Project Design

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.**](#_heading=h.gxmga0bondar) **User Interface and Functionality** 3

[**2.**](#_heading=h.vsb2op5mn706) **Application Structure** 6

[**3.**](#_heading=h.9uofblri3o31) **UML Diagrams** 15

[**4.**](#_heading=h.xjm83njjtpb9) **APPENDIX A:** 18

[**5.**](#_heading=h.oc1egve1vnnt) **APPENDIX B:** 20

[**6.**](#_heading=h.arcjt6h07tjm) **APPENDIX C:** 22

[**7.**](#_heading=h.3eixzrkr1ry1) **APPENDIX D:** 23

[**8.**](#_heading=h.9bld1nvd9d2h) **APPENDIX E:** 29

[**9.**](#_heading=h.7kx8tag5dc1g) **References** 35

# **User Interface and Functionality**

**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.
* **Unit Testing:**
  + **Unit Testing 1: 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.
  + **Unit Testing 2: 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.
  + **Unit Testing 3: High Score & User Progress Saving:** 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.
  + **Unit Testing 4: Debugging Lives and Scoreboard Display:** Test to ensure that the number of lives and scores are being correctly tracked, displayed, and updated during gameplay.
  + **Unit Testing 5: 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.

**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.

So far, Group 1 has developed an intuitive user interface for the Python Game Hub