Test Plan

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.xdajmz58d75y)

[**2. Testing Objectives 3**](#_heading=h.uyoopjk7ozwl)

[**3. Test Strategy 5**](#_heading=h.8kffypomfliw)

[**4. Test Scope: 7**](#_heading=h.4qsvsozcy6i6)

[**5. Test Cases 8**](#_heading=h.b95g9a3pz0sy)

[**6. Testing Procedures 12**](#_heading=h.v1lf5t13bkl3)

[**7. Testing Schedule 15**](#_heading=h.6e3xpnblfwi)

[**8. Testing Resources 18**](#_heading=h.uupk02tae1vc)

[**9. Defect Management 21**](#_heading=h.w111q6zd1nf5)

[**10. APPENDIX A: 24**](#_heading=h.5ohq9zveqqpb)

[**11. APPENDIX B: 29**](#_heading=h.w2h0mq9gj5tw)

[**12. APPENDIX C: 36**](#_heading=h.p0gc76i7xat5)

[**13. APPENDIX D: 37**](#_heading=h.ptzpvhv1n8uq)

[**14. APPENDIX E: 41**](#_heading=h.pe3p4on3pj4q)

[**14. References 49**](#_heading=h.hmw9sc4sq8wc)

# **Introduction**

Our team will use **3 main Testing methods: Unit Test, Manual Test, and GitHub Actions.** The Python Game Hub project has three modules: Tic Tac Toe, Trivia, and Breakout. The goal is to create a Python-based game suite with interactive, user-friendly gameplay. Testing will ensure each module functions as intended, covering both functional and non-functional aspects, including performance and usability. The testing plan includes 24 unit tests: 5 for the Main Menu, 6 for Trivia, 5 for Tic-Tac-Toe, and 8 for Breakout, along with 20 manual tests based on the provided Excel sheet, totaling 40 tests. These tests will assess functionality (game mechanics, AI behavior), performance (speed, responsiveness), usability (UI intuitiveness), and reliability (bug-free behavior). The testing scope covers game rules, AI responses, collision detection, and game state transitions but excludes cross-platform or stress testing. Test cases will focus on specific features like Tic Tac Toe’s winning conditions, Trivia’s score tracking, and Breakout’s ball movement. Testing will follow a 1-week schedule (Unit 5), including preparation, functional and non-functional testing, defect resolution, and final reporting. Resources include personnel, equipment, and a budget for tools. Any defects found will be tracked, prioritized, and resolved through retesting.

# **Testing Objectives**

**Testing Purpose and Objectives:** Our team will use 3 main Testing methods: **Unit Test, Manual Test, and GitHub Actions**. The primary purpose and objectives of the testing effort for this software project are to ensure the functionality, reliability, and usability of the Python Game Hub, which includes the Tic Tac Toe, Trivia, and Breakout modules. For testing out all four parts of the .py files (MainMenu, Tic-Tac-Toe, Trivia, and Breakout), we will implement about ~40 Tests (unittest and manual tests) (functional and non-functional parts). Specifically, 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, included with 20 manual testing based on the EXCEL sheet provided, which is a total of about 40 Tests for the Python Game Hub Test Plan. This makes it 24 unit tests, 20 manual tests for the Python Game Hub Test Plan. The primary objectives of the testing effort focus on the Quality Attributes below:

1. **Functionality**: Verifying that each game component (Tic Tac Toe, Trivia, and Breakout) works as expected, adhering to defined rules and mechanics.
   1. Tic Tac Toe
      1. validate win condition detection and AI behavior
      2. CPU only and make AI smarter.
      3. goes back to the main menu / play again after the game ends
   2. Trivia
      1. Check question data integrity and answer validation
      2. Allow 1-2 players and measure scores as well
      3. allow the user to choose what subject to play with and include questions in the JSON file.
      4. goes back to the main menu / play again after the game ends
   3. Breakout
      1. Confirm ball reset logic and collision detection.
      2. allow the user to use the paddle, which includes music, the user can hit some blocks with 1 hit, else the user has to hit the block multiple times to get it to be removed.
      3. The user has 3 lives, and scores are updated
      4. music is integrated for sound effect and game over music
      5. goes back to the main menu / play again after the game ends
2. **Game Flow:** Assess the smooth transition between game states, including initialization, user input handling, and game completion.
3. **Performance / Stress Testing**: Ensuring the application performs optimally under typical conditions and handles edge cases without significant delays or crashes.
4. **Usability Testing**: Evaluating the user interface and interaction flow to make sure that players can easily interact with the game, understand the rules, and navigate between menus.
5. **Regression Testing:** Ensure new updates or changes do not break existing functionalities or cause unintended side effects.
6. **Accuracy and Robustness**: Verifying the correctness of game logic, including win detection in Tic Tac Toe, answer validation in Trivia, and block collision detection in Breakout. And identifying potential error scenarios, such as invalid inputs, and ensuring the system handles these gracefully without failures or crashes.

**Unit Testing and Manually Testing:** Automated testing was implemented using Python’s unittest framework to validate critical components of the Tic Tac Toe, Trivia, and Breakout modules. The test suite verifies game mechanics such as win detection, board state, and AI behavior in Tic Tac Toe. For Trivia, tests were written to inspect the structure of the question data loaded from JSON, as well as the correctness of answer indexing. The Breakout tests include health reduction of blocks, ball reset logic, paddle boundaries, and collision detection between game elements. We also approached the manual testing by following the sample Test plan on the EXCEL sheet for the manual tests and did it similarly to the unittest, but without the code.

**Next Phase Testing:** Following the Waterfall development methodology, future testing will be structured as a distinct phase conducted after the completion of full system implementation. This dedicated testing phase uses unit testing and manual testing, targeting each module, Tic Tac Toe, Trivia, and Breakout to verify core functionalities such as AI decisions, answer validation, collision detection, and object behavior. Once unit-level verification is complete, the process will move into integration testing to ensure smooth interaction between components, including menu navigation, game transitions, and audio/visual responses.

System testing will follow, simulating real-world usage scenarios to confirm that the application performs reliably under typical and extreme conditions. Lastly, acceptance testing will be performed to validate that all project requirements have been met and the application aligns with stakeholder expectations. Testing outcomes will be documented in detail, and any issues uncovered will be addressed through a feedback loop before final deployment. This structured approach ensures a thorough and sequential validation of the system’s readiness for delivery. Through the testing approach, we will also use the software engineering techniques of Unit Testing and Manual Testing (functional/non-functional parts) to ensure that each part is tested and debugged correctly.

**GitHub Actions Testing with .yml files:** 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 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.

**Testing 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 ofthe features and games are clearly defined from the outset. **As of now, for Unit 5, we are currently in both the Implementation and Verification parts.** In Unit 6, we will be on the Verification and Maintenance. In Unit 7, we will be on 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 and APPENDIX B. APPENDIX C is the agreement for delays.



# **Test Strategy**

The testing approach will employ both manual and automated(unit test and GitHub actions) testing techniques using the following methodologies of (1) unit test (2) manual test (3) GitHub actions through the Waterfall approach as mentioned in 2. Test Objectives:

1. **(1 and 2) Unit Testing and Manual Testing**: Automated unit tests using Python’s unittest framework will be employed to verify that individual components (e.g., AI decision-making, question indexing, collision detection) perform as expected. Manul will be done through the EXCEL sheet provided by the sample on Unit 5. It should have 24 unittests and about 20 manual tests, with a total of about ~40 tests.
   1. **Integration Testing**: After testing, integration testing will be conducted to ensure the individual modules (Tic Tac Toe, Trivia, Breakout) interact seamlessly within the application, including game state transitions, menu navigation, and overall flow.
   2. **System Testing**: System testing will simulate real-world usage scenarios to assess how the entire system performs under normal and extreme conditions. This will include testing the full game suite to ensure consistency, reliability, and stability.
   3. **Acceptance Testing**: Before release, acceptance testing will verify that the game meets all project requirements and stakeholder expectations.
2. **(3) Automated Testing (GitHub Actions):** This will use GitHub actions to test the unittest from the previous unit and current unit 5 and make sure it confirms as if the person ran it manually through a Python IDE (3 GitHub Actions for old test.py and 3 for new test.py)
3. To **track progress** on each coding / bugs section, we will use **Git for Version Control**.
   1. Each process will include a Git commit section
   2. Each readme file will ensure that the user knows how the set the requirements of the software and run the application to play the games.
   3. 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 and APPENDIX B. APPENDIX C is the agreement for delays.

Previously, it was mentioned that we approached Testing through Unit Testing and Manual Testing and then the next process moves into integration testing, which ensures the modules interact seamlessly, including checking menu navigation, game transitions, and audio/visual responses. Following integration, system testing will simulate real-world conditions to ensure reliable performance under both normal and extreme scenarios. Finally, acceptance testing will confirm that all project requirements are met and that the game aligns with stakeholder expectations. The third way we approached this is the GitHub Actions to make sure that it work sas well virtually through Git’s (CI/CD) platform for all three systems (Mac, Windows, Ubuntu/Linux). Throughout this process, any identified issues will be addressed through a feedback loop to ensure a thorough validation before final deployment. This structured, step-by-step approach aims to ensure that each part of the system is tested and debugged properly.

Our team opted to use the unittest module from Python for unit testing, as opposed to relying on GitHub Actions for automated unit testing with each push in the first half of unit 5 week. Using the unittest module allows teams to write and run tests locally on their development environment (**test.py** file), making it simple to test individual components without needing to interact with version control systems. This approach is highly flexible, requires the team to manually run tests, and using this method of import eliminates the difficulties and adapts to each of the team members’ skill sets, effectively making the project’s progress complete faster. In contrast, GitHub Actions automates the testing process with each push and can be rerun again manually when the workflow is enabled, which ensures tests are always run as part of the development cycle. For this case, during the last half week of unit 5, Victoria has made GitHub Actions (3 for old test.py file on main branch and 3 on new test.py file on james branch) to double check with Jame’s runs on the IDE manually. **Unit 5** will mostly have the **.py file where you manually run them through the IDE** or the project itself, including the manual testing based on the EXCEL sample using the “james” branch on GITHUB. But with Project Manager/ Tester Victoria, she created a **GitHubActions to double check** different systems of Jame’s unittest test.py files (Mac and Windows, with exception to Ubuntu as pygame and pyautogui has ongoing issues on their git repository where it creates glitches for Ubuntu).

# **Test Scope:**

**Testing Boundaries:** The testing scope will focus on the core game mechanics and interactions within each game module (Tic Tac Toe, Trivia, and Breakout). Specifically, the following elements will be included in the testing effort:

1. **Tic Tac Toe**: Verification of win conditions, board state updates, AI behavior, and player input handling.
2. **Trivia**: Validation of question structure loaded from JSON, correctness of answer indexing, and handling of user input.
3. **Breakout**: Testing of paddle movement, block health reduction, ball reset logic, collision detection, and game-over conditions.
4. **In Scope**: Functionality of game mechanics, game logic (win conditions, scoring), interaction with user interface, game transitions, input handling, and AI behavior.
5. **Out of Scope**: Testing of the Python environment, external libraries, or the hardware it runs on. Non-functional aspects like detailed performance under extreme conditions beyond typical gameplay will be assessed, but not deeply stress-tested.
6. **Other:** Testing out manually (through PyCharm/Python IDE) with team members running the game file and making sure the game goes back to the main menu after it ends, or allows the user to play again.

**Testing Limitations:** Certain aspects of the system will not be included in the testing efforts due to time constraints or project scope limitations:

1. **Performance Testing**: While the system will be tested for basic performance in typical use cases, extensive performance testing (e.g., stress testing or load testing with a high number of users) is outside the scope.
2. **User Interface Design**: Although usability is a quality attribute under evaluation, extensive testing of user interface aesthetics and responsiveness across different screen sizes and resolutions will not be performed unless a critical issue arises during the testing process.
3. **Third-Party Integrations**: Any external libraries or tools, such as audio or visual assets, will not be thoroughly tested unless they directly affect the functionality of the game mechanics.
4. **Other:** Limited by available resources and time. The test will primarily focus on functional testing, with less emphasis on advanced performance or stress testing. Testing will be constrained to the core features of the game, and certain peripheral features (e.g., non-essential animations) may be deprioritized.

Note: The extra screenshots for Testing and Git version control processes are further detailed in APPENDIX D and APPENDX E. The Milestones and Gnatt chart updates with the Contribution Report are in APPENDIX A and APPENDIX B. APPENDIX C is the agreement for delays.

