Phase 1:

Source Code

UMGC CMSC 495

Python Game Hub

Lee, Victoria (Member 1)

Mutry, James (Member 2)

Foster, Todasha (Member 3)

Chung, Dajin (Member 4)

Payne, Javon (Member 5)

Ipaye, Oluwatumininu (Member 6)

Table of Contents

[**1. Introduction 3**](#_heading=h.8mc06n9bvdd3)

[**2. Objective 3**](#_heading=h.odwjil74txou)

[**3. Software Architecture 6**](#_heading=h.m2qp29dr7jcw)

[**4. Implementation the Software 8**](#_heading=h.raqkpphy5zgd)

[**5. Test the Software 10**](#_heading=h.czkb4lmj4ygi)

[**3. Source Code (Phase 1) 12**](#_heading=h.c0o9ze87z3se)

[**4. Conclusion: 20**](#_heading=h.qgvgeuhbqaqu)

[**5. APPENDIX A: 22**](#_heading=h.1v1i6tge9dgk)

[**6. APPENDIX B: 27**](#_heading=h.zebs1zdt82nz)

[**7. APPENDIX C: 30**](#_heading=h.iqcq3tytjku2)

[**8. APPENDIX D: 31**](#_heading=h.3eixzrkr1ry1)

[**9. APPENDIX E: 39**](#_heading=h.9bld1nvd9d2h)

[**10. References 46**](#_heading=h.vsbsv1myrrd0)

# **Introduction**

This document outlines the Phase 1 deliverables for our CMSC Capstone project: a multi-game application built using Python and Pygame. The project integrates various mini-games, Tic-Tac-Toe, Trivia, and Breakout. All within a unified and interactive graphical user interface.

We are adopting the Waterfall model for our software development lifecycle. This methodology structures the project into clearly defined, sequential phases: Requirements Analysis, System Design, Implementation, Testing, and Maintenance. In keeping with this model, each phase will be completed and reviewed before proceeding to the next.

Phase 1 focuses on laying a strong foundation through detailed requirements gathering and initial design and prototyping. It also includes the development of early functional modules such as the main menu interface and working versions of the games. The components developed in this phase provide a solid starting point for future enhancements and integration.

The goal of this phase is to ensure alignment between the project’s objectives and its technical execution strategy, setting the stage for reliable and maintainable development in the subsequent phases.

# **Objective**

**Project Purpose:** The goal of this project is to develop a Python-based Game Hub, a centralized platform that hosts multiple simple and engaging games. The hub will provide an interactive user interface, allowing users to select and play different games within a single application. The games included will be Break-Out, Tic-tac Toe, and Trivia. To track the progress of our coding updates, we will also integrate Git for version control, tracking, collaboration, and future enhancements. To track the testing in the development and integration portion, we will use the concept of unit testing to ensure that all the components meet the requirements.

**Project Deliverables:** The deliverables for the Python Game Hub project include a fully functional and interactive game hub application that serves as a centralized platform for multiple mini games. The application will feature a main menu for game selection, allowing users to navigate between different games. Each game will be developed as a separate modular component, creating easy maintenance and potential expansion in the future. The project will also include a graphical user interface for an easy-to-navigate user experience, built mainly using Pygame. If applicable, a lightweight SQLite database will be integrated to store user profiles, high scores, and game progress. The project’s source code repository will be hosted on GitHub or another version control platform for version tracking, collaboration, and future enhancements. There will also be a process to test the components through unit testing as we make subtle changes to the project.

**Project Requirements:**

**Functional Requirements:**

* Games must be fully playable with clear rules.
* Tracking Winners, Losers, Ties, High Scores, and Lives for one or more of the games.
* Optionally, user profiles must save game progress and high scores.
* To track progress on each coding section, we will use Git for Version Control.
* To track the progress of the games/UI Gam Hub testing section, we will use the concept of unit testing to ensure that each functional requirement is working during debug processing.
* **Note: Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process**

**Non-Functional Requirements:**

* Performance: The games must load within 3 seconds and run without noticeable lag.
* Score Board and lives: It should show the winner, loser, tie, the lives, and the scoreboard. If not, then the coder must debug.
* Usability: The user interface must be simple and intuitive, with a responsive design
* To track progress on each coding section, we will use Git for Version Control.
* To track the progress of the games/UI Gam Hub testing section, we will use the concept of unit testing to ensure that each non-functional requirement is working during debug processing.
* **Note: Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process**

**Testing/Debugging Requirements:**

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

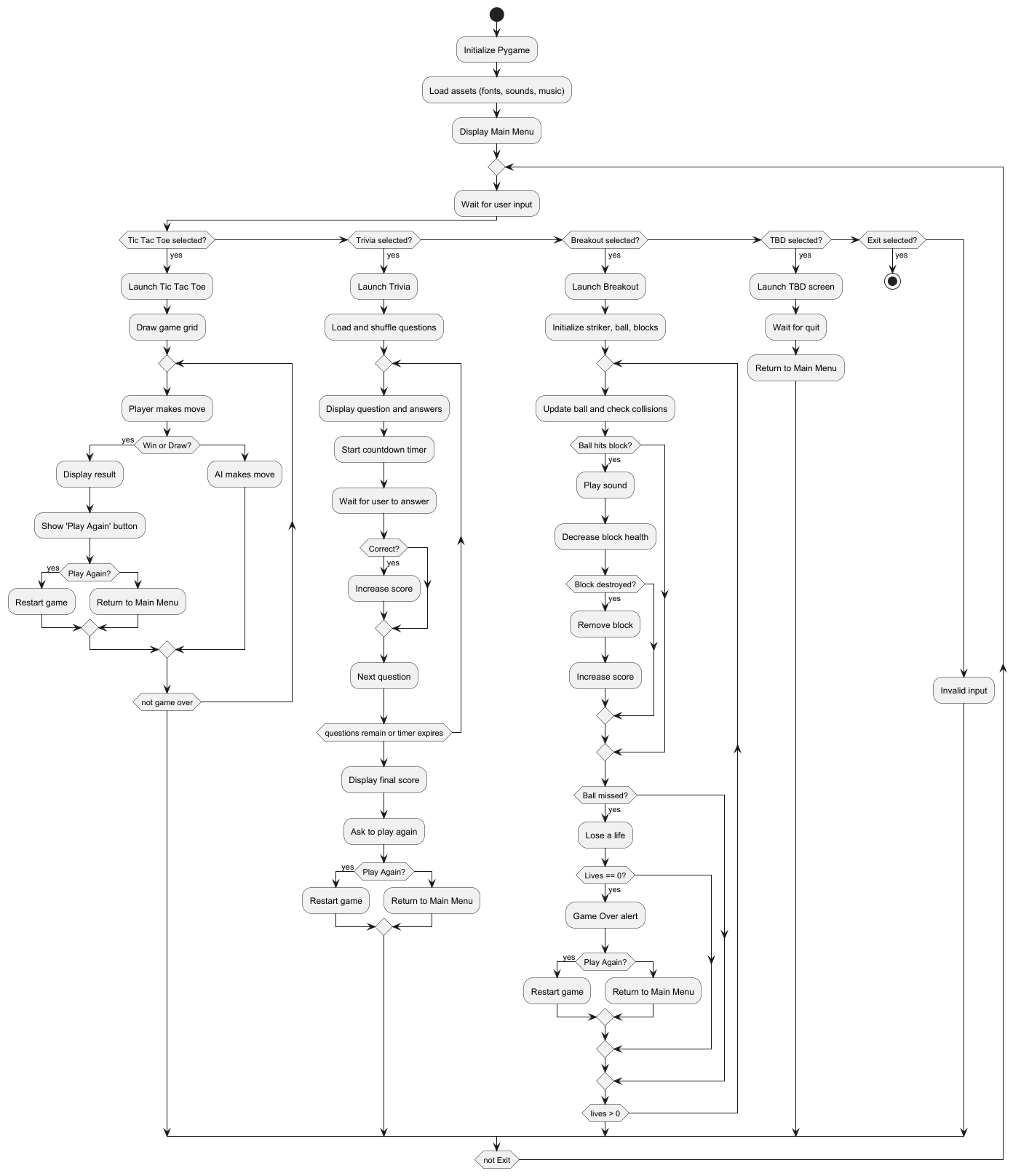
**Version Control with Git:**

* **Git as Version Control:**
  + Git will be used to track the progress of the project, manage code changes, and ensure collaboration between developers. Every change made to the codebase will be tracked with commit messages, providing a clear history of the project's development.
  + **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.

