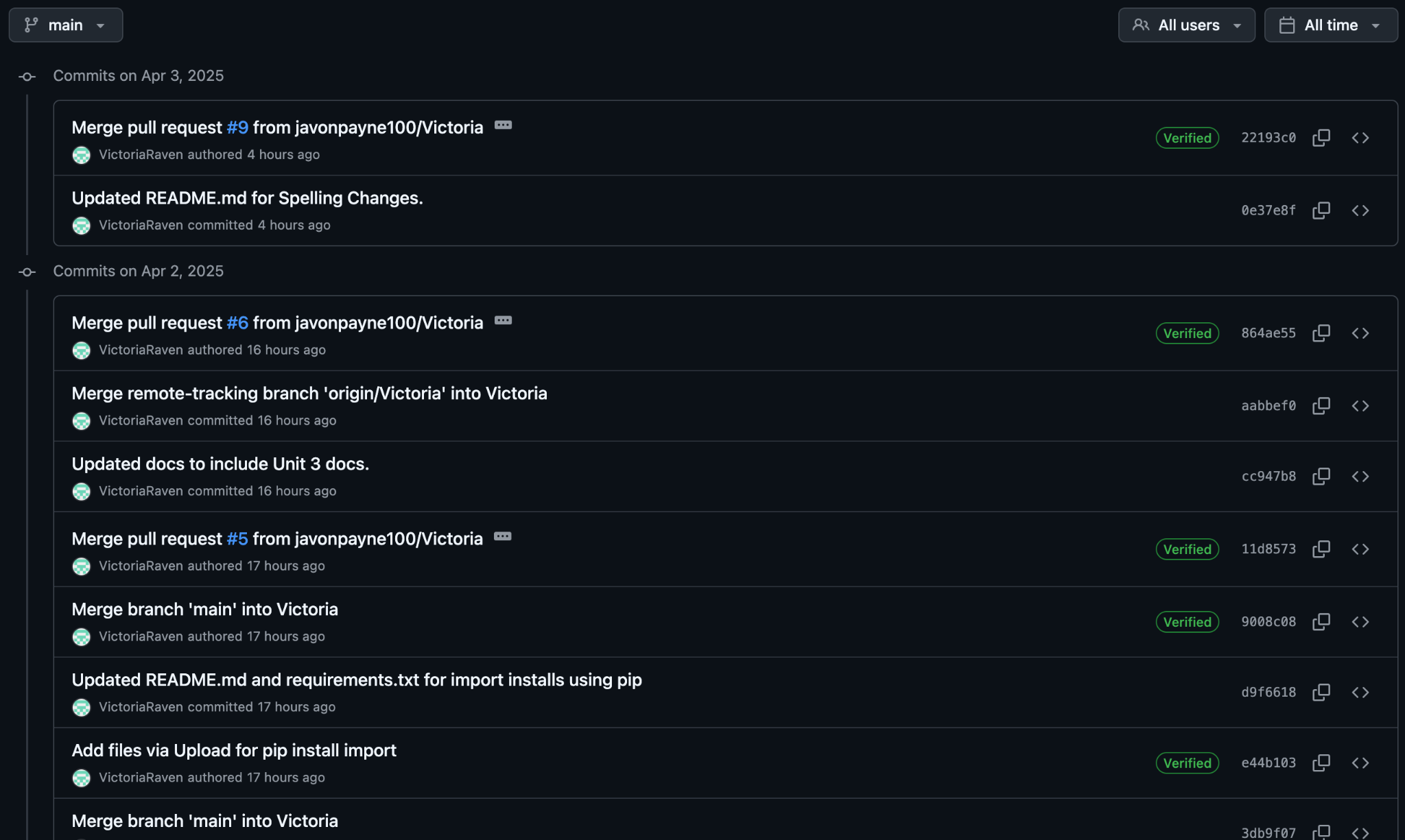


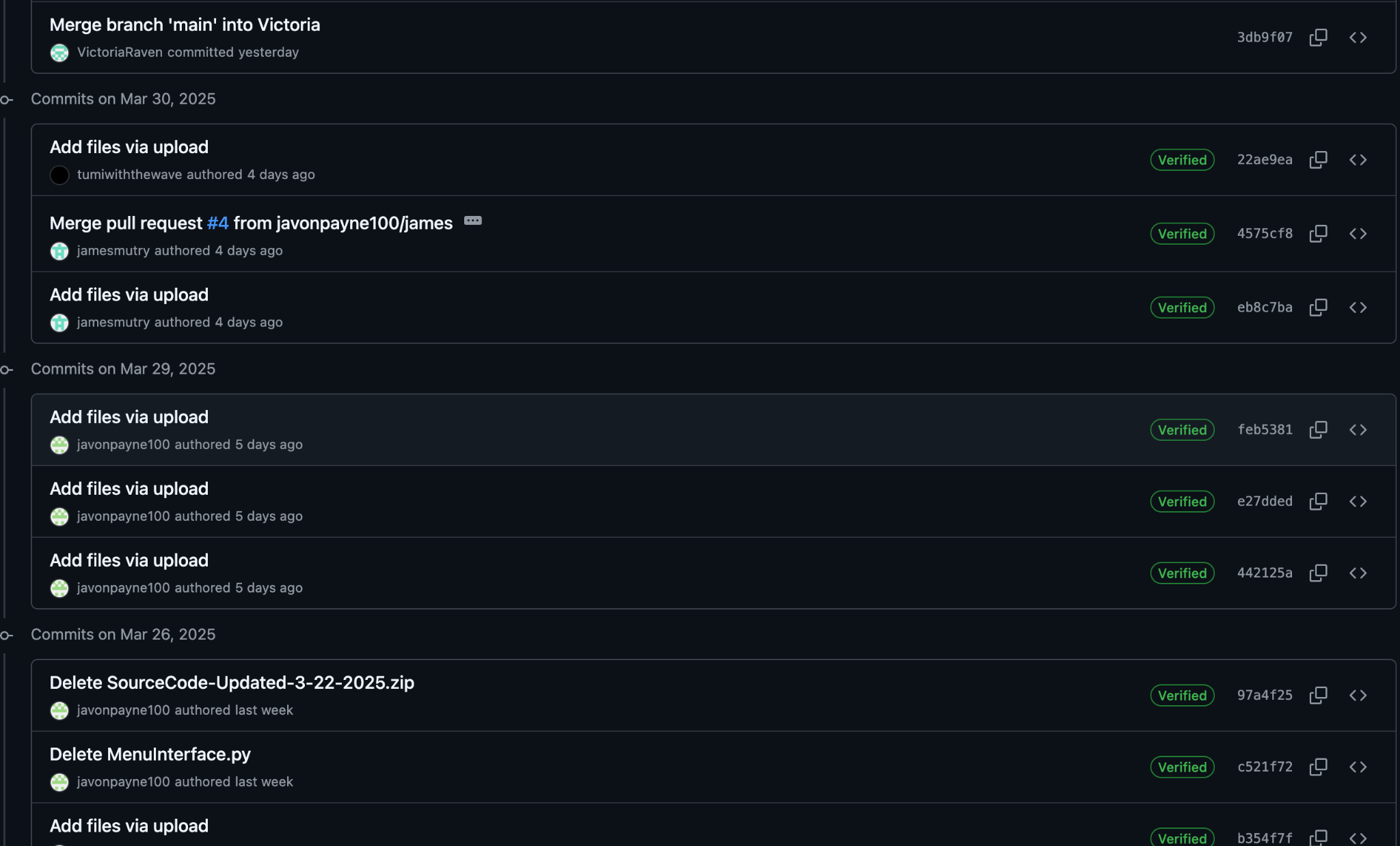














# **13. References**

Amazon Web Services. (n.d.). *What is unit testing?* Amazon Web Services, Inc.<https://aws.amazon.com/what-is/unit-testing/>

Deitel, P. J., & Deitel, H. (2015). *Java: How to program (Early objects)* (10th ed.). Pearson. Online textbook available. (n.d.)

Devaraj, K. (2024, December 19). 6 key phases of testing in software testing. *Testsigma*.<https://testsigma.com/blog/phases-of-testing/>

Doe, J., & Lee, A. (2020). Creating effective contribution reports in software projects: Guidelines and strategies. *IEEE Software Engineering Journal*, 45(6), 102–115.<https://doi.org/10.1109/SEJ.2020.3082548>

Erickson, J. (n.d.). *Algorithms*. University of Illinois at Urbana-Champaign. Online textbook available.