# **Test Cases**

**Unittest (Python file):** 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.

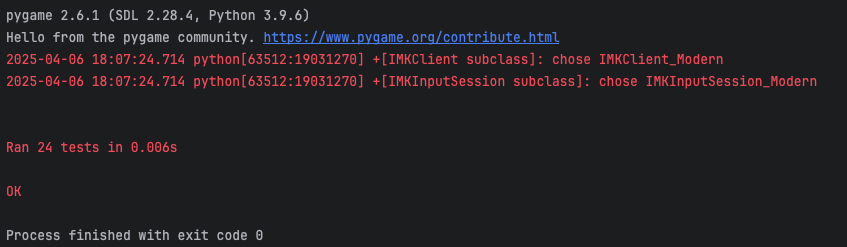
**test.py:** The link below renders the old **test.py from unit 4 (main branch)**. This file contains all the Unit Testing code and the 13 ways of testing our application using the import unittest. We added this to unit 5 to show we used GitHub Actions to run it through and also confirm it from past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

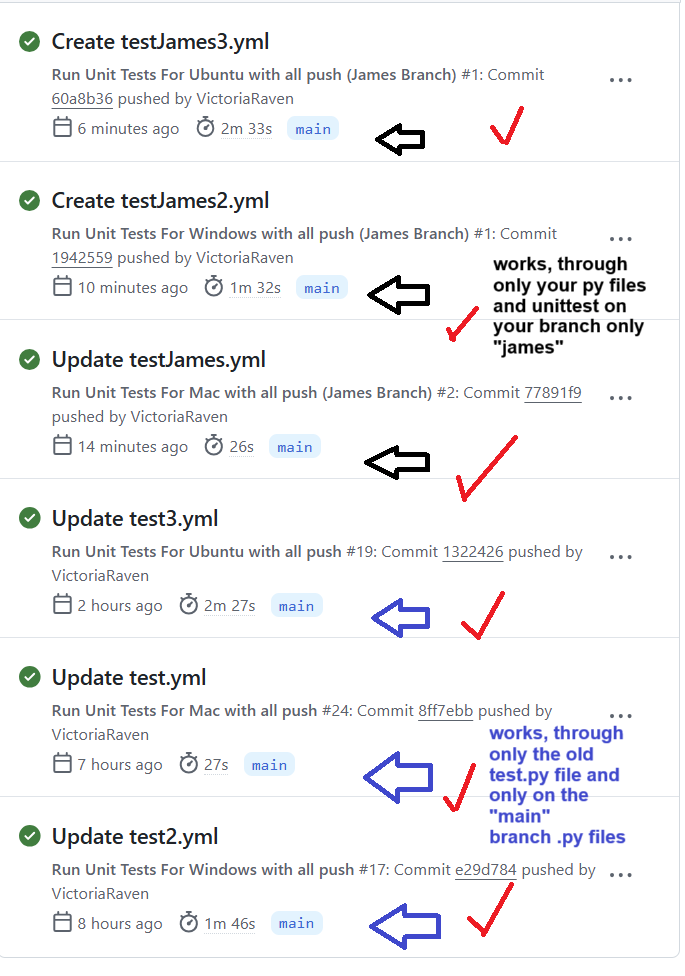
<https://gist.github.com/9ebc800353d91b027d9912409596f271.git>

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

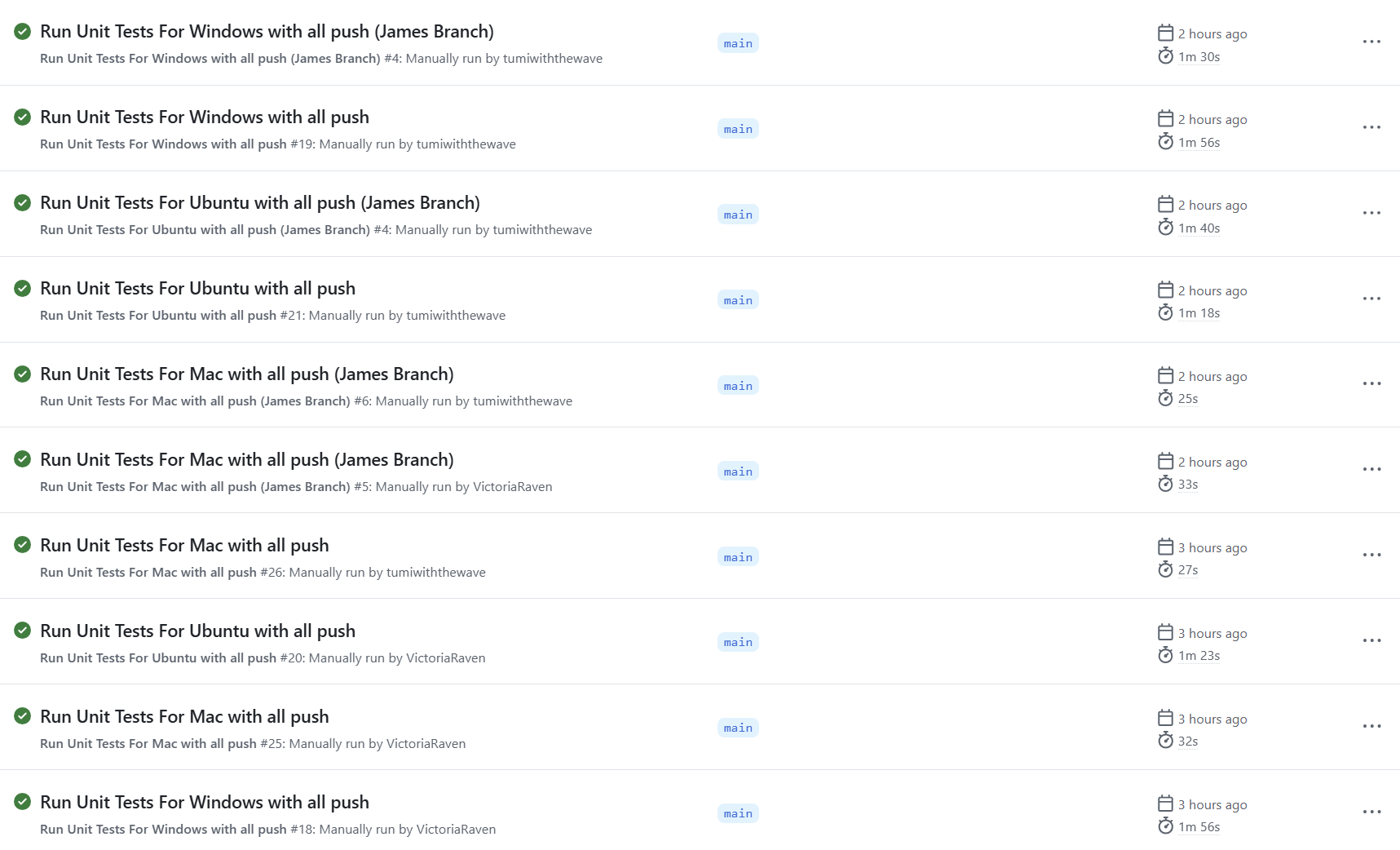
Link to the new test.py file:

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

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



**Additional Git Hub Action Screen Shots testing on all ends:**

****

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

**Manual Testing (Excel):** 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. 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.

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.

# **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 the user how the unit test is processed.

<https://gist.github.com/22eecdaa8f994da6d90b70ebf9fb1370.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)**. 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 unnitest 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 Repositry 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:**

[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 manual, 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 manual, 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 Git and Tests. The Milestones and Gnatt chart updates with the Contribution Report are in APPENDIX A and APPENDIX B. APPENDIX C is the agreement for delays.

# **Testing Schedule**

### **Roles based on Testing Schedule:**

### **Tester / Technical Writer (Dates: 4/5/2025 - 4/11/2025)**

* **Team Members:** James the Tester will create and test about 40 tests, including Unit Testing and Manual Testing phases. He will:
  + Create and test the functionality based on unit and manual tests.
  + Fill out sections 5, 6, and 9 of the Test Plan documentation.
  + Assist Victoria with the overall Test Plan, and Victoria will assist with the other sections that need to be filled out and she will inclement 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 (total 6 GitHub actions yml files)

### **Development Team Involvement (Dates: 4/5/2025 - 4/11/2025)**

* **Team Members:** Javon, Todasha, Dajin, and Oluwatumininu will assist in the Unit Testing and Manual Testing phases. They will:
  + Test the functionality based on unit and manual tests.
  + Provide feedback and identify issues.
  + Update the current test schedule, tasks, milestones, and GitHub charts, as required.

### **Test Plan Technical Writer / Tester (Dates: 4/5/2025 - 4/11/2025)**

* **Role:** Victoria will:
  + Develop the Test Plan document, excluding specific sections that are for the tester to complete.
  + Modify the README.md file to clarify testing instructions for the development and testing teams.
  + Ensure that the final plan is detailed and comprehensive.
  + 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
    - this should double check with Jame’s manual IDE python runes for the test.py files.

We will use **6 phases of the Software Testing Life Cycle (STLC)** (Devaraj, 2024, December 19) for our Test Scheudle to make tasks, milestones, and complete them as below:

1. **Requirement Analysis** – Understanding and documenting the requirements for testing
   1. Dates: 4/4/2025 to 4/6/2025
2. **Test Planning** – Creating a detailed test plan, defining objectives, timelines, and responsibilities
   1. Dates: 4/5/2025 to 4/7/2025
3. **Test Case Development** – Writing detailed test cases and planning for test automation
   1. Dates: 4/8/2025 to 4/11/2025
4. **Test Environment Setup** – Configuring the environment required for testing
   1. Dates: 4/8/2025 to 4/9/2025
5. **Test Execution** – Running the test cases on the system and logging results
   1. Dates: 4/8/2025 to 4/11/2025
6. **Test Closure** – Analyzing test results, defect reporting, and finalizing testing activities
   1. Dates: 4/11/2025 to 4/12/2025

#### **1. Requirement Analysis (Dates: 4/4/2025 - 4/5/2025)**

* **Objective:** Gather functional and non-functional requirements of the software.
* **Actions:**
  + **Team:** The Testing team collaborates with the Project Manager and Technical Writers to define and clarify the system’s functionality and requirements.
  + **Output:** Clear and documented software requirements, ensuring alignment with the overall goals and user needs.

#### **2. Test Planning (Dates: 4/5/2025 - 4/7/2025)**

* **Objective:** Develop a detailed plan for the testing process, including tools, resources, and timelines.
* **Actions:**
  + **Team:** Testers, developers, Project Manager, and Technical Writers to define roles, responsibilities, test objectives, methods, tools, and test schedule.
  + **Plan:** A test plan document will outline the scope, test strategies, resources, roles, responsibilities, and milestones, ensuring a systematic testing approach.
  + **Timeline:** The test plan will be finalized within 6 days (4/5/2025 to 4/11/2025), with a focus on integration, system, and acceptance testing phases.

#### **3. Test Case Development (Dates: 4/8/2025 - 4/9/2025)**

* **Objective:** Design, develop, and review test cases based on the defined requirements.
* **Actions:**
  + **Team:** Testers create test cases that cover functional, integration, and system scenarios, ensuring each case is well-defined with expected outcomes.
  + **Automation:** Identify test cases suitable for automation and begin preparing for test automation where applicable.
  + **Review:** The test cases will be reviewed to ensure completeness and accuracy.

#### **4. Test Environment Setup (Dates: 4/8/2025 - 4/9/2025)**

* **Objective:** Set up the necessary hardware, software, networks, and databases to simulate the production environment.
* **Actions:**
  + **Team:** Ensure that the test environment replicates real-world conditions, optimizing for performance and load testing.
  + **Automation Setup:** Configure automated tests for execution once the environment is ready.
  + **Output:** A fully operational test environment ready for execution.

#### **5. Test Execution (Dates: 4/8/2025 - 4/10/2025)**

* **Objective:** Execute test cases on the actual system, validating functionality and performance against the requirements.
* **Actions:**
  + **Team:** Testers execute the defined test cases, logging results, and reporting defects.
  + **Types of Testing:** Unit Testing, Integration Testing, System Testing, and Acceptance Testing will all be conducted.
  + **Manual Testing:** Testers will also perform manual tests, especially using EXCEL for specific scenarios.
  + **Documentation:** Ensure thorough documentation of test results for future analysis and defect reporting.

#### **6. Test Closure (Dates: 4/10/2025 to 4/12/2025)**

* **Objective:** Finalize all testing activities, analyze results, and assess the software’s readiness for release.
* **Actions:**
  + **Team:** Conduct a review meeting with all stakeholders to assess if all objectives were met and if the software is free of critical defects.
  + **Defect Reporting:** Document all defects and ensure they are addressed before the final release.
  + **Test Closure Report:** A final report will be generated, summarizing the results of all testing phases and confirming that the software meets the acceptance criteria.