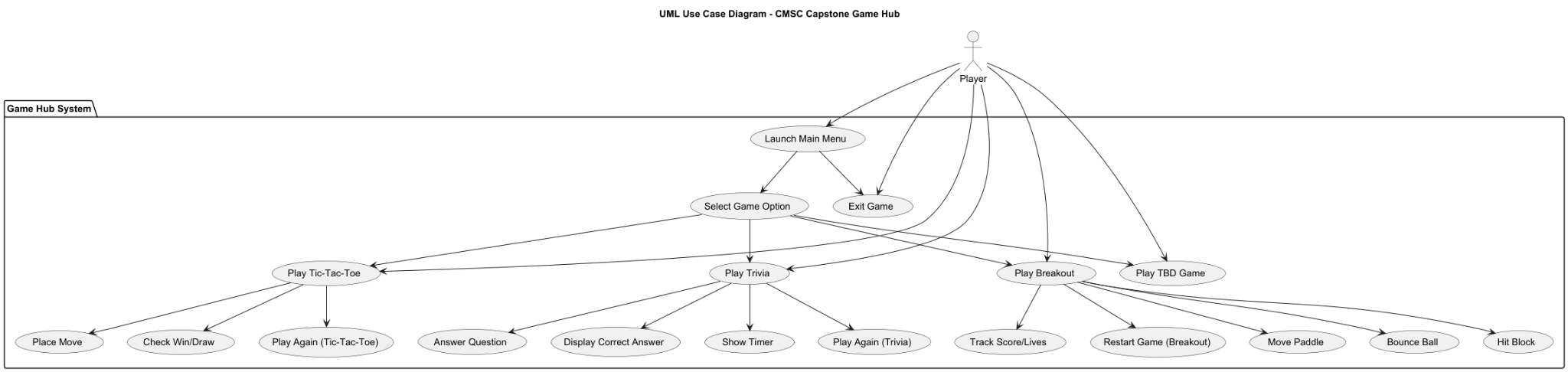
**Process:** This team group will be following the SDLC process model to help develop and deploy the Python Game Hub. This is a waterfall methodology that allows the team to plan in the early stages to prevent major design flaws from developing and below is the SDLC (GeeksforGeeks, n.d.) According to 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. This is particularly suited to projects where the requirements are well-defined upfront and are unlikely to change during development. 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.Waterfall is Suitable for This Project because ofthe features and games (Tic-Tac-Toe, Breakout, Trivia) are clearly defined from the outset. Given the project’s focused nature, the requirements are unlikely to change once the design is finalized. Therefore, Waterfall provides clear deadlines and checkpoints for each stage, ensuring systematic development of the project. **Note:** **Refer to APPENDIX A for an update on our milestones and Gantt chart, and APPENDIX B for the contribution report. And Refer to APPENDIX C for the agreement for delays/push backs.**



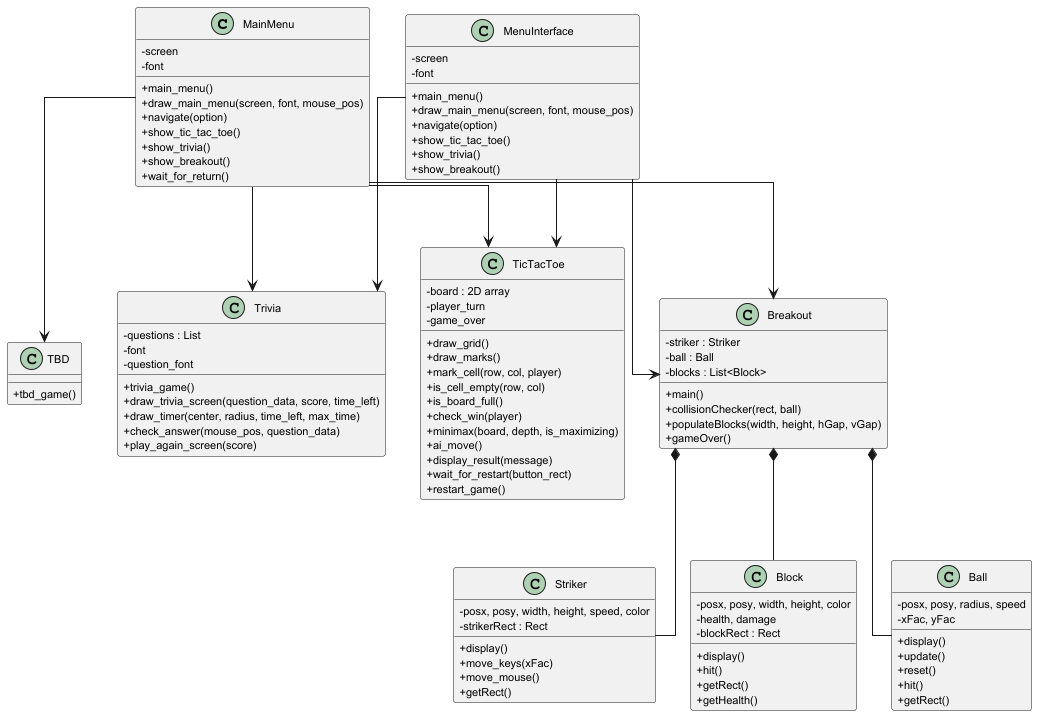
# **Software Architecture**

**Use Case Diagram:** The architecture of this software system was designed using object-oriented principles and modular design, ensuring each game component functions independently while integrating seamlessly through a central interface. The use case diagram illustrates the primary interactions: the user launches the application, navigates the main menu, selects a game, and then engages with that game’s specific mechanics, whether playing against an AI (Tic Tac Toe), answering timed questions (Trivia), or clearing blocks (Breakout). The system responds with real-time feedback, score tracking, and sound effects.

**Activity Diagram:** The activity diagram provides a step-by-step breakdown of user flow. It begins with the application startup, followed by main menu interaction. Based on the selected game, control flows into the respective module. Each module contains its own game loop, event handler, rendering system, and win/loss conditions. Once the game concludes, the user is either prompted to restart or return to the main menu.



**Class Diagram:** The class diagram defines the system’s key components and their relationships. The main interface oversees transitions between modules. Each game, Tic Tac Toe, Trivia, and Breakout, contains dedicated classes such as Board, AI, and Player in Tic Tac Toe, QuestionManager and Timer in Trivia, and Ball, Striker, and Blockin Breakout. These classes encapsulate state and behavior, promoting encapsulation and reusability. Audio and graphical elements are managed through pygame, with separate media assets for background music and sound effects.



# **Implementation the Software**

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

**Core Functionalities:**

**Implementation:** The implementation phase involved building a multi-game desktop application in Python using the Pygame library, which provides robust support for real-time graphics, audio playback, and user input handling. The project follows a modular design, where each mini-game—Tic Tac Toe, Trivia, and Breakout—is implemented as a standalone Python module and integrated into the application through a centralized menu system. **Note:** **Refer to APPENDIX A for an update on our milestones and Gantt chart, and APPENDIX B for the contribution report. And Refer to APPENDIX C for the agreement for delays/push backs.**

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

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

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

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

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

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

**Note:** **Refer to OBJECTIVES to see the Git and Unit Testing in the implementation process. It was not included here due to repetition, but there is a brief mention of it.**

# **Test the Software**

**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 correctness of answer indexing. The Breakout tests include health reduction of blocks, ball reset logic, paddle boundaries, and collision detection between game elements.

**Compare/Contrast Testing Methods:** 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. This decision was made due to the fact that most of my team members are not familiar with proper Git usage, and to avoid potential complications or errors that could result from improper integration into the GitHub repository. 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 the import eliminates the difficulties and adapts to each of the team member's skill sets effectively making the project's progress complete faster. By using unittest, we were able to directly run unit tests on the game’s Python files without modifying the Git workflow, ensuring smoother collaboration and minimizing the risk of problematic changes being committed. In contrast, GitHub Actions automates the testing process with each push, which ensures tests are always run as part of the development cycle. However, it requires team members to be fluent and comprehend Git workflow and CI/CD configuration, which can be difficult for teams unfamiliar with these tools. Our team will attempt to try out the Git Actions automatics unit test pushes ***only*** on the ***separate branches*** during ***unit 5 of the GitHub Actions automated Unit Tests*** if we have time. So far we have made the ***test.yml*** to demonstrate it as it is the other option.

**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 will begin with unit 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 prior to 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 to ensure that each part is tested and debugged correctly.

Below is the screenshot of the **test.py** with import unittest running and clearing its test.



Phase 1 Source Report - Last Update: 04 APR 2025

CMSC495

Capstone in Computer Science (2252)

**test.py:** The link below renders the code to test.py. This file contains all the Unit Testing code and the 13 ways of testing our application using the import unittest. Refer to APPENDIX E for the copied and pasted code version.

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

**test.yml:** This link below renders the code to the test.yml file which uses the test.py to automate the unit tests for our application. This is **ONLY** the layout and is not tested yet if it works since we wanted to use the manual old style version. Use test.py for the real method to test it. This is included since it was mentioned in the Unit 4 feedback to try out. This is continued in unit 5.