GeeksforGeeks. (n.d.). *Software Development Life Cycle (SDLC)*.<https://www.geeksforgeeks.org/software-development-life-cycle-sdlc/>

GeeksforGeeks. (2024). *Control Flow Software Testing*.<https://www.geeksforgeeks.org/control-flow-software-testing/>

GeeksforGeeks. (2024). *Software engineering: Differences between coupling and cohesion*.<https://www.geeksforgeeks.org/software-engineering-differences-between-coupling-and-cohesion/>

GeeksforGeeks. (2024). *Difference between Function-Oriented Design and Object-Oriented Design*.<https://www.geeksforgeeks.org/difference-between-function-oriented-design-and-object-oriented-design/>

GeeksforGeeks. (2024). *Object-Oriented Design (OOP)*.<https://www.geeksforgeeks.org/oops-object-oriented-design/>

GeeksforGeeks. (2024). *Functions decomposition in software engineering*.<https://www.geeksforgeeks.org/functions-decomposition-in-software-engineering/>

GeeksforGeeks. (2024). *What is a Project Plan*.<https://www.geeksforgeeks.org/project-plan/>

Gehry, F. (Producer). (2019, April 10). *How to make your design stand out* [Video]. YouTube.<https://www.youtube.com/watch?v=Va64slVDpI4>

Git. (n.d.). *Git user manual*.<https://git-scm.com/docs/user-manual>

Jones, M., & Davis, L. (2019). Effective project design planning in software engineering: Best practices and strategies. *IEEE Software Engineering Conference Proceedings*, 31(2), 95–110.<https://doi.org/10.1109/SEConf.2019.2787093>

Khan, S., & Patel, A. (2018). Designing and implementing a centralized game hub for multiplayer platforms. *IEEE Transactions on Game Development*, 39(1), 77–89.<https://doi.org/10.1109/TGD.2018.2876062>

Kleinberg, J., & Tardos, É. (2006). *Algorithm design*. Pearson Education.

LeetCode. (n.d.). *LeetCode - The world's leading online programming learning platform*. LeetCode. Retrieved April 12, 2025, from<https://leetcode.com/>

Lehman, E., Leighton, F. T., & Meyer, A. R. (2021). *Mathematics for computer science*. LibreTexts, MIT, & UC Davis.

Liang, Y. D. (2023, May 23). *Introduction to Java programming and data structures* (13th ed.). Pearson. Online textbook available. (n.d.)

Mount, D. (n.d.). *Algorithms [Lectures CMSC 251]*. University of Maryland Global Campus.

Nievergelt, J. (n.d.). *Algorithms and data structures*. University of Maryland Global Campus.

Sierra, K., & Bates, B. (2005). *Head First Java* (1st ed.). O'Reilly Media.

Smith, J., & Taylor, R. (2021). Using Git for version control in Python-based game development. *IEEE Transactions on Software Engineering*, 47(3), 215–230.<https://doi.org/10.1109/TSE.2021.3056789>

Solomon, J. (2015). *Numerical algorithms*. MIT.

Stellman, A., & Greene, J. (2005). Software project planning. In *Applied software project management* (Part I, Chapters 5, 6, 8, 9, 10, 11). O'Reilly Media.<https://go.oreilly.com/umgc/https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch08.html>

Sweigart, A. (n.d.). *Making games with Python & Pygame*.<https://inventwithpython.com/makinggames.pdf>

Tsui, F., Karam, O., & Bernal, B. (2014). *Essentials of software engineering* (3rd ed.). Jones and Bartlett Learning.<https://library-books24x7-com.ezproxy.umgc.edu/toc.aspx?site=VGX8U&bookID=51648>

University of Maryland Global Campus. (n.d.). *Sample Project Plan*.<https://leocontent.umgc.edu/content/dam/course-content/tus/cmsc/cmsc-495/document/2020%20Sample%20Project%20Design%2004-2022.pdf?ou=1248245>

Unity Technologies. (2020, September 3). *Testing and quality assurance tips for Unity projects*.<https://unity.com/how-to/testing-and-quality-assurance-tips-unity-projects>

Winters, T., Manshreck, T., & Wright, H. (2020). Software engineering at Google (Chs. 18–19). O'Reilly Media.<https://learning.oreilly.com/library/view/software-engineering-at/9781492082781/ch18.html>