Here is a simplified version of it if you want to exclude all details and get straight to the point:

* **Unit Testing (Mostly import unittest with .py file and very few GitHub Actions)**
* **Dates: 3 days from 4/8/2025 to 4/10/2025**
  + Also Includes:
    - **Integration Testing**
    - **System Testing**
    - **Acceptance Testing**
* **Manual Testing (EXCEL)**
* **Dates: 3 days from 4/8/2025 to 4/10/2025**
  + Also Includes:
    - **Integration Testing**
    - **System Testing**
    - **Acceptance Testing**
* **Adjusting Testing Schedules (Milestones/Tasks/Git/Other Trackers):**
* **Dates: 1 week from 4/5/2025 to 4/11/2025**
  + **Developers:** TJavon, Todasha, Dajin, and Oluwatumininu will test out the Unit Test and Manual Tests after the Tester creates it and completes it, and they will update the current Test Schedule and Current (tasks/milestones/git/charts/contributions), since developers are not needed this week..
  + **Test Plan Technical Writer:** Victoria will write out the document (except for 5 and 6, and 9, which are for the Tester to explain). This person will also modify the README.md file to make sure the instructions are clear. This person will also 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. This should double check with Jame’s manual IDE python runes for the test.py files.

# **Testing Resources**

**The Testing Roles for preparing the Testing Resources are as follows:**

* **Tester:** James will create the Unit Tests and Manual Tests (Total of 40 Tests). The tester will also write down the steps and explain the purpose of each in sections 5 and 6 of the Test Plan. The tester will also write down the defect management part to resolve defects.
* **Developers:** TJavon, Todasha, Dajin, and Oluwatumininu will test out the Unit Test and Manual Tests after the Tester creates it and completes it, and they will update the current Test Schedule and Current (tasks/milestones/git/charts/contributions), since developers are not needed this week..
* **Test Plan Technical Writer:** Victoria will write out the document (except for 5 and 6, which are for the Tester to explain). This person will also modify the README.md file to make sure the instructions are clear. This person will also implement the GitHub Actions.
* **Additional Features:** Javon will complete and finish the Trivia game for the additional features, as the rest are all done (Main menu, Tic-Tac-Toe, Breakout). This makes sure that we can also include the testing in this section as well.
* **Testing Resources - Equipment**: Personal computers with Python 3.x installed, necessary IDEs such as PyCharm are highly recommended, and testing libraries (e.g., unittest, mock, imports, etc.).
* **Testing Resources - Budget**: Testing will be performed using available hardware and free tools. No additional budget is allocated.The total budget is $0.
* 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 and APPENDIX B. APPENDIX C is the agreement for delays.

**Testing Resources Report:**

This report outlines the identification and allocation of necessary resources for the testing effort. It incorporates the key project management concepts, project design, testing methods, and project budgeting principles. Below are the details of the allocated personnel, equipment, budget, and tracking processes.

### **1. Personnel Allocation**

| **Role** | **Responsibilities** | **Project Management Concept** |
| --- | --- | --- |
| **Tester (James)** | - Create **Unit Tests** and **Manual Tests** (total of 40 tests). | Task ownership and clear responsibility in test creation. |
|  | - Write test steps and explain the purpose in **sections 5 and 6** of the Test Plan. | Resource allocation to ensure testing coverage and detailed documentation. |
|  | - Manage **defect management** to address issues found during testing. | Defect resolution as part of the overall project management process. |
| **Developers (Tjavon, Todasha, Dajin, Oluwatumininu)** | - Validate tests after creation by the Tester. | Iterative testing and collaboration between testers and developers. |
|  | - Update the **Test Schedule** and track **milestones** (Git commits, tasks, progress). (Help out with project manager concepts) | Regular progress tracking and milestone management. |
| **Test Plan Technical Writer (Victoria)** | - Write the Test Plan document, excluding sections 5 and 6. | Documentation ensures clear instructions and efficient team communication. |
|  | - Modfy/Implement **GitHub Actions .yml** file and modify **README.md** for clear setup and testing instructions. | Clear and concise documentation to support testers and developers. |
| **Additional Features Developer (Javon)** | - Complete the **Trivia Game** to ensure all features are available for testing. | Incremental development and ensuring that features are ready for testing. |

### **2. Equipment Allocation**

| **Equipment** | **Purpose** | **Project Management Concept** |
| --- | --- | --- |
| **Personal Computers** | - Each team member will use a personal computer with **Python 3.x** installed. | Resource management to provide the necessary tools for each team member. |
| **Integrated Development Environment (IDE)** | - **PyCharm** or preferred IDE will be used for development and testing. | Efficient use of development tools for streamlined testing and coding. |
| **Testing Libraries** | - Use libraries like **unittest**, **mock**, and other Python testing tools for executing Unit and Manual tests. | Ensuring the appropriate resources (tools) are allocated to meet testing requirements. |
| **Testing with GitHub Actions (only with old test.py file!)** | -Create a YAML file in the github and create .github/workflow to activate the github actions to run the py file automatically.. This will only work for the old test.py file from unit 4. The new test.py file will not include this setup and will run it manually for the .py and include additional manual testing. | Ensuring the appropriate resources (tools) are allocated to meet testing requirements.  This will also test the OS compatibility and what other issues remain. (Victoria) will implement this to ensure that the old test file works. |

### **3. Budget Allocation**

| **Category** | **Description** | **Amount** | **Project Management Concept** |
| --- | --- | --- | --- |
| **Testing Tools** | - Use of **open-source testing tools** like **unittest**, **mock**, and **Python** libraries. | $0 | Cost management by utilizing free resources to stay within budget. |
| **Hardware** | - Personal computers (already available) with **Python 3.x** installed. | $0 | Resource management leveraging existing hardware. |
| **IDE** | - **PyCharm** or other free IDEs. | $0 | Efficient tool allocation using free IDEs for development and testing. |
| **Additional Budget for Testing** | - No additional budget is allocated for testing; all tools and resources are free or already owned. | $0 | Adherence to budget constraints with efficient resource utilization. |

### **4. Progress Tracking and Updates**

| **Task** | **Responsibilities** | **Project Management Concept** |
| --- | --- | --- |
| **Test Schedule Updates** | - Developers will update the **Test Schedule** and track **milestones** (e.g., Git commits, tasks, progress). | Regular progress updates for effective tracking. |
| **Defect Management** | - Log and address defects found during testing. Categorize defects by severity and prioritize resolution. | Ongoing defect management to maintain project quality. |

### **5. Testing Methods and Processes**

| **Testing Method** | **Purpose** | **Project Management Concept** |
| --- | --- | --- |
| **Unit Testing** | - Test individual components of the application to ensure their functionality is correct. | Iterative testing to catch issues early. |
| **Manual Testing** | - Perform exploratory testing to cover edge cases and ensure usability. | Comprehensive testing covering user experience. |
| **Continuous Feedback** | - Regular feedback loops between developers, testers, and technical writers. | Ensures all issues are addressed promptly, reducing rework. |

# **Defect Management**

### **Defect Identification:**

During the testing process, any defects (bugs or issues) that are identified will be logged onto a defect tracking table for tracking and managing all defects. This tracking system ensures transparency, accountability, and allows everyone to stay aligned on defect resolution priorities. Each defect will be logged as an **ID number** based on the **Modules (MainMenu.py, TicTacToe.py, Trivia.py, Breakout.py, Others).** The defect/ bug report will clearly define the issue, including:

* **Module**: The specific game or module (Tic Tac Toe, Trivia, Breakout) where the issue was found.
* **Defect Description**: A clear description of the behavior or issue that caused the defect, including how it was triggered during testing.
* **Categorization**: The defect will be categorized by type, such as functional errors, UI issues, or performance concerns. This will be done using **labels** for easy filtering and identification.

**Defect Reporting:**

Testers will create a detailed issue report based on id number about the defect they identify. The defect card will include essential information, such as:

* **Steps to Reproduce**: A step-by-step guide that explains how to reproduce the issue.
* **Severity**: An assessment of the defect’s severity, categorized as **Critical**, **Major**, **Minor**, or **Trivial**, based on its impact on game functionality or user experience.
* **Impact**: A description of how the defect impacts the game. For example, a critical bug, like an AI failure in Tic Tac Toe that prevents the game from progressing, will be marked as high priority, while cosmetic defects may be marked as lower priority.

### **Defect Tracking / Verification and Closure:**

Once defects are logged in to the document table bleow, where the testers and developers will analyze and fix the defect. It will represent different stages of defect resolution:

* **To-Do** (Defect logged and under review)
* **In Progress** (Developer actively working on the fix)
* **Resolved/Closed** (Fix implemented, awaiting retesting | Issue resolved and confirmed)

After a developer/tester has implemented a fix, the defect will be retested. The original tester will rerun the test cases that identified the defect to ensure the issue has been resolved without introducing new problems. If the defect is successfully fixed, it is marked as resolved/closed. However, if the issue persists or the fix introduces new problems, it will be marked as in progress or to-do. Additionally, regression testing will be conducted to ensure that the fix has not negatively impacted other areas of the game. By following this detailed defect management process within this table, all issues encountered during testing will be documented, tracked, and resolved systematically. This approach ensures that defects are properly handled before the project moves forward to the next phase, whether that be integration, system testing, or deployment. This method aligns with the Waterfall methodology, ensuring that defects are addressed before proceeding further with development. Refer to APPENDIX D and APPENDIX E for Testing Schreenshots and Git verison control. The Milestones and Gnatt chart updates with the Contribution Report are in APPENDIX A and APPENDIX B. APPENDIX C is the agreement for delays.

**Below is the Defect Management Table using the Waterfall methodology and the Debugging/Testing issues report concepts/process for identifying, reporting, and tracking defects or bugs encountered during testing. It should list the bugs here that was encountered during testing and the mitigation:**

| **Category** | **Label** | **MainMenu.py** | **Tic-Tac-Toe.py** | **Triva.py** | **Breakout.py** | **Other (test.py)** |
| --- | --- | --- | --- | --- | --- | --- |
| Bug ID | **ID Number** | **#1** | **#1** | **#1** | **#1** | **#1** |
| Title | Music | Delay | Quit on Trivia | Ball glitch | Pop up |
| Reporter | James | James | James | James | James |
| Submit date | 4/8/2025 | 4/8/2025 | 4/8/2025 | 4/8/2025 | 4/8/2025 |
| Bug overview | Summary | Open opening application, no music played. | Slight delay in the first move of the AI in comparison to the other moves the AI does. | The “Quit” does not fit in the window. | Upon missing the ball, if the paddle is on the left side of the screen. The ball will continually hit up under it and the user will lose all lives. | When running test.py. The game application was opening up. |
| Screenshot | See below this table labeled Trivia #1 | See below this table labeled TicTacToe #1 | See below this table labeled Trivia #1 | See below this table labeled Breakout #1 | See below this table labeled Trivia #1 |
| Environment | Platform | Mac | Mac | Mac | Mac | Mac |
| Operating System | Sequoia | Sequoia | Sequoia | Sequoia | Sequoia |
|  | PyCharm | PyCharm | PyCharm | PyCharm | PyCharm |
| Bug details | Steps to reproduce | Run MainMenu.py | Run TicTacToe.py | Run Trivia.py  Play one game and check the score screen. | Run Breakout.py | Run test.py |
| Expected result | Music played | AI makes move after user | “Quit” button fits into the screen. | The ball goes up to the bricks. | Test run and user is notified through terminal. |
| Actual result | No music Played | Stalled moved by the AI | The Quit button is not fitted into the screen. | Ball stays under the paddle if it is in the left side of the screen. | MainMenu.py briefly opens and closes |
| Bug tracking | Severity | Minor | Minor | N/A (visual) | Minor | Minor |
| Assigned to | Javon | Any | Any | Any | James |
| Priority | Low | Low | Low | Medium | Minor |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| Bug ID | **ID Number** |  | **Trivia #3** | **Trivia #2** |  |  |
| Title |  | Overlapping name | Questions |  |  |
| Reporter |  | James | James |  |  |
| Submit date |  | 4/8/2025 | 4/8/2025 |  |  |
| Bug overview | Summary |  | If users name is too long it will overlap onto the score | Longer questions do not fit into the screen. |  |  |
| Screenshot |  |  |  |  |  |
| Environment | Platform | Mac | Mac | Mac | Mac | Mac |
| Operating System | Sequoia | Sequoia | Sequoia | Sequoia | Sequoia |
| Browser | PyCharm | PyCharm | PyCharm | PyCharm | PyCharm |
| Bug details | Steps to reproduce |  | Type a long name into the player name.  Observe name over the score |  |  |  |
| Expected result |  | Name is on the left of the score. |  |  |  |
| Actual result |  | Name overlaps score |  |  |  |
| Bug tracking | Severity |  | Minor |  |  |  |
| Assigned to |  |  |  |  |  |
| Priority |  | Medium |  |  |  |
| Notes | Notes |  |  |  |  |  |

# **APPENDIX A:**

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 **70% 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 (70% 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 Week (Current Week)**: We are currently focused on testing the implemented features, and tests are being executed on the project. This is completed when we turn in the document. This includes finishing and testing the final feature, **Trivia.py**, which is part of the game suite for Unit 5.