<https://gist.github.com/VictoriaRaven/8ce8069252ec46b6b28271c3f65e6edb>

* **Unit Testing:**
  + **Testing Individual Game Logic:** Each game (Break-Out, Tic-Tac-Toe, and Trivia) will undergo unit testing to verify that the core game mechanics function correctly. For example, testing game loops, score updates, and win/loss conditions.
  + **Testing User Interface Components:** UI elements like buttons, menus, and score displays will be unit-tested to ensure they are interactive, responsive, and properly linked to their respective game functions.
  + **High Scores:** If user profiles and high scores are implemented, we will test if the game progress and scores are correctly saved and retrieved from the database.
  + **Debugging Lives and Scoreboard Display:** Test to ensure that the number of lives and scores are being correctly tracked, displayed, and updated during gameplay.
  + **Error Handling and Robustness:** Ensure that the game properly handles unexpected inputs, errors, or edge cases, providing a smooth user experience even when something goes wrong.
  + **Note: Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process**
* **Debugging Process:**
  + **Debugging Each Game Module:** As each game is tested, any issues will be identified and debugged using systematic methods to ensure that the games operate seamlessly.
  + **UI Debugging:** Thorough debugging of the graphical user interface will be conducted to ensure that the design is intuitive and responsive across different screen sizes and devices.
  + **Performance Testing:** Ensure that all games load in under 3 seconds and perform without lag, especially on lower-end devices.
  + **Cross-Platform Compatibility:** Testing on different platforms to ensure the application runs without issues across all operating systems.
  + **Note: Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process**

# **Source Code (Phase 1)**

The following is the Phase I source code for a Python Game Hub application:

Phase 1 Source Report - Last Update: 03 APR 2025

CMSC495

Capstone in Computer Science (2252)

**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/VictoriaRaven/cde9d58550020450900cd8742dbe9b1e>

**requirements.txt:** The link 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 doens’t work the user must remove all but keep pygame, pyautogui, and numpy.

**Code:**

pygame

pyautogui

sys

numpy

random

math

time

json

unittest

Phase 1 Source Report - Last Update: 03 APR 2025

CMSC495

Capstone in Computer Science (2252)

**test.py:** The link below renders the code to test.py. This file contains all the Unit Testing code and the 13 ways of testing our application using the import unittest. Refer to APPENDIX E for the copied and pasted code version.

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

**test.yml:** This link below renders the code to the test.yml file which uses the test.py to make GitHub Action to automate the unit tests for our application. This is **ONLY** the layout and is not tested yet if it works since we wanted to use the manual old style version. Use test.py for the real method to test it. This is included since it was mentioned in the Unit 4 feedback to try out. This is continued in unit 5.

<https://gist.github.com/VictoriaRaven/8ce8069252ec46b6b28271c3f65e6edb>

Phase 1 Source Report - Last Update: 03 APR 2025 10:45am PST

CMSC495

Capstone in Computer Science (2252)

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

<https://gist.github.com/javonpayne100/90c47e66c31f775af7c265b90a635107>

Phase 1 Source Report - Last Update: 03 APR 2025 10:45am PST

CMSC495

Capstone in Computer Science (2252)

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

<https://gist.github.com/VictoriaRaven/748b45bb1ab842413553637a27eb147a>

Phase 1 Source Report - Last Update: 03 APR 2025 10:45am PST

CMSC495

Capstone in Computer Science (2252)

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

<https://gist.github.com/javonpayne100/6a4bf8a1df71437bcc28f42e591abc7b>

Phase 1 Source Report - Last Update: 03 APR 2025

CMSC495

Capstone in Computer Science (2252)

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

<https://gist.github.com/VictoriaRaven/d7ada730088300ee7ed25d5d12a31b9d>

Phase 1 Source Report - Last Update: 03 APR 2025

CMSC495

Capstone in Computer Science (2252)

**Breakout.py:** The link below renders the code to Breakout.py. The Breakout file is where players can play the Breakout game. There are several different features to this game which includes counting scores, lives, if you hit more than one block it will make it disappear, and other graphic features. Below is the link. If it does not work, the code is below (copy and pasted).

<https://gist.github.com/VictoriaRaven/93f0d66e311319dc76192f4c2c32dd7b>

**Breakout.py:**

import pygame

import random

import pyautogui

pygame.init()

WIDTH, HEIGHT = 750, 450

# Colors

BLACK = (0, 0, 0)

WHITE = (255, 255, 255)

GREEN = (51, 204, 51)

RED = (204, 0, 0)

BLUE = (0, 0, 255)

font = pygame.font.Font(None, 25)

screen = pygame.display.set\_mode((WIDTH, HEIGHT))

pygame.display.set\_caption("Game Hub: Breakout")

# to control the frame rate/speed of the game

clock = pygame.time.Clock()

FPS = 70

# Initialize mixer for sound

pygame.mixer.init()

# Set volume (ensure the volume is not too low)

pygame.mixer.music.set\_volume(1.0)

# Load sounds with error handling

try:

wall\_sound = pygame.mixer.Sound("media/wall.wav")

paddle\_sound = pygame.mixer.Sound("media/paddle.wav")

brick\_sound = pygame.mixer.Sound("media/paddle.wav")

losing\_sound = pygame.mixer.Sound("media/mixkit-player-losing-or-failing-2042.wav")

except pygame.error as e:

print("Error loading sound:", e)

# Striker class

class Striker:

def \_\_init\_\_(self, posx, posy, width, height, speed, color):

self.posx, self.posy = posx, posy

self.width, self.height = width, height

self.speed = speed

self.color = color

# The rect variable is used to handle the placement

# and the collisions of the object

self.strikerRect = pygame.Rect(self.posx, self.posy, self.width, self.height)

self.striker = pygame.draw.rect(screen, self.color, self.strikerRect)

# Used to render the object on the screen

def display(self):

self.striker = pygame.draw.rect(screen, self.color, self.strikerRect)

# Used to update the state of the object

def update(self, xFac):

self.posx += self.speed \* xFac

# Restricting the striker to be in between the left and right edges of the screen

if self.posx <= 0:

self.posx = 0

elif self.posx + self.width >= WIDTH:

self.posx = WIDTH - self.width

self.strikerRect = pygame.Rect(self.posx, self.posy, self.width, self.height)

# Returns the rect of the object

def getRect(self):

return self.strikerRect

# Block Class

class Block:

def \_\_init\_\_(self, posx, posy, width, height, color):

self.posx, self.posy = posx, posy

self.width, self.height = width, height

self.color = color

self.damage = 100

# The white blocks have the health of 200. So, the ball must hit it twice to break

if color == RED:

self.health = 300

elif color == BLUE:

self.health = 200

else:

self.health = 100

# The rect variable is used to handle the placement and the collisions of the object

self.blockRect = pygame.Rect(self.posx, self.posy, self.width, self.height)

self.block = pygame.draw.rect(screen, self.color, self.blockRect)

# Used to render the object on the screen if and only if its health is greater than 0

def display(self):

if self.health > 0:

self.brick = pygame.draw.rect(screen, self.color, self.blockRect)

# Used to decrease the health of the block

def hit(self):

self.health -= self.damage

# Used to get the rect of the object

def getRect(self):

return self.blockRect

# Used to get the health of the object

def getHealth(self):

return self.health

# Ball Class

class Ball:

def \_\_init\_\_(self, posx, posy, radius, speed, color):

self.posx, self.posy = posx, posy

self.radius = radius

self.speed = speed

self.color = color

self.xFac, self.yFac = 1, 1

self.ball = pygame.draw.circle(screen, self.color, (self.posx, self.posy), self.radius)

# Used to display the object on the screen

def display(self):

self.ball = pygame.draw.circle(screen, self.color, (self.posx, self.posy), self.radius)

# Used to update the state of the object

def update(self):

self.posx += self.xFac \* self.speed

self.posy += self.yFac \* self.speed

# Reflecting the ball if it touches either of the vertical edges

if self.posx <= 0 or self.posx >= WIDTH:

self.xFac \*= -1

wall\_sound.play() # Play wall hit sound

# Reflection from the top most edge of the screen

if self.posy <= 0:

self.yFac \*= -1

# If the ball touches the bottom most edge of the screen, True value is returned

if self.posy >= HEIGHT:

return True

return False

# Resets the position of the ball

def reset(self):

self.posx = 0

self.posy = HEIGHT

self.xFac, self.yFac = 1, -1

# Used to change the direction along Y axis

def hit(self):

self.yFac \*= -1

paddle\_sound.play() # Play paddle hit sound

# Returns the rect of the ball

def getRect(self):

return self.ball

# Function used to check collisions between any two entities

def collisionChecker(rect, ball):

if pygame.Rect.colliderect(rect, ball):

return True

return False

# Function used to populate the blocks

def populateBlocks(blockWidth, blockHeight, horizontalGap, verticalGap):

listOfBlocks = []

for i in range(0, WIDTH, blockWidth + horizontalGap):

for j in range(0, HEIGHT // 2, blockHeight + verticalGap):

listOfBlocks.append(Block(i, j, blockWidth, blockHeight, random.choice([RED, BLUE, GREEN])))

return listOfBlocks

# Once all the lives are over, this function waits until exit or space bar is pressed and does the corresponding action

def gameOver():

gameOver = True

while gameOver:

# Event handling

for event in pygame.event.get():

if event.type == pygame.QUIT:

return False

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_SPACE:

return True

def main():

running = True

lives = 3

score = 0

scoreText = font.render("score", True, WHITE)

scoreTextRect = scoreText.get\_rect()

scoreTextRect.center = (20, HEIGHT - 10)

livesText = font.render("Lives", True, WHITE)

livesTextRect = livesText.get\_rect()

livesTextRect.center = (120, HEIGHT - 10)

striker = Striker(0, HEIGHT - 50, 100, 20, 10, WHITE)

strikerXFac = 0

ball = Ball(0, HEIGHT - 150, 7, 5, WHITE)

blockWidth, blockHeight = 40, 15

horizontalGap, verticalGap = 10, 10

listOfBlocks = populateBlocks(blockWidth, blockHeight, horizontalGap, verticalGap)

# Game loop

while running:

screen.fill(BLACK)

screen.blit(scoreText, scoreTextRect)

screen.blit(livesText, livesTextRect)

scoreText = font.render("Score : " + str(score), True, WHITE)

livesText = font.render("Lives : " + str(lives), True, WHITE)

# If all the blocks are destroyed, then we repopulate them

if not listOfBlocks:

listOfBlocks = populateBlocks(blockWidth, blockHeight, horizontalGap, verticalGap)

# All the lives are over. So, the gameOver() function is called

if lives <= 0:

pyautogui.alert('Press SPACE to restart or quit!', "Game Over!") # notification to instruct player

# Play the losing sound when game ends

losing\_sound.play()

running = gameOver()

while listOfBlocks:

listOfBlocks.pop(0)

lives = 3

score = 0

listOfBlocks = populateBlocks(blockWidth, blockHeight, horizontalGap, verticalGap)

# Event handling

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_LEFT:

strikerXFac = -1

if event.key == pygame.K\_RIGHT:

strikerXFac = 1

if event.type == pygame.KEYUP:

if event.key == pygame.K\_LEFT or event.key == pygame.K\_RIGHT:

strikerXFac = 0

# Collision check

if collisionChecker(striker.getRect(), ball.getRect()):

ball.hit()

for block in listOfBlocks:

if collisionChecker(block.getRect(), ball.getRect()):

ball.hit()

block.hit()

if block.getHealth() <= 0:

listOfBlocks.pop(listOfBlocks.index(block))

score += 5

brick\_sound.play() # Play brick hit sound

# Update

striker.update(strikerXFac)

lifeLost = ball.update()

# If the ball goes off the screen, a life is lost

if lifeLost:

lives -= 1

ball.reset()

# Display all blocks and entities

for block in listOfBlocks:

block.display()

ball.display()

striker.display()

pygame.display.update()

clock.tick(FPS)

pygame.quit()

if \_\_name\_\_ == "\_\_main\_\_":

main()

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

# **Conclusion:**

The Phase I source code is an important part of the Phase I project. It demonstrates our team’s understanding of software design, implementation, and testing. By following these instructions and using the example code as a template, we can write our Phase I source code that meets the requirements of the project plan and project design. The Git (version control) and unit testing will help out with the debugging and testing process. This will allow the team to keep track of the milestones, make sure that there are no delays, and help the team organize additional features into the game, such as adding other game features or even other ongoing changes. We are currently changing things, but we haven’t finished it and will complete it fully by the time unit 5 begins. To further explain the Git and Unit Testing process, the project will use Git for version control to manage code changes and facilitate collaboration among developers. Each developer will create feature branches for specific tasks, making incremental changes and committing them with descriptive messages. After completing their work, they will push their changes to the remote repository and create a pull request (PR) for code review. The project lead and testers will review and approve the PRs, merging them into the main branch once the code meets the required standards. Unit testing will ensure the core game mechanics, user interface, and features function correctly. Additionally, the debugging process will focus on identifying and fixing issues, fixing the additional features made in Units 4 and 5, testing the UI for responsiveness, ensuring the games load quickly, and confirming compatibility. Regular updates will be made to keep the main branch stable and up-to-date, and any merge conflicts will be resolved promptly to maintain consistency. 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.

# **APPENDIX A:**

A Large Table Gantt Chart repeat of Project Plan in Unit 2:

| 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  Peer Review 2 |
| 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  Peer Review 2 |
| 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:

Note: To see more, save the image or use CTRL + to zoom in.

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 **60% 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 (60% complete):**

* **Project Plan**: Fully completed, outlining key tasks, milestones, and deadlines.
* **Project Design**: All design elements and documentation are complete.
* **Phase 1 (Current Week)**: The development phase, where all initial features and functionalities were implemented, including core game functionality.

#### **Upcoming Tasks (Remaining 40%):**

1. **Testing Plan:**This includes modifying tests from unit 3, unit 4, and continuing the testing while finishing up additional features in the games.
2. **Phase 2**: This includes finishing the final feature, **Trivia.py**, which is part of the game suite for Unit 5.
3. **User Guide Documentation**: Unit 7 - The user guide is yet to be completed, which will provide comprehensive instructions for using the software.
4. **Deployment**: Unit 8 - 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.).

### **Feature Completion and Phase 1**

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 of the additional features | No remaining tasks.  -Updated buttons, UI, updated look, and transitions. | 100% |
| **TicTacToe.py** | Completed and fully implemented of the additional features | No remaining tasks.  -Updated AI/CPU/transition, music | 100% |
| **Breakout.py** | Completed and fully implemented of the additional features | No remaining tasks.  -Updated scores/lives, including transition, music | 100% |
| **Trivia.py** | Partially completed. | Finish the implementation for **Unit 5 not in unit 4**. Some testes are not implmented fully.  -Updates 1-2 player mode and also scores, music, and subject questions. | 70% |
| **Unit Testing (all games)** | Ongoing testing of all games. | Complete more tests of different ranges and concepts for all files in unit 5 after adding some in unit 4 | 100% |

### 

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

As we are currently in **Phase 1 Week**, the main focus is on ensuring the correctness and reliability of the implemented features and also starting the unittest and git version implmentaiton.

### 

### **Next Phases: Weeks 5-8**

* **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.
* **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.
* **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.
* **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.
* **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, 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: | 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 (Implmenting feautres of Mainmenu.py, TicTacToe.py, Breakout.py, Trivia.py; Implement Git Version Control, and start out Unit Tests) | 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)) | N/A % |
| **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** |  | **60% Complete** |

### **Total Progression**

The project is currently **60% 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 (Unit 4 and 5)**, ensuring all **Tests (Unit 5)** are completed, and producing the **Phase 2 (Unit 6)**, **User Guide (Unit 7)** and **final deployment documentation (Unit 8)**. By working together during weeks 6 to 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.

.

# **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 | Jin created base code for tic tac toe game ~1 hour | Chung, Dajin | ~1 hour |  |
| 3/16/2025 to 3/22/2025 | Worked on Breakout portion of the game hub. Todasha coded 1hr for sound.  Worked on tic tac toe game functions ~1 hour | Ipaye, Oluwatumininu;  Foster, Todasha.  Chung, Dajin | ~7 hour ~1 hour  ~1 hour | $0 |
| 3/16/2025 to 3/26/2025 | Worked on the Main Menu portion and Todasha coded 1 hr for sound (~1 hour).  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 hour | $0 |
| 3/16/2025 to 3/22/2025 | Worked on the Trivia portion. Todasha coded 1hr for sound.  Worked on tic tac toe functions/logic 1hr | Foster, Todasha;  Chung, Dajin; | ~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 | 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 created the media folder and adjusting on the other files to movie 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 hour)  Tested Links, updated it, and checked documentation updates (~ 1 hour) | Payne, Javon;  Lee, Victoria; | ~ 3 hours  ~ 1 hour | $0 |
|  |  |  |  | $0 |

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

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

# **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 2 to 4 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.

Acknowledgment:

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

# **APPENDIX D:**

**Unit Testing Process (Updates per Unit week):**

Below is a table that includes all aspects of 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 based on GeeksforGeeks (n.d.) and Tsui, F., Karam, O., & Bernal, B. (2014). **The Unit Test(actual code) file is: test.py. See it on GitHub Gists link.**