#### **Upcoming Tasks (Remaining 30%):**

1. **Phase 2**: This includes finishing and checking (testing) the final feature, **Trivia.py**, which is part of the game suite for Unit 5.
2. **User Guide Documentation**: The user guide is yet to be completed, which will provide comprehensive instructions for using the software.
3. **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:

| **Feature** | **Current Status** | **Remaining Tasks** | **Progress** |
| --- | --- | --- | --- |
| **Mainmenu.py** | Completed and fully implemented. | No remaining tasks.  -Updated buttons, UI, updated look, and transitions. | 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** | Partially completed and will be completed and fully integrated during Unit 5. | Finish the implementation for **Unit 5** and include tests.  -Updates 1-2 player mode and also scores, music, and subject questions. | 100% |
| **Unit Testing (all games)** | Ongoing testing of all games. | Complete tests for **Trivia.py** and implement all tests (unittest and manual tests based on the Excel sample) | 100% |

### 

### **Current Testing Week and Plan for Final Testing:**

As we are currently in **Testing Week**, the main focus is on ensuring the correctness and reliability of the implemented features. Here's an overview of the progress and upcoming actions:

1. **Unit 5: Testing (Unit Tests and Manual Tests based on EXCEL SAMPLE)**:
   * All games and parts of Trivia.py have been implemented and are undergoing unit tests and manual tests.
   * The team is currently conducting manual tests and unit tests on the completed game features in Mainmenu.py, TicTacToe.py, Breakout.py, and Trivia.py.
   * The Trivia.py file needs to be completed for the extra text file halfway through the testing, so that it can be conducted.
   * The team will use GitHub actions to double check each IDE run manually with the test.py file.
2. **Unit 5 – Trivia.py**:
   * The **Trivia.py** file is still in progress. Once completed, it will be integrated with the existing game framework. Following this, unit tests will be executed.
3. **Unit 5 – Test Completion**:
   * After the Trivia.py implementation, the team will double-check all tests based on the changes made to Trivia.py and ensure all game features are thoroughly tested.
   * 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 6, 7, & 8 – Phase 2, User Guide Documentation, 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:

* **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. 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. And rerun tests through the GitHub actions or on the IDE.
* **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) | N/A % |
| **Week 7** | User Guide Documentation and Start Deployment | N/A % |
| **Week 8** | Deployment: Finish Docs, Code, Finalize Application | N/A % |
| **Overall Progress** |  | **70% Complete** |

### **Total Progression**

The project is currently **70% 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 **Trivia.py (Units 4 and 5)**, ensuring all **Tests (Unit 5)** 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.

# **11. APPENDIX B:**

CONTRIBUTION REPORT

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

Project: Python Game Hub

| 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 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 4/11/2025 | 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 |
|  |  |  |  |  |

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

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 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 onee 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 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).

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

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

# **12. 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 **4/11/2025**, our team has not made any delays and completing each unit within a week.

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

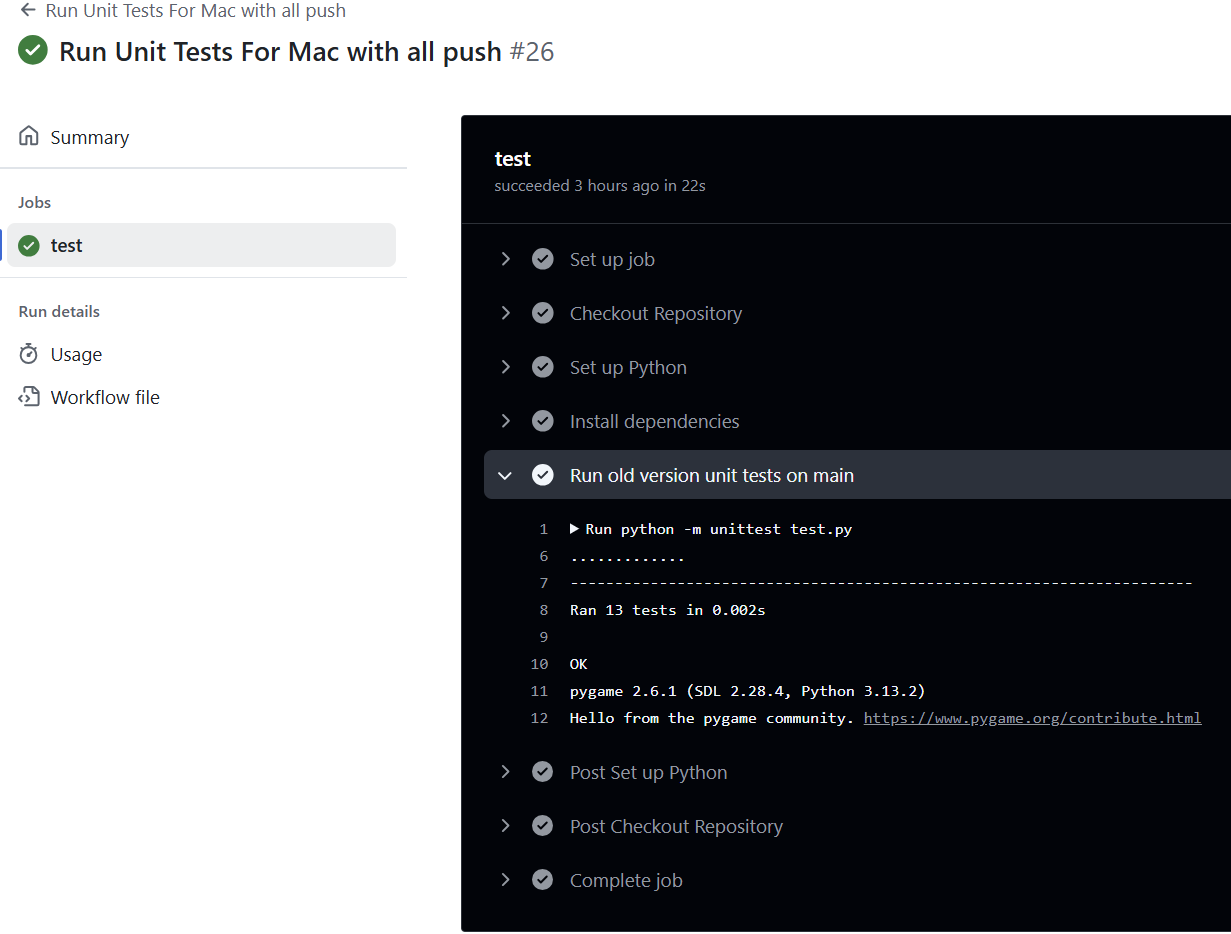
**Signed:  
Lee, Victoria; Mutry, James; Foster, Todasha; Chung, Dajin; Payne, Javon; Ipaye, Oluwatumininu**

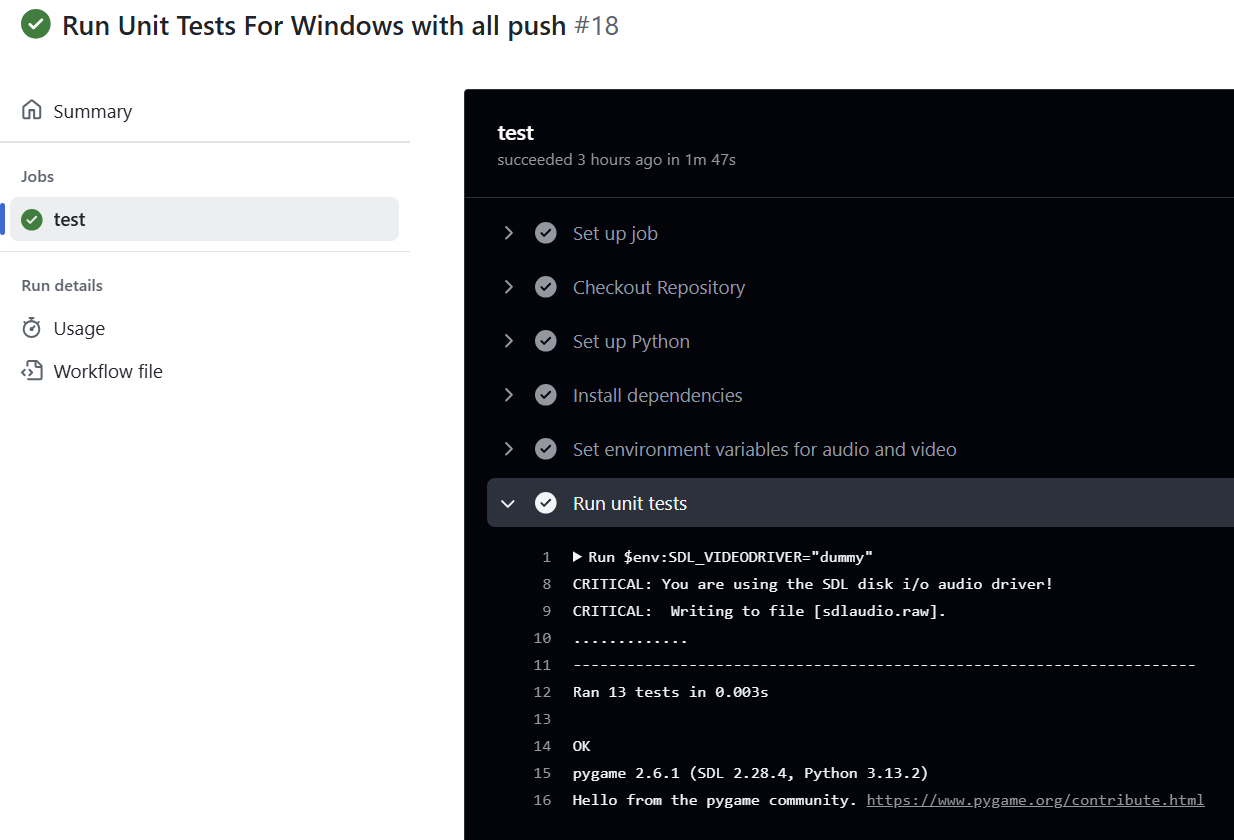
**Date: 3/18/2025 - 5/3/2025**

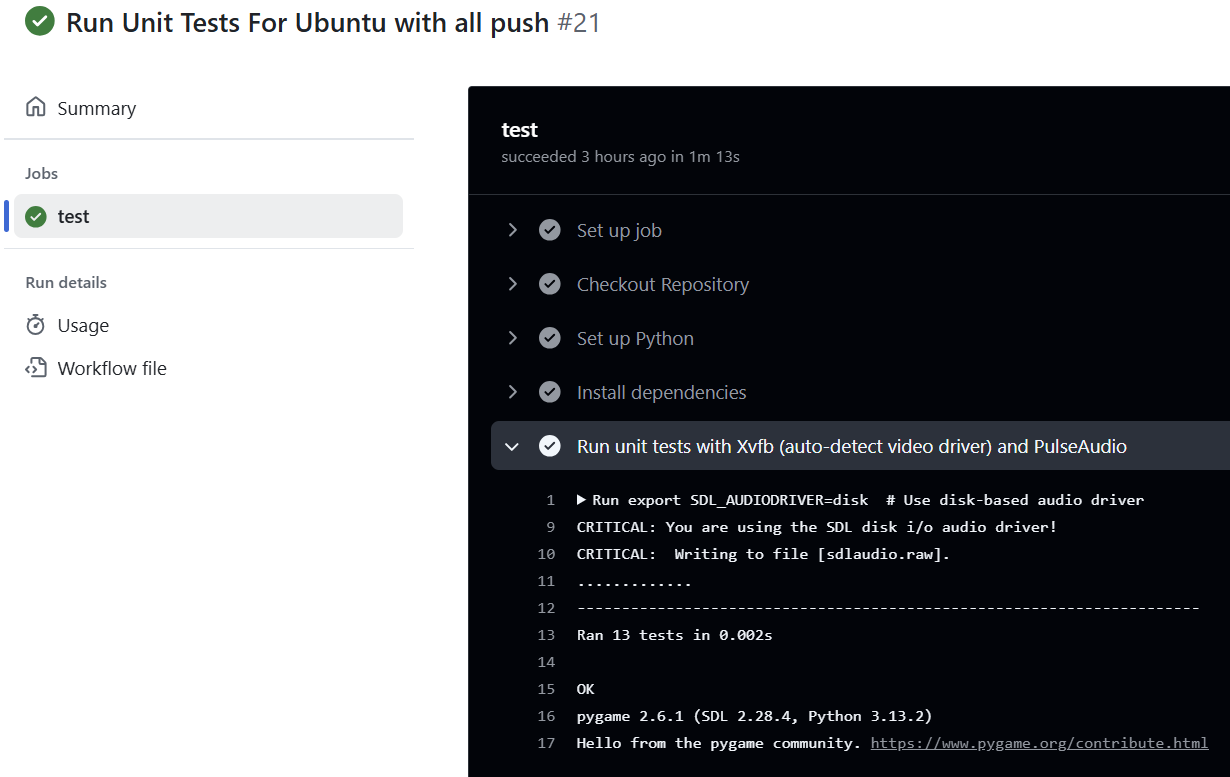
# **13. APPENDIX D:**

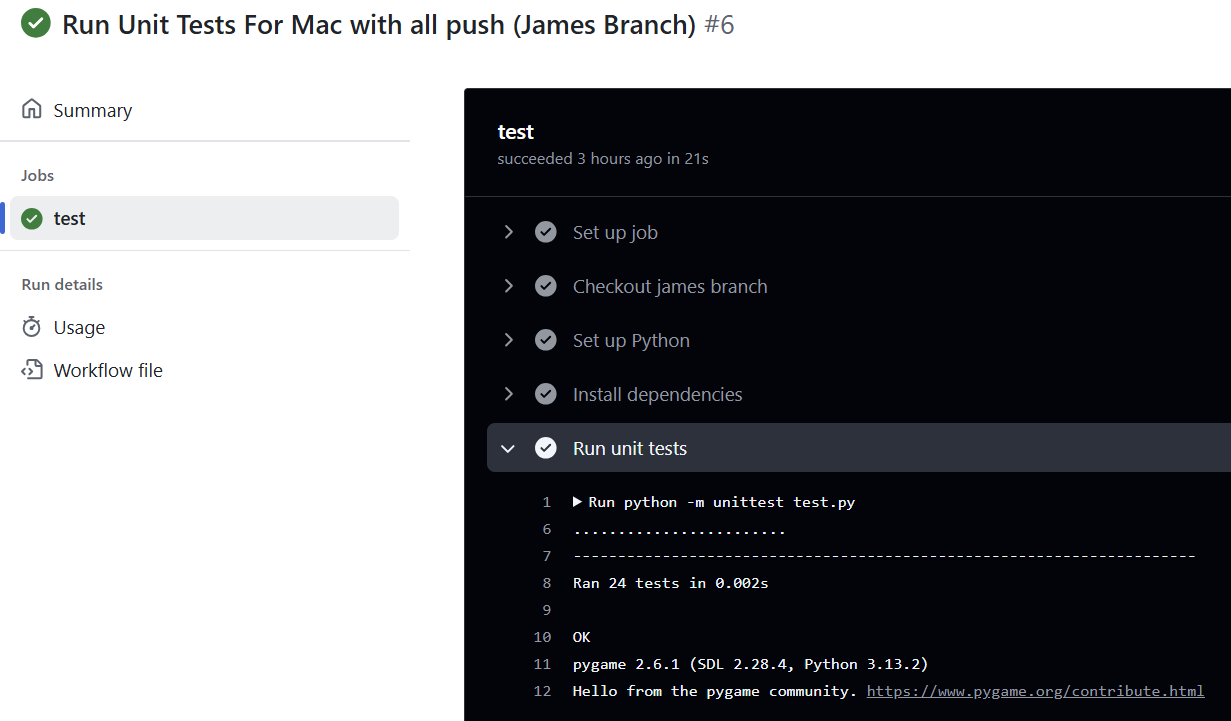
**Continued Screenshots for GitHub Actions working for all 6 unit tests:**

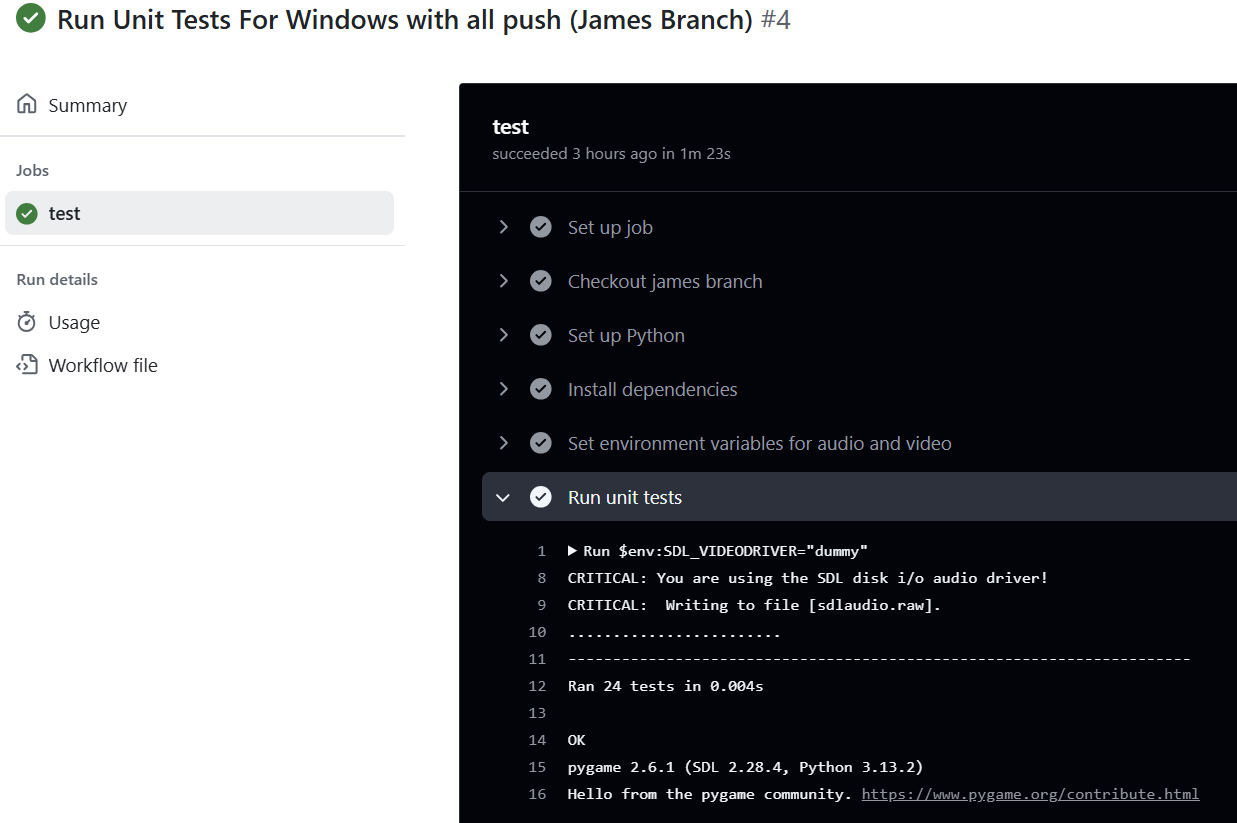
**(I placed it here as the document was becoming long if I didn’t put it in the APPENDIX)**

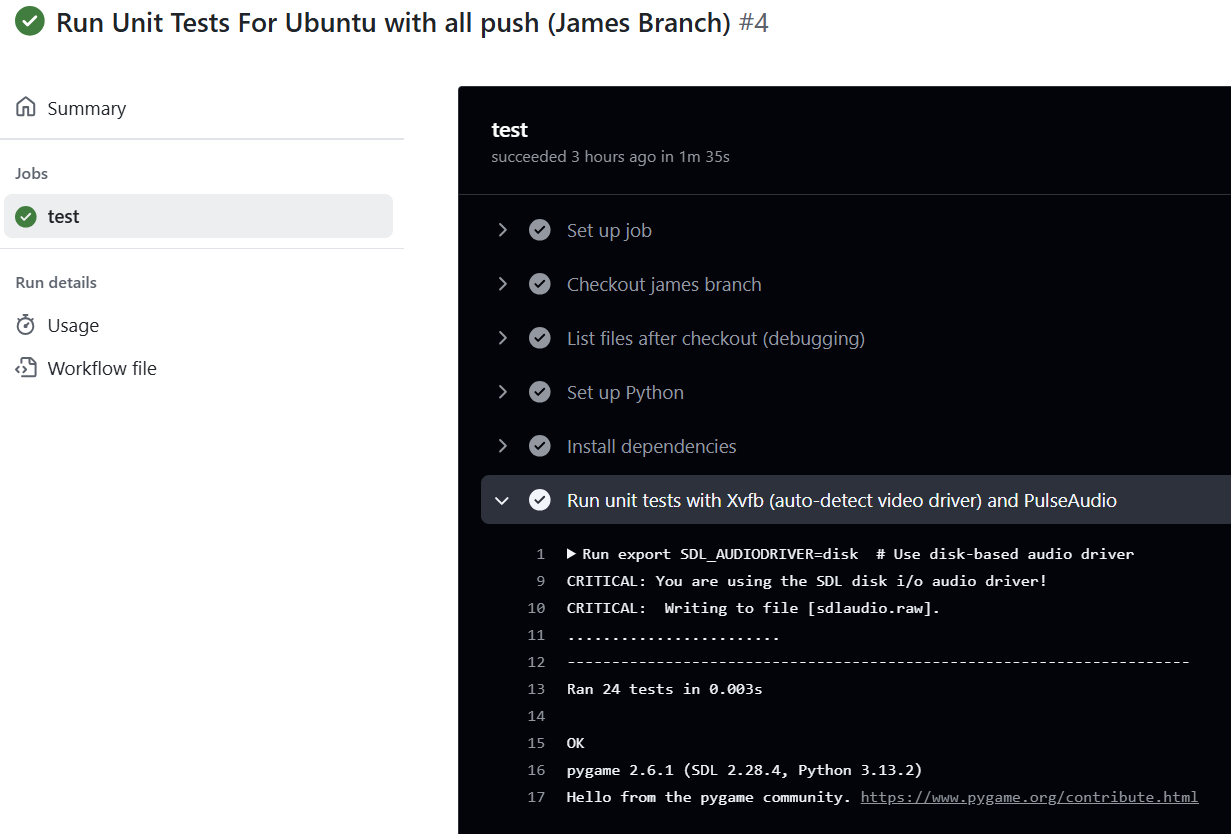
****

****

****

****

****

****

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

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. This is the Unit 4 approach which is different than Unit 5. I amplacing this here so that you know the transition form Unit 4 to Unit 5. 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.

Below is a test of our unit testing approach from unit 4:

| **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 |

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

# **14. APPENDIX E:**

**Git (GitHub Version Control):**

**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.
  + **Note: 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:**
    - 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.
    - Example commit message: Added game over condition for Tic-Tac-Toe.
  + **Push Changes to the Remote Repository:**
    - Once a developer has completed their changes locally, they will push the branch to the remote repository (e.g., GitHub).
  + **Pull Request (PR) for Code Review:**
    - After pushing the changes, the developer will create a Pull Request (PR) in GitHub to merge their feature branch into the main branch.
    - The PR will include a description of the changes made and any relevant context, making it easy for reviewers to understand what has been implemented.
  + **Code Review and Approval:**
    - **Javon**, as the project lead or reviewer, will review the PR. Javon will check for code quality, logic, functionality, and any potential bugs.
    - If the code meets the standards, Javon (Position of Developer but controls the main git Files) will approve the PR and merge it into the main branch. If any issues are found, Javon may provide feedback for necessary changes.
    - Example of feedback: Please update the comments in your code for clarity and refactor the score calculation logic.
  + **Merge and Update Main Branch:**
    - Once the PR is approved, the feature branch will be merged into the main branch. 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.