| **Category** | **Game** | **Description** | **Steps/Questions** | **Software Engineering Technique** | **Goal/Outcome** |
| --- | --- | --- | --- | --- | --- |
| Unit Testing: Core Logic | Break-Out | Test the core mechanics (ball movement, paddle interaction, and collision detection). | 1. Does the ball move correctly in all directions? 2. Does the ball bounce off the paddle and walls appropriately? 3. Are the bricks destroyed when hit by the ball? | Test-Driven Development (TDD) | Ensure the core mechanics of ball movement and collision detection work as expected. |
|  | Tic-Tac-Toe | Test the game logic, such as checking for win/loss conditions and player input. | 1. Does the game recognize a win condition for both X and O? 2. Is the game able to check for a tie? 3. Does the game reject invalid moves? | Unit Testing, Boundary Testing | Validate that win/loss conditions are properly evaluated and that invalid moves are correctly handled. |
|  | Trivia | Test the scoring system, timer, and question-answer logic. | 1. Is the score correctly updated after each question? 2. Does the timer function as expected for each question? 3. Are correct answers being tracked properly? | Functional Testing, Integration Testing | Ensure the trivia scoring system, timer, and question-answer interactions are working as intended. |
| Unit Testing: UI Components | Break-Out | Test the user interface, buttons, score displays, and feedback elements. | 1. Are the start, pause, and reset buttons functional? 2. Does the score update in real-time? 3. Are visual elements (e.g., bricks, paddle) drawn correctly? | UI Testing, Automated Testing | Ensure that the UI is responsive and all visual elements function correctly and update dynamically during gameplay. |
|  | Tic-Tac-Toe | Test UI for game board interactions, player turn indication, and reset functionality. | 1. Are the cells clickable and responsive? 2. Does the game board display the player's turn? 3. Is the reset button working correctly? | UI Testing, Usability Testing | Ensure the UI for player interaction, board display, and reset functions are intuitive and operational. |
|  | Trivia | Test UI components for question display, answer selection, and timer functionality. | 1. Are questions displayed correctly with proper formatting? 2. Is the timer visible and accurate? 3. Do the answer options update based on user selection? | UI Testing, Cross-Platform Testing | Ensure the trivia UI components are interactive, responsive, and function as expected across platforms. |
| Unit Testing: Data Handling | Break-Out | Test if user data such as scores and progress are saved and loaded. | 1. Is the user’s score saved after each game session? 2. Does the game load the saved score correctly on restart? 3. Is data integrity maintained between sessions? | Data Persistence Testing | Verify that the game correctly saves and retrieves user data, such as scores or game progress, and maintains data integrity. |
|  | Tic-Tac-Toe | Test saving user profiles, high scores, and game states (if applicable). | 1. Does the game store the highest score for each user? 2. Can the player’s progress be saved and loaded correctly? 3. Does the game track and display high scores? | Unit Testing, Data Integrity Testing | Ensure that user data such as profiles, high scores, and game progress are saved, retrieved, and maintained correctly. |
|  | Trivia | Test score saving and retrieval for users across sessions. | 1. Does the game store and retrieve high scores correctly? 2. Are all game session scores stored accurately? 3. Can user data (e.g., name, score) be retrieved? | Data Persistence, Automated Testing | Verify the accuracy of data storage and retrieval for scores, profiles, and progress in the trivia game. |
| Debugging Process | Break-Out | Debug the ball movement, paddle interactions, and brick destruction behavior. | 1. Does the ball correctly bounce off walls/paddle? 2. Are all brick collisions detected correctly? 3. Does the score update when bricks are destroyed? | Debugging, Issue Isolation | Resolve any issues related to ball movement, collision detection, and destruction logic to ensure smooth gameplay. |
|  | Tic-Tac-Toe | Debug game logic errors such as win condition detection and invalid move handling. | 1. Does the game properly detect when a player wins? 2. Are invalid moves correctly rejected? 3. Are tie conditions correctly recognized? | Debugging, Logical Error Fixing | Resolve logical issues in win/loss detection, input validation, and tie conditions. |
|  | Trivia | Debug the timer, scoring system, and question-answer logic. | 1. Does the timer reset after each question? 2. Are scores updating correctly after each correct/incorrect answer? 3. Are questions being loaded correctly? | Debugging, Issue Isolation | Resolve issues related to the timer, score tracking, and question-answer interactions to ensure game functionality. |
| Performance Testing | Break-Out | Test game loading speed, ball movement performance, and visual rendering speed. | 1. Does the game load within 3 seconds? 2. Are there any lags when the ball is in motion? 3. Are graphical elements rendering smoothly without delay? | Performance Testing, Load Testing | Ensure the game performs efficiently, with no significant delays in game loading or graphical rendering. |
|  | Tic-Tac-Toe | Test UI responsiveness and performance, especially with multiple player inputs. | 1. Does the game load and play smoothly on various devices? 2. Are inputs being registered without lag? 3. Does the game respond instantly to player actions? | Performance Testing, Usability Testing | Ensure that Tic-Tac-Toe performs well across different devices, with no performance issues in UI interaction or input processing. |
|  | Trivia | Test the responsiveness of the UI and performance under high traffic (e.g., multiple answers selected). | 1. Does the game handle multiple answers quickly without lag? 2. Is the timer accurate under various conditions? 3. Does the score system update instantly? | Stress Testing, Load Testing | Ensure that the trivia game performs well under various user interactions and handles multiple inputs without lag or errors. |
| Cross-Platform Testing | Break-Out | Ensure the game runs on different operating systems (Windows, Mac, Linux). | 1. Does the game launch on all supported platforms? 2. Are all visual and interactive elements working across different operating systems? 3. Is the performance consistent? | Cross-Platform Testing, Compatibility Testing | Ensure the game is compatible and functions properly across multiple operating systems and devices. |
|  | Tic-Tac-Toe | Test the game’s UI and functionality across different screen sizes and devices. | 1. Does the game render correctly on mobile and desktop? 2. Is the game playable on different screen resolutions? 3. Are all UI elements responsive and sized correctly? | Cross-Platform Testing, Responsive Design | Verify the game’s usability and performance across various platforms, ensuring a seamless user experience on different screen sizes and operating systems. |
|  | Trivia | Test the display and interaction of the game on different screen sizes and resolutions. | 1. Does the UI display correctly on various devices? 2. Are question texts and answer options properly formatted for different screens? 3. Does the timer and score Main menu function properly? | Cross-Platform Testing, UI Responsiveness | Ensure the trivia game displays correctly and functions well across various devices, with proper formatting for different screen sizes and resolutions. |

Below is the Chart diagram of this instead of a table (Simplified):

Note: To see more, save the image or use CTRL + to zoom in.

A close-up of a grid

AI-generated content may be incorrect.

Below is a test of our unit testing approach:

| **Category** | **Game/Menu** | **Meets Requirements** |
| --- | --- | --- |
| **Unit Testing: Core Logic** | Break-Out | Yes |
|  | Tic-Tac-Toe | Yes |
|  | Trivia  Main Menu | Yes  Yes |
| **Unit Testing: UI Components** | Break-Out | Yes |
|  | Tic-Tac-Toe | Yes |
|  | Trivia  Main Menu | Yes  Yes |
| **Unit Testing: Data Handling** | Break-Out | Yes |
|  | Tic-Tac-Toe | Yes |
|  | Trivia  Main Menu | Yes  Yes |
| **Debugging Process** | Break-Out | Yes |
|  | Tic-Tac-Toe | Yes |
|  | Trivia  Main Menu | Yes  Yes |
| **Performance Testing** | Break-Out | Yes |
|  | Tic-Tac-Toe | Yes |
|  | Trivia  Main Menu | Yes  Yes |
| **Cross-Platform Testing** | Break-Out | Yes |
| (Windows, Mac, Linux, IDE, etc) | Tic-Tac-Toe | Yes |
|  | Trivia  Main Menu | Yes  Yes |
| **Additional Features & Fixes** | Break-Out | No (Currently being fixed during Unit 4 and 5: Game ending and main menu transition) |
|  | Tic-Tac-Toe | No (Currently being fixed during Unit 4 and 5: Game ending and main menu transition) |
|  | Trivia | No (Currently being fixed during Unit 4 and 5: Game ending and main menu transition) |
|  | Main Menu | No (Currently being fixed during Unit 4 and 5: main menu transition after game ends) |

**Unit Testing Code (Test.py):**

import unittest

import numpy as np

import json

# Import specific functions and variables from the TicTacToe module

from TicTacToe import mark\_cell, is\_cell\_empty, check\_win, is\_board\_full, board

# Import classes and functions from the Breakout module

from Breakout import Block, Ball, Striker, collisionChecker, populateBlocks

# ----------------------------

# Test Cases for Tic Tac Toe

# ----------------------------

class TestTicTacToe(unittest.TestCase):

# Test marking a cell and ensuring it's no longer empty

def test\_mark\_and\_check\_cell(self):

board[:, :] = 0 # Reset board

mark\_cell(0, 0, 1)

self.assertFalse(is\_cell\_empty(0, 0))

# Test a winning condition (horizontal row)

def test\_win\_conditions(self):

board[:, :] = 0

board[0] = [1, 1, 1] # Simulate player 1 win on row 0

self.assertTrue(check\_win(1))

# Test for full board detection (used to check for draw)

def test\_board\_full(self):

board[:, :] = 1 # Fill all cells

self.assertTrue(is\_board\_full())

# ----------------------------

# Test Cases for Trivia Game

# ----------------------------

class TestTrivia(unittest.TestCase):

# Test that the questions JSON is structured correctly

def test\_questions\_json\_structure(self):

with open("Questions.json", "r") as file:

data = json.load(file)

# Check top-level structure

self.assertIsInstance(data, dict)

# Check each question object

for category, questions in data.items():

self.assertIsInstance(questions, list)

for q in questions:

self.assertIn("question", q)

self.assertIn("answers", q)

self.assertIn("correct", q)

self.assertIsInstance(q["answers"], list)

self.assertIsInstance(q["correct"], int)

self.assertTrue(0 <= q["correct"] < len(q["answers"]))

# Test that the correct answer index maps to the correct text

def test\_trivia\_answer\_index(self):

sample\_question = {

"question": "Sample?",

"answers": ["A", "B", "C", "D"],

"correct": 2

}

self.assertEqual(sample\_question["answers"][sample\_question["correct"]], "C")