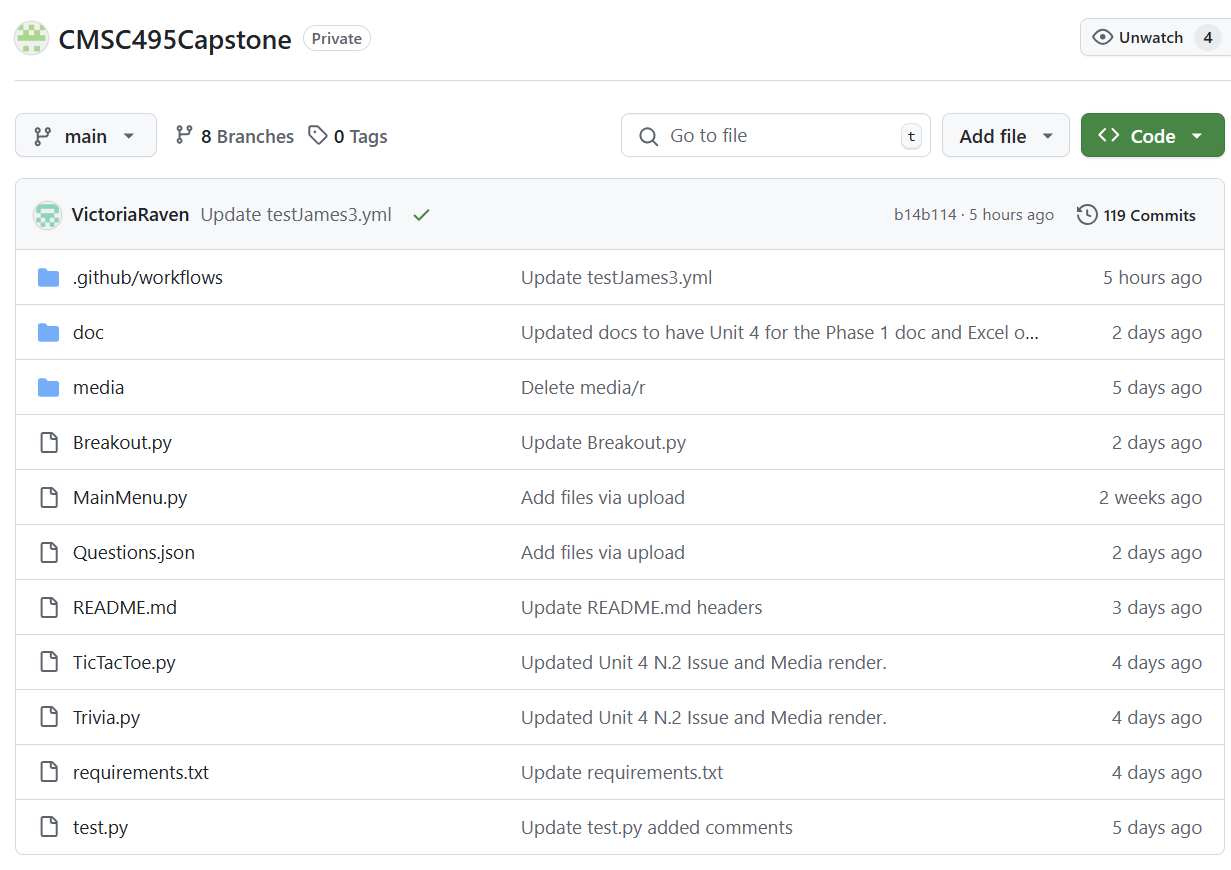
**Explanation:**

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:**

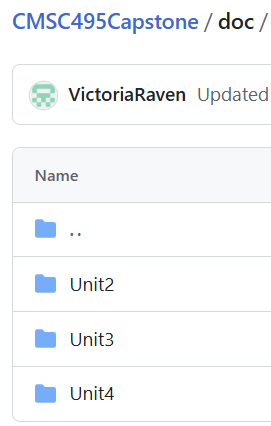
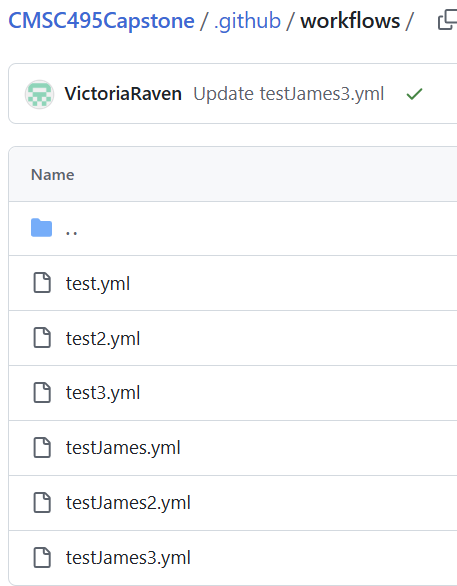
1. Feature Development: Each team member worked on a separate feature within their branches
2. Review and Commit: Before commits are made, each team member uploads their changes to their branch. The Git Lead then pulls each member's changes and tests them on the local environment. Once all code is confirmed 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:**

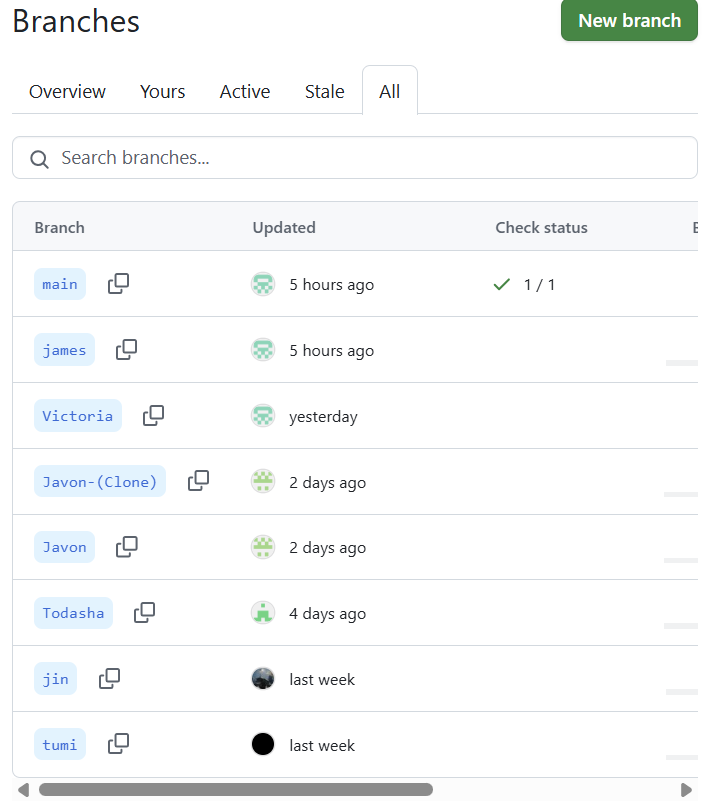


Media files and doc files and .github/workflow files:

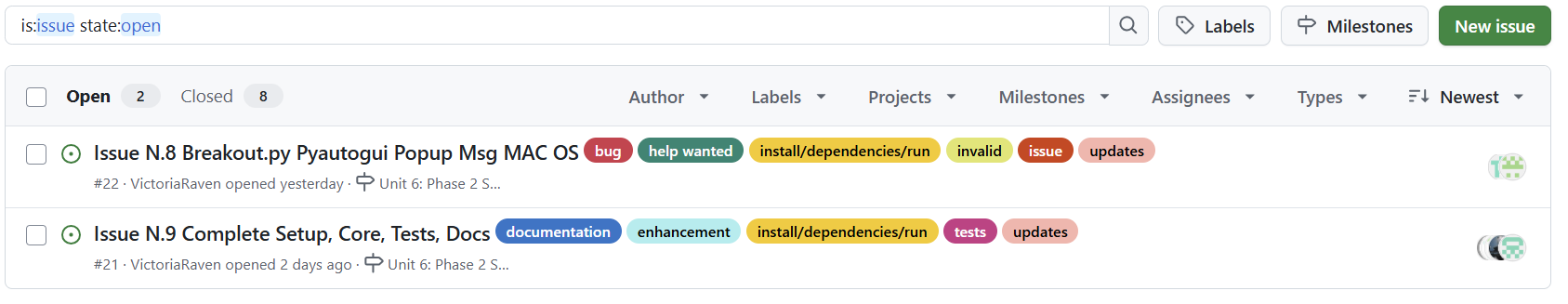
A screenshot of a computer

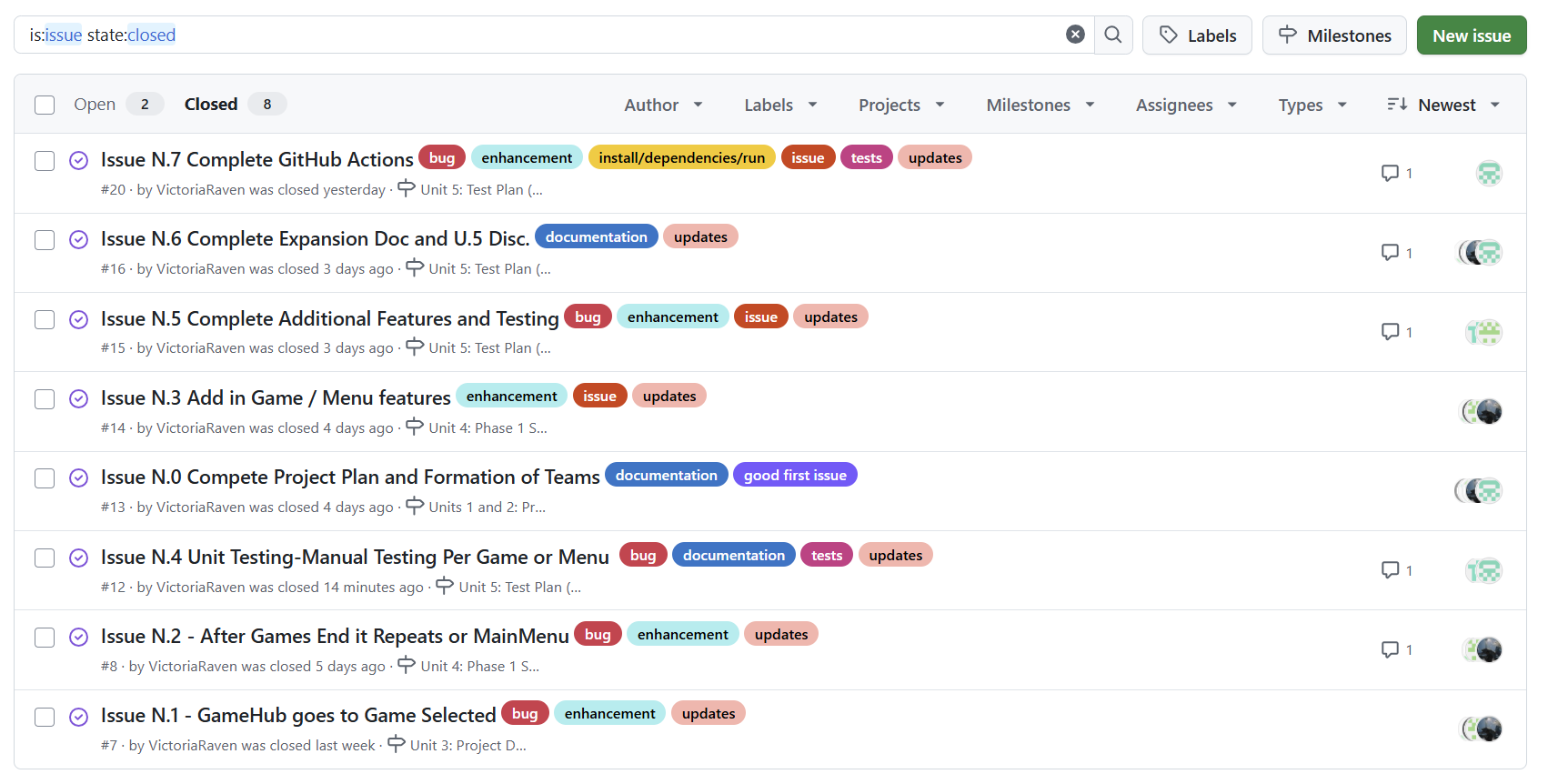
AI-generated content may be incorrect.  