# ----------------------------

# Test Cases for Breakout Game

# ----------------------------

class TestBreakout(unittest.TestCase):

# Ensure block health is reduced on hit

def test\_block\_health\_reduction(self):

block = Block(0, 0, 40, 20, (255, 0, 0))

original\_health = block.getHealth()

block.hit()

self.assertEqual(block.getHealth(), original\_health - 100)

# Check that blocks can be destroyed after repeated hits

def test\_block\_destruction(self):

block = Block(0, 0, 40, 20, (0, 255, 0))

while block.getHealth() > 0:

block.hit()

self.assertLessEqual(block.getHealth(), 0)

# Confirm ball resets to correct position and direction

def test\_ball\_reset(self):

ball = Ball(200, 200, 5, 5, (255, 255, 255))

ball.reset()

self.assertEqual((ball.posx, ball.posy), (0, 450))

self.assertEqual((ball.xFac, ball.yFac), (1, -1))

# Test ball bounces off the wall and reverses direction

def test\_ball\_bounce\_off\_wall(self):

ball = Ball(0, 200, 5, 5, (255, 255, 255))

ball.xFac = -1

ball.update()

self.assertEqual(ball.xFac, 1)

# Test ball direction changes after hitting the paddle

def test\_ball\_hit\_paddle(self):

ball = Ball(100, 100, 5, 5, (255, 255, 255))

ball.yFac = 1

ball.hit()

self.assertEqual(ball.yFac, -1)

# Ensure the striker stays within screen bounds

def test\_striker\_position\_bounds(self):

striker = Striker(0, 0, 100, 20, 10, (255, 255, 255))

# Test left boundary

striker.posx = -50

striker.posx = max(0, min(striker.posx, 750 - striker.width))

self.assertGreaterEqual(striker.posx, 0)

# Test right boundary

striker.posx = 1000

striker.posx = max(0, min(striker.posx, 750 - striker.width))

self.assertLessEqual(striker.posx, 750 - striker.width)

# Test that collisions between ball and block are detected

def test\_collision\_detection(self):

block = Block(100, 100, 40, 20, (0, 255, 0))

ball = Ball(110, 110, 5, 5, (255, 255, 255))

self.assertTrue(collisionChecker(block.getRect(), ball.getRect()))

# Test that populateBlocks creates valid block objects

def test\_populate\_blocks(self):

blocks = populateBlocks(40, 15, 10, 10)

self.assertGreater(len(blocks), 0)

for block in blocks:

self.assertIsInstance(block, Block)

# Entry point to run the tests

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

**test.yml:** This link below render the code to the test.yml file which uses the test.py to make GitHub Action to automate the unit tests for our application. This is **ONLY** the layout and is not tested yet if it works since we wanted to use the manual old style version. Use test.py for the real method to test it. This is included since it was mentione din the Unit 4 feedback to try out. This is continued in unit 5.

<https://gist.github.com/VictoriaRaven/8ce8069252ec46b6b28271c3f65e6edb>

# **APPENDIX E:**

**Git (GitHub Version Control):**

Below is the table for the Git Version Control Process and Steps that the team will utilize. This is based on GeeksforGeeks (n.d.) and Tsui, F., Karam, O., & Bernal, B. (2014). We will update this regularly per unit week.

**Git Version Control Steps Table**

| **Step Number** | **Step Description** | **Commands/Action** | **Explanation/Detail** | **Software Engineering Technique** | **Goal/Outcome** |
| --- | --- | --- | --- | --- | --- |
| 1 | Initialize Git Repository | git init | Initializes a new Git repository in the project folder. Creates a .git folder. | Version Control, Initialization | Creates the foundation for tracking and managing changes in the codebase. |
| 2 | Add files to the staging area | git add <file> or git add . | Adds specific files or all files to the staging area to prepare for committing. | Version Control, Staging | Ensures that the changes are tracked and ready to be committed. |
| 3 | Commit changes | git commit -m "message" | Commits the staged changes to the local repository. The commit message describes the changes. | Version Control, Commit | Records a snapshot of the changes in the project history. |
| 4 | Verify commit history | git log | Displays the commit history for the repository, showing all commits made in chronological order. | Version Control, History Tracking | Allows developers to track changes over time and revert if necessary. |
| 5 | Branch creation | git branch <branch-name> | Creates a new branch for working on specific features or changes. | Version Control, Branching | Enables isolated work on different features, preventing code conflicts in the main branch. |
| 6 | Switch between branches | git checkout <branch-name> | Switches to an existing branch to work on a different feature or fix an issue. | Version Control, Branch Switching | Allows developers to work on different features or fixes independently without interference. |
| 7 | Merge branches | git merge <branch-name> | Merges changes from one branch into another (typically merging a feature branch into the main branch). | Version Control, Merging | Integrates the work from different branches into the main project. |
| 8 | Handle merge conflicts | Manual resolution of conflicts | Occurs when changes in two branches contradict each other. Git flags the conflict, and developers must manually resolve it. | Version Control, Conflict Resolution | Resolves discrepancies between branches to maintain a consistent codebase. |
| 9 | Push changes to remote repository | git push origin <branch-name> | Pushes local commits to the remote repository for sharing and collaboration. | Version Control, Remote Collaboration | Ensures that local changes are shared with the team and are available in the central repository. |
| 10 | Pull latest changes from remote repository | git pull origin <branch-name> | Fetches and merges the latest changes from the remote repository into the local branch. | Version Control, Collaboration | Keeps the local repository up-to-date with the latest changes from other developers. |
| 11 | Clone a repository | git clone <repository-url> | Clones an existing repository from a remote location to start a local copy. | Version Control, Collaboration | Allows the developer to work on a local version of an existing remote repository. |

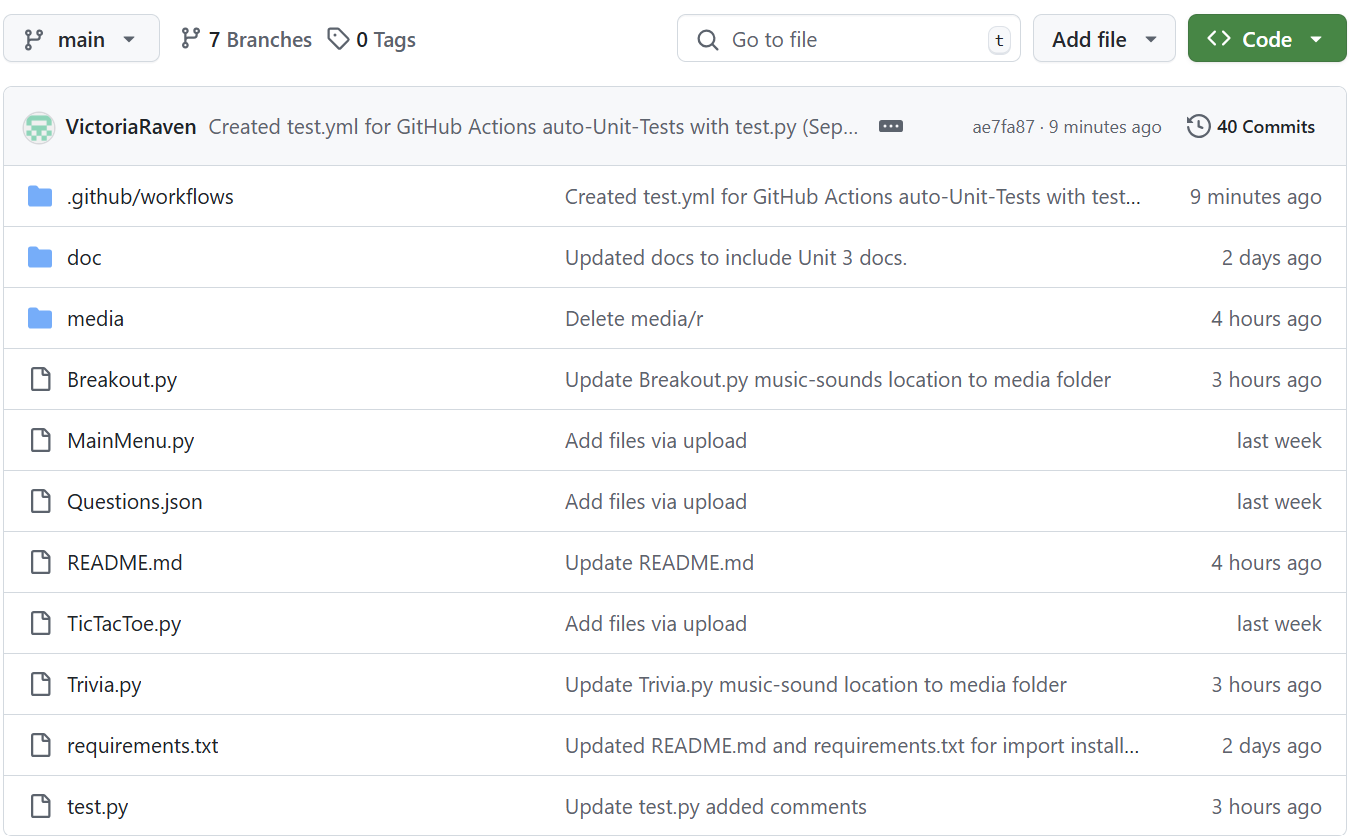
**Explanation:**

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.

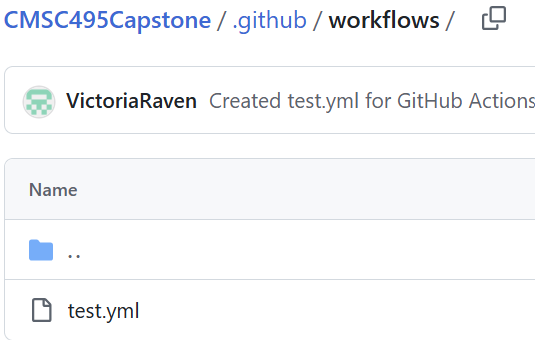
Main Branch Files:



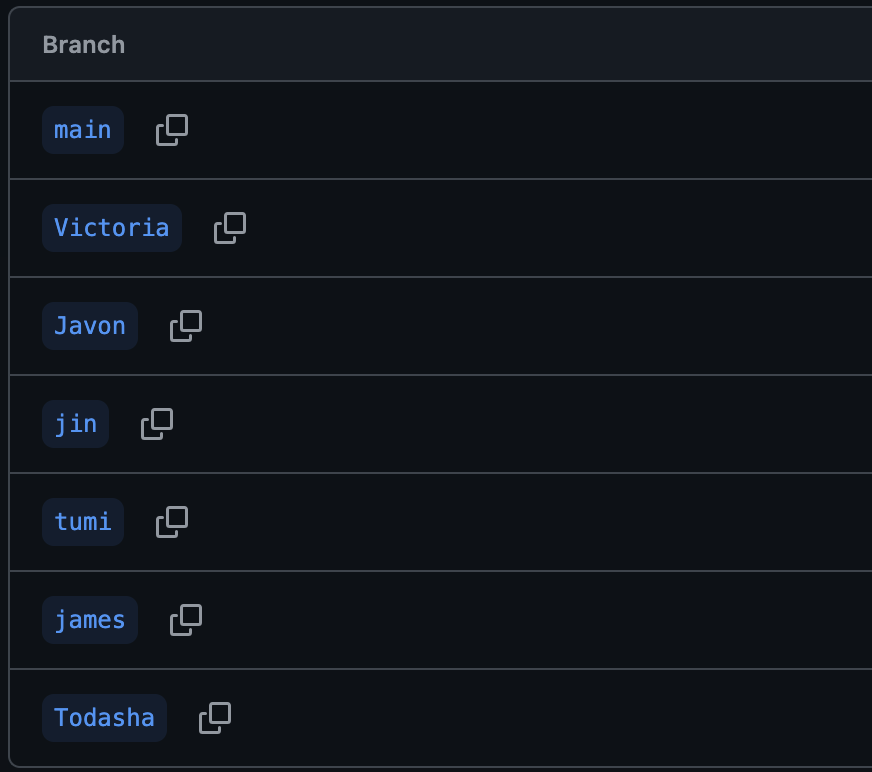
Media files and doc files and .github/workflow files:

A screenshot of a computer

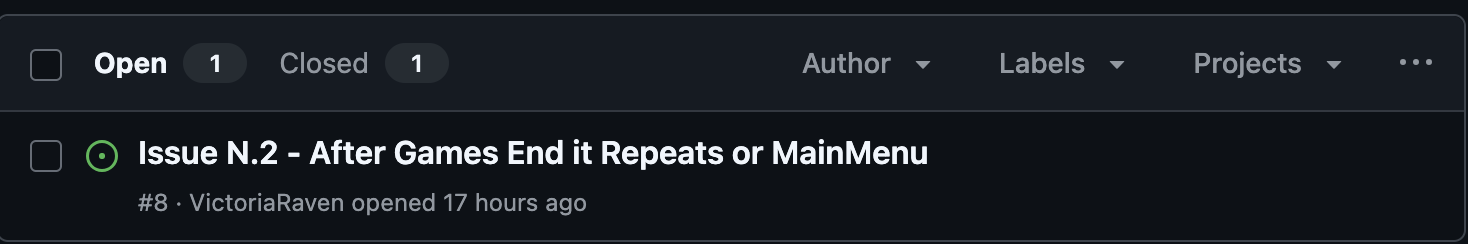
AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect. 

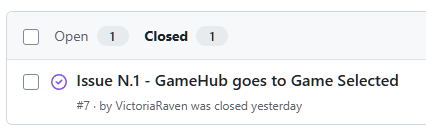
We have a total of 7 branches:



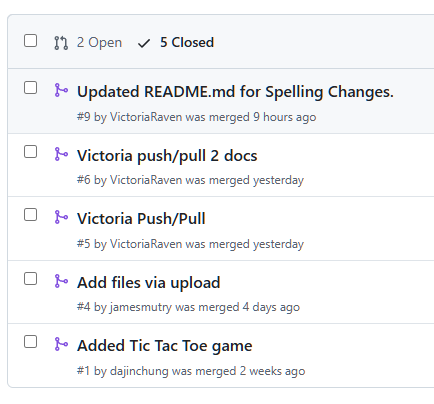
We have reported 1 glitch so far:



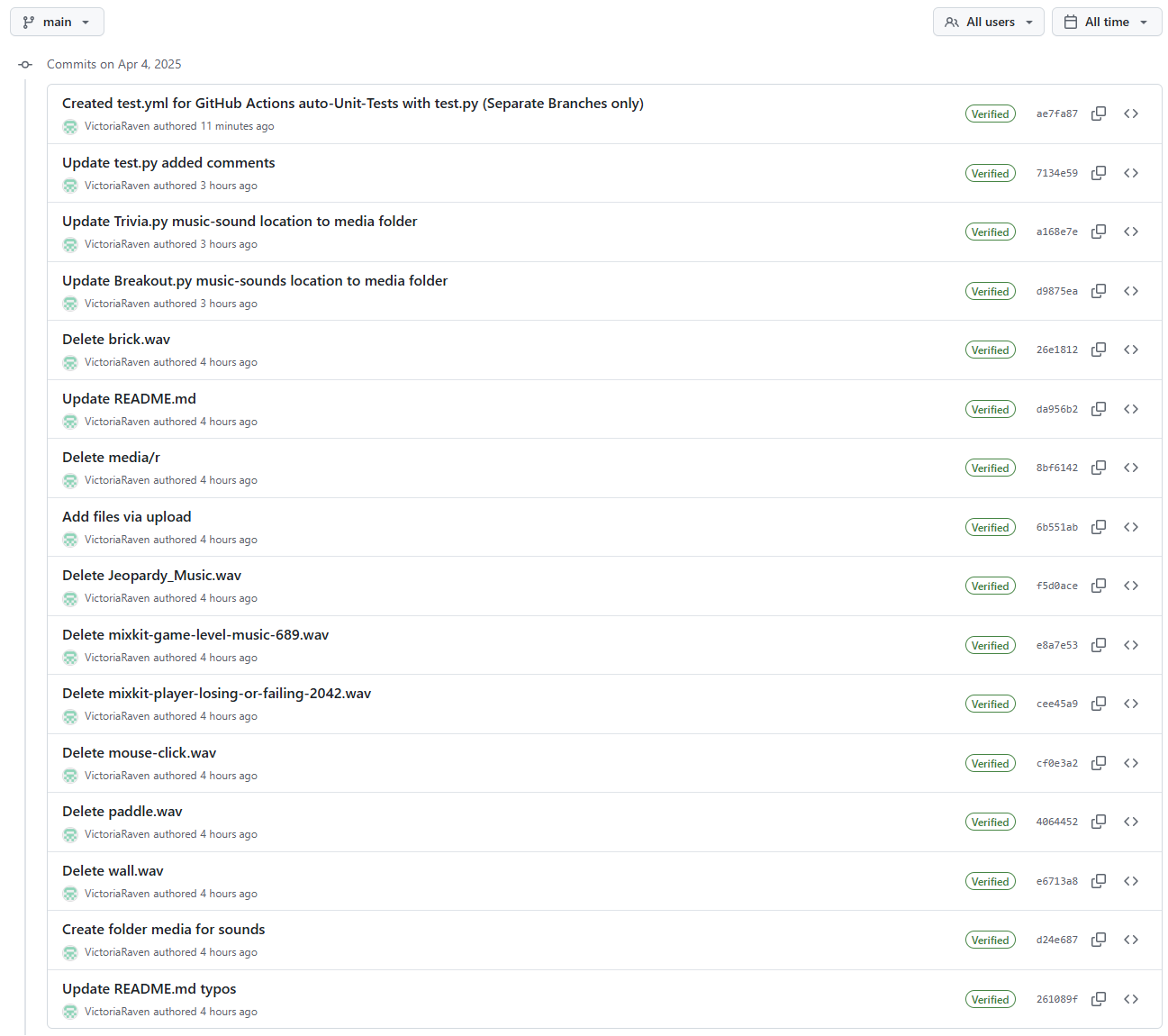
We have closed 1 issue:

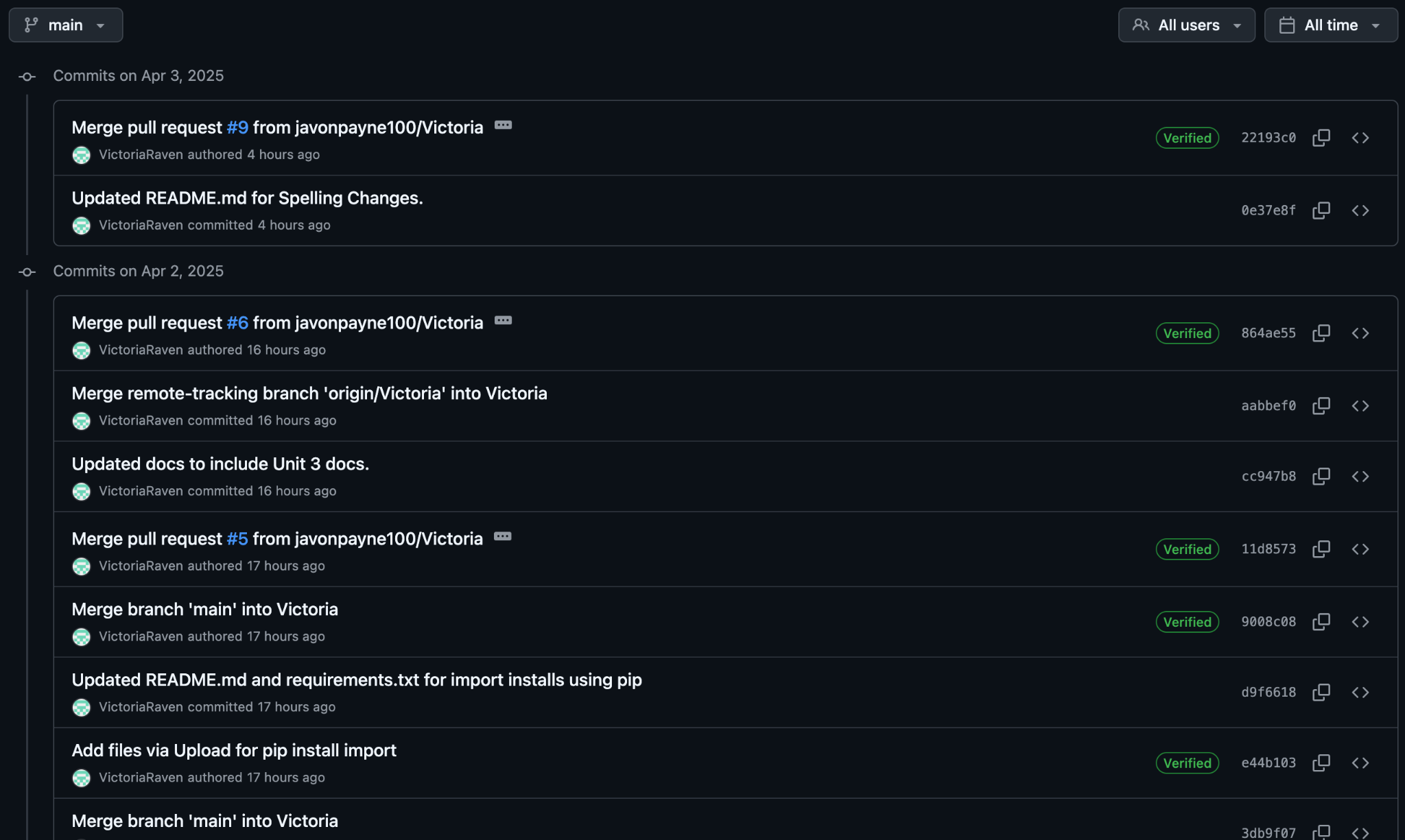


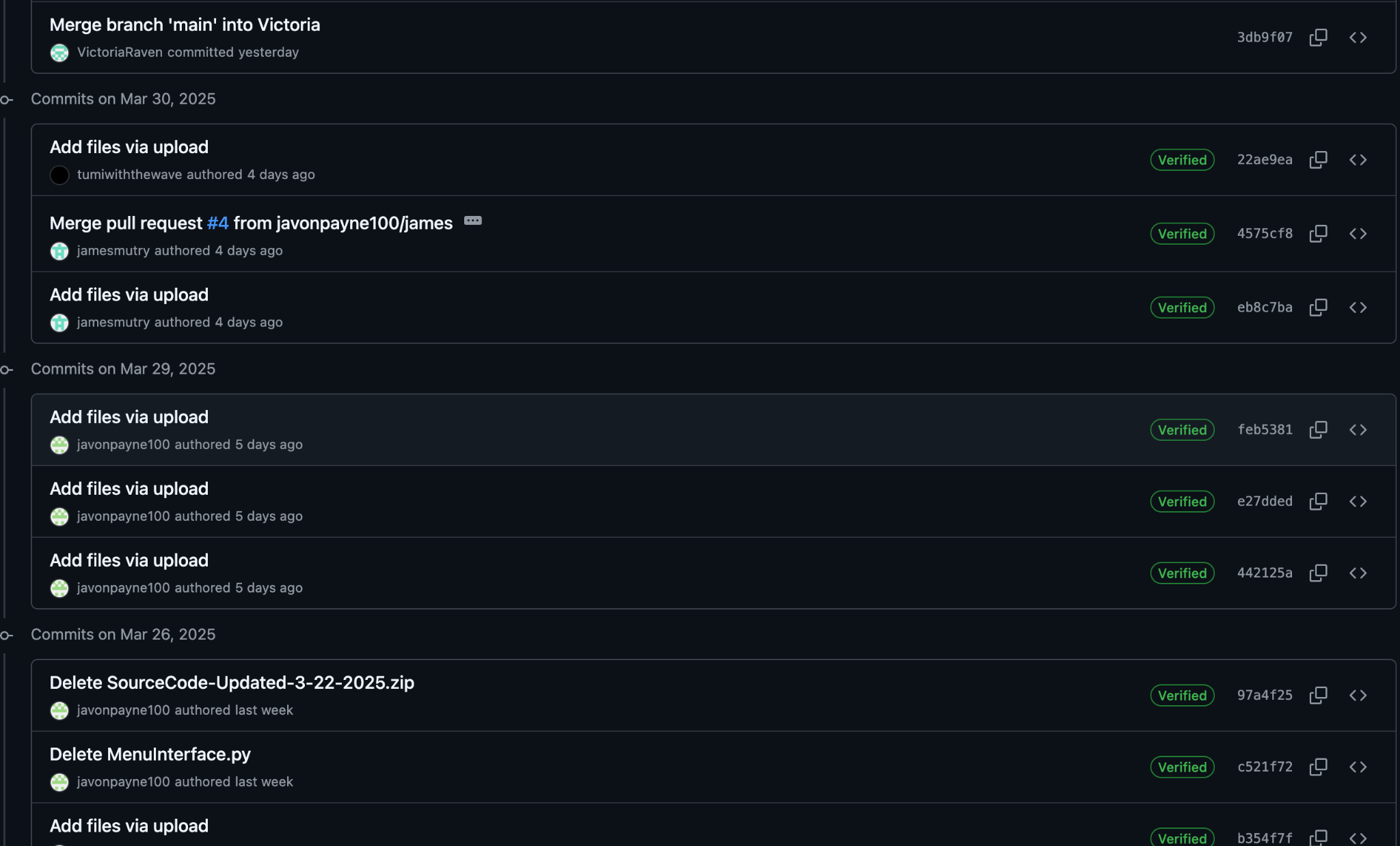
Here is our pull requests to merge onto the main using the branches:



As of April 4th, 2025 we have made a total of 40 commits:









# **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>.

Creately. (n.d.). *UML diagram types & examples*. Creately. <https://creately.com/blog/diagrams/uml-diagram-types-examples/>

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>

Draw.io. (2021, June 15). *3 diagrams every computer science student should know*. draw.io. <https://drawio-app.com/blog/3-diagrams-every-computer-science-student-should-know/>

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

GeeksforGeeks. (2024, October). *Use case diagram*. GeeksforGeeks. [https://www.geeksforgeeks.org/use-case-diagram/](https://www.geeksforgeeks.org/use-case-diagram/?form=MG0AV3)

GeeksforGeeks. (2024). *UML class diagram notation*. GeeksforGeeks. <https://www.geeksforgeeks.org/unified-modeling-language-uml-class-diagrams/#uml-class-notation>

GeeksforGeeks. (2025). *Activity Diagrams*. GeeksforGeeks. <https://www.geeksforgeeks.org/unified-modeling-language-uml-activity-diagrams/>

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. In *Applied software project management* (Part I, Chapters 3, 4, and 6). O'Reilly Media. <https://go.oreilly.com/umgc/https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch03.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>