We have a total of 8 branches:

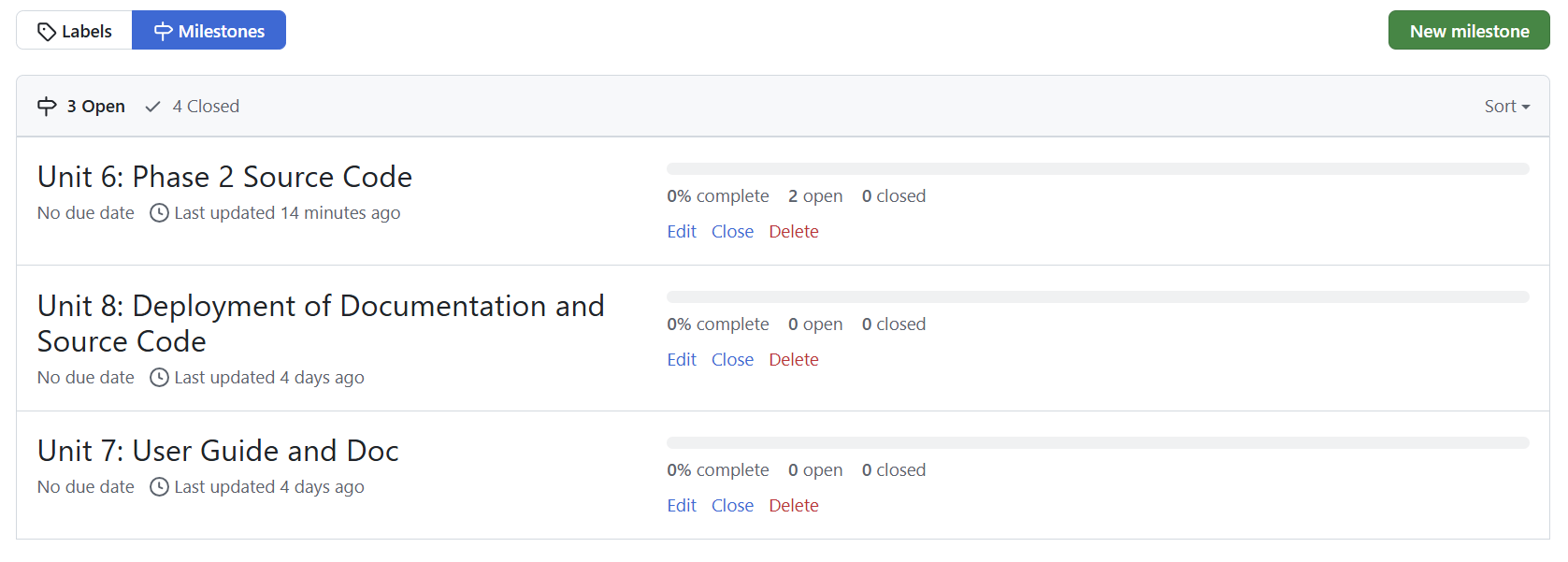


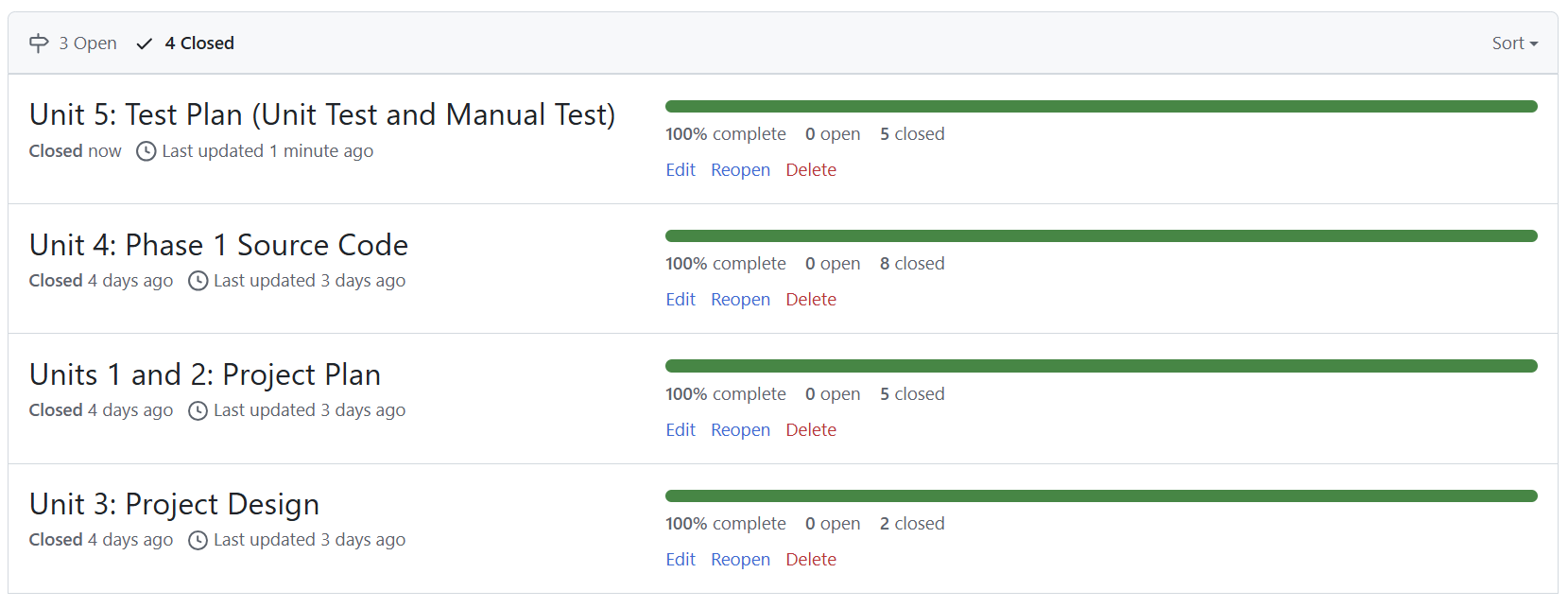
These are All our issues both Open and Closed:



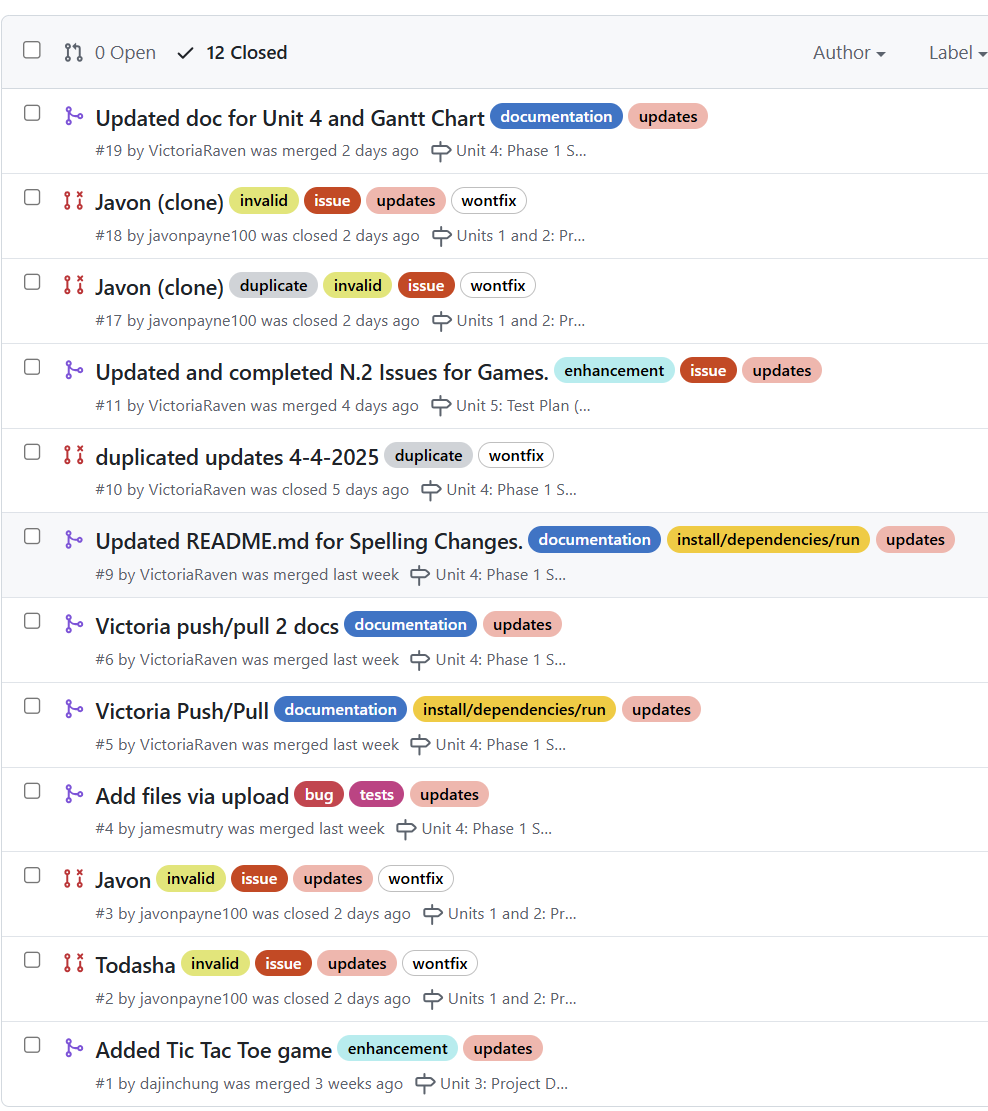


These are our miles stones (open and closed) (this was updated on 4/11/2025):

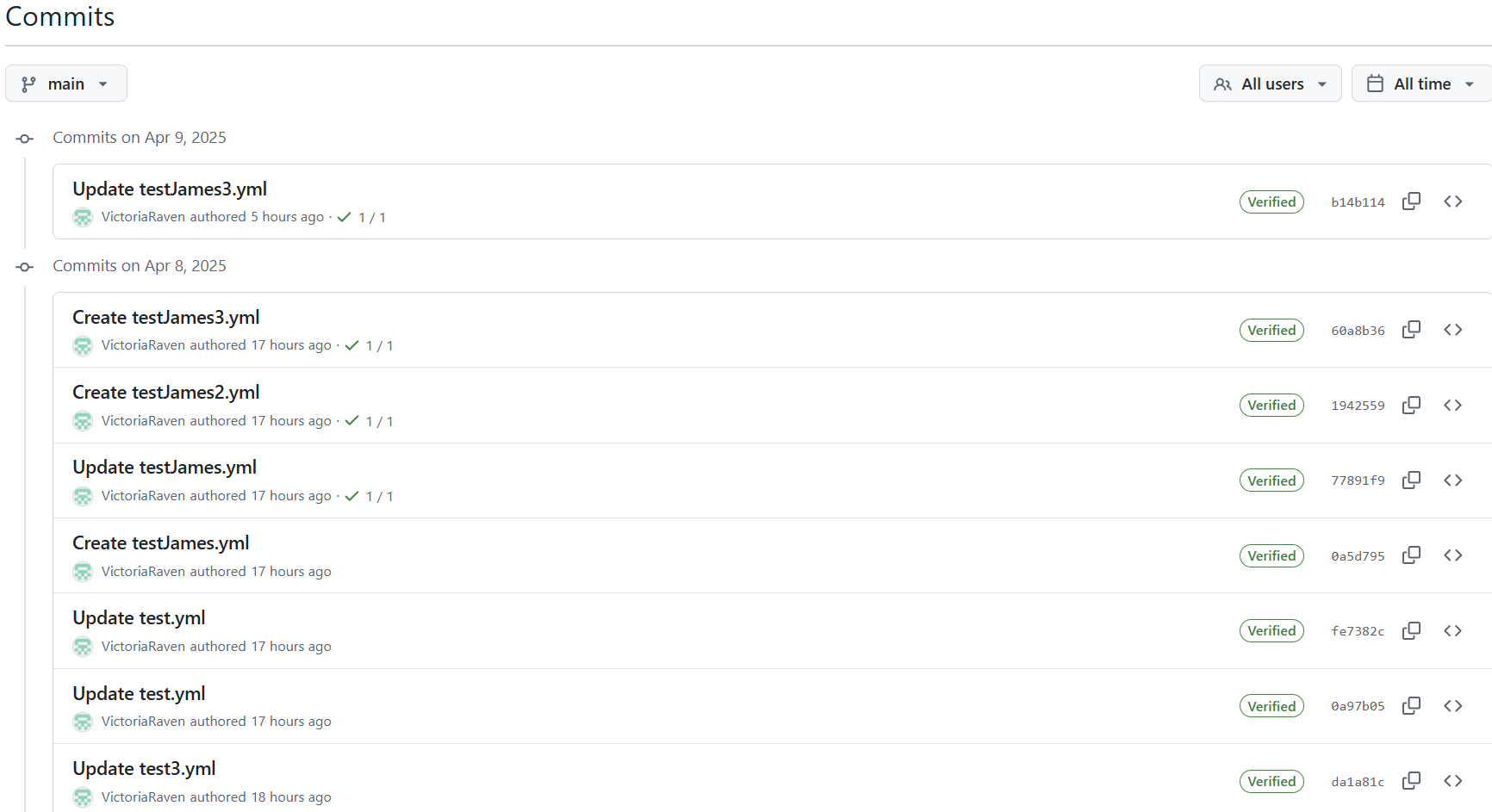


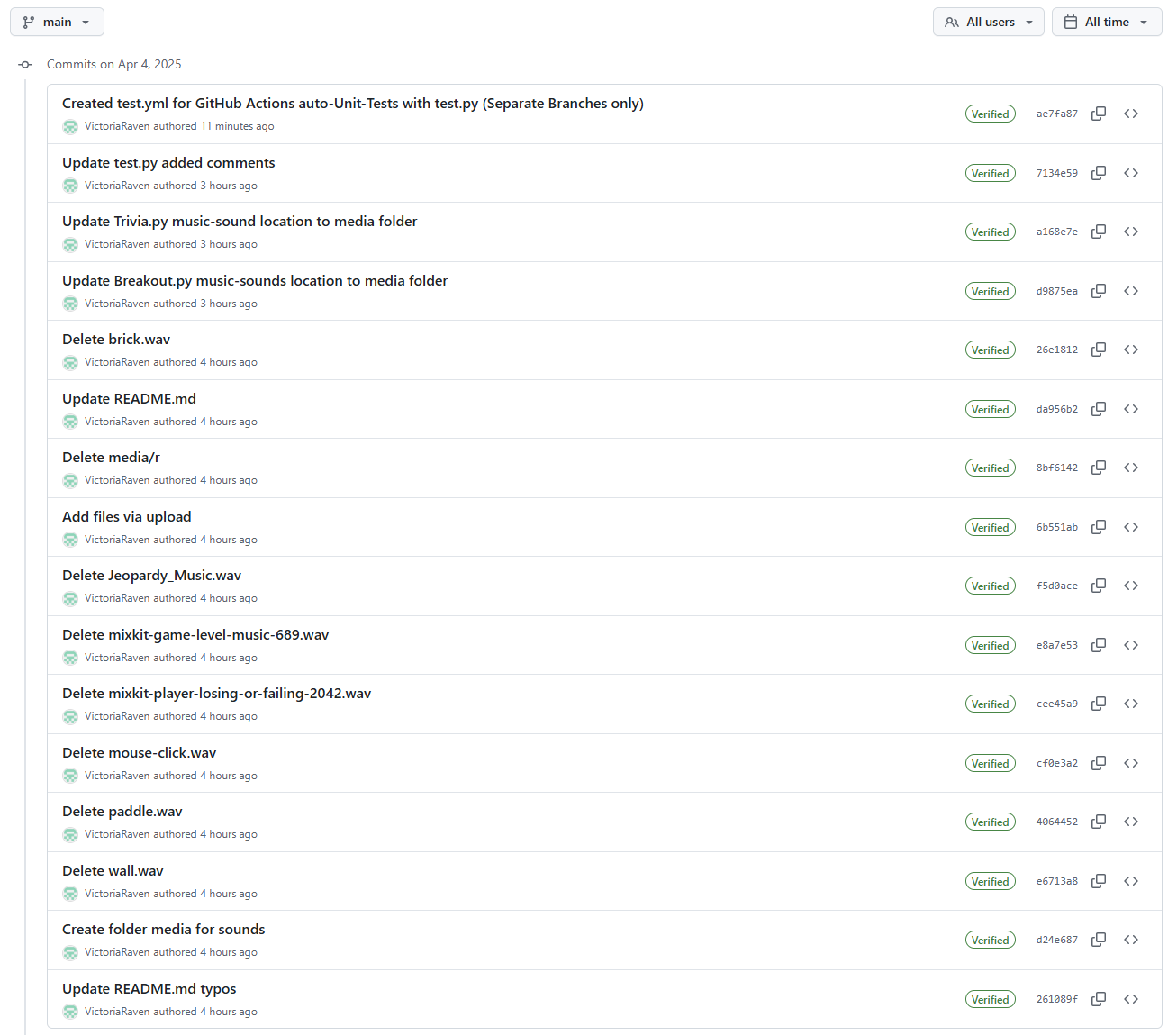


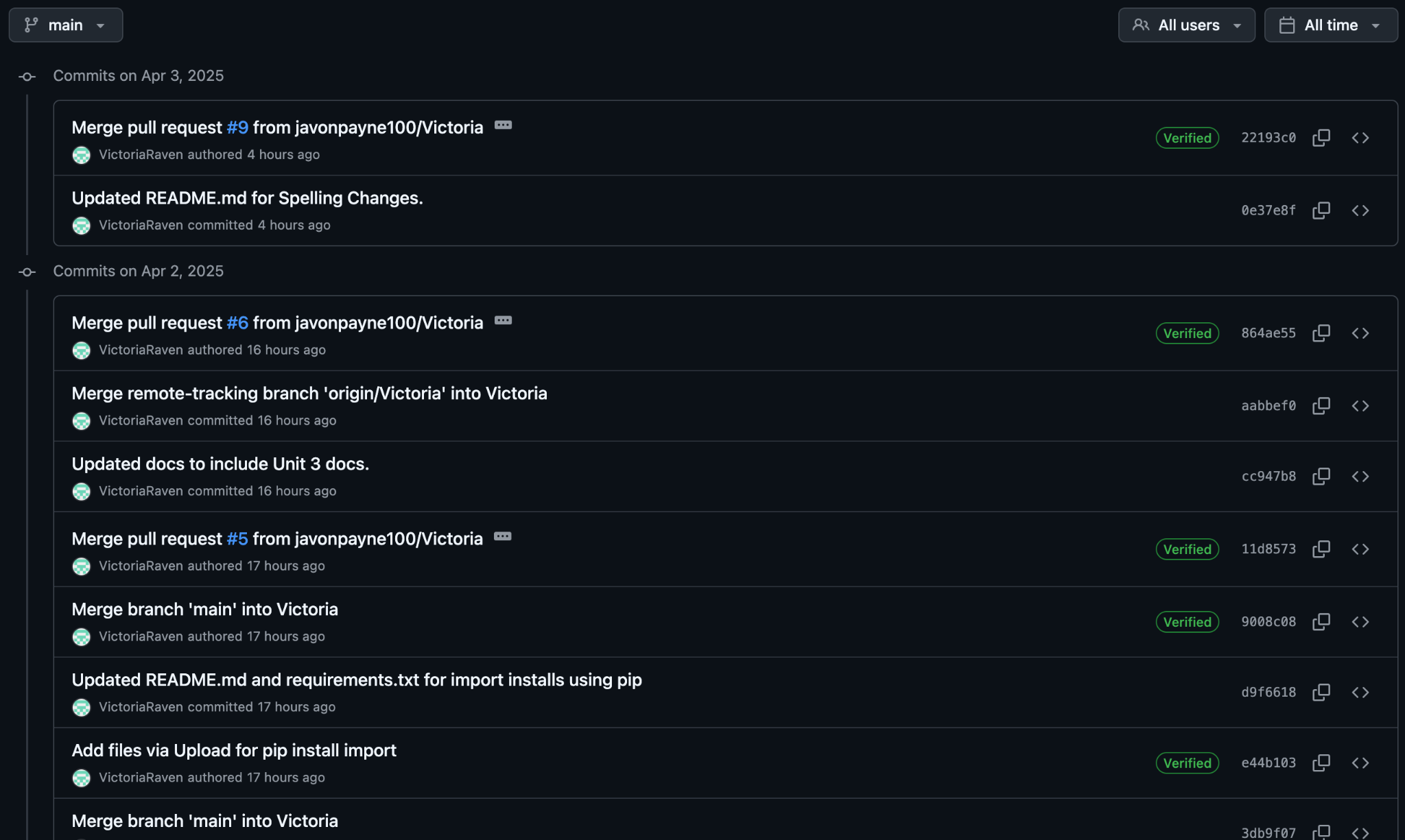
Here is our pull requests to merge onto the main using the branches:

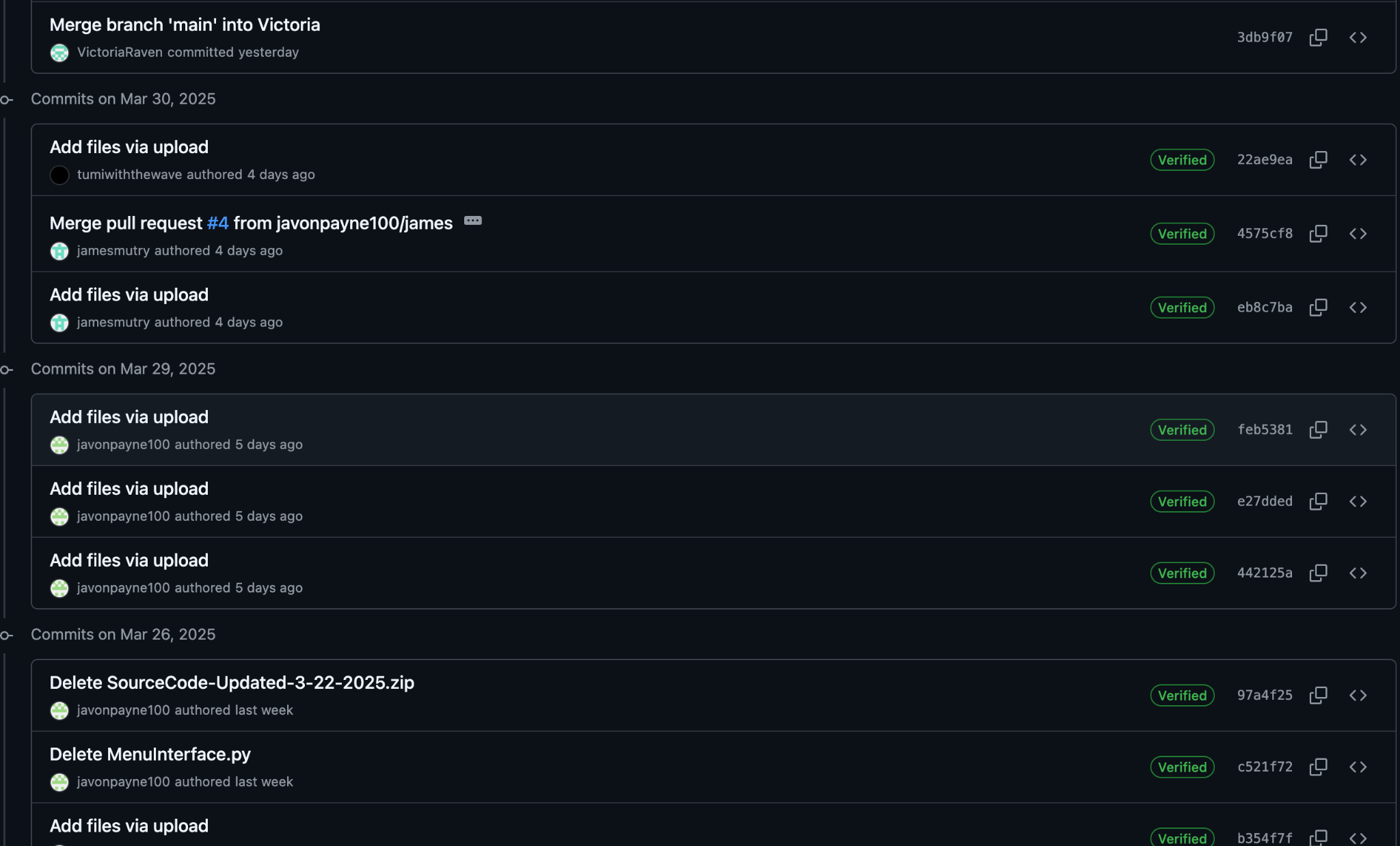


We have a total of 119 Commits (I will not show all screenshots as it is quite long; only main parts):











# **14. References**

Amazon Web Services. (n.d.). *What is unit testing?* Amazon Web Services, Inc. <https://aws.amazon.com/what-is/unit-testing/#:~:text=Unit%20testing%20is%20the%20process,test%20for%20each%20code%20unit>.

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>

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

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

GeeksforGeeks. (2024). *Software engineering: Differences between coupling and cohesion*. GeeksforGeeks. [https://www.geeksforgeeks.org/software-engineering-differences-between-coupling-and-cohesion/?ref=oin\_asr1](https://www.geeksforgeeks.org/software-engineering-differences-between-coupling-and-cohesion/?ref=oin_asr1&form=MG0AV3)

GeeksforGeeks. (2024). *Difference between Function-Oriented Design and Object-Oriented Design*. GeeksforGeeks. [https://www.geeksforgeeks.org/difference-between-function-oriented-design-and-object-oriented-design/](https://www.geeksforgeeks.org/difference-between-function-oriented-design-and-object-oriented-design/?form=MG0AV3)

GeeksforGeeks. (2024). *Object-Oriented Design (OOP)*. GeeksforGeeks. <https://www.geeksforgeeks.org/oops-object-oriented-design/?ref=oin_asr7>

GeeksforGeeks. (2024). *Functions decomposition in software engineering*. GeeksforGeeks. [https://www.geeksforgeeks.org/functions-decomposition-in-software-engineering/](https://www.geeksforgeeks.org/functions-decomposition-in-software-engineering/?form=MG0AV3)

GeeksforGeeks. (2024). *What is a Project Plan*. GeeksforGeeks. <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*. Retrieved March 31, 2025, from <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>

Python Software Foundation. (n.d.). Python 3 documentation. Python. <https://docs.python.org/3/>

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>

Stellman, A., & Greene, J. (2005). Software project planning. *Applied software project management* (Part I, Chapter 8). O'Reilly Media. [https://go.oreilly.com/umgc/https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch08.html](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. Open Library Textbook. <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*. UMGC. <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*. Unity. <https://unity.com/how-to/testing-and-quality-assurance-tips-unity-projects>  
Williams, H., & Zhang, K. (2020). *Improving unit testing frameworks for Python in software engineering*. *IEEE Software Engineering Journal*, 42(4), 135-145. <https://doi.org/10.1109/SEJ.2020.3078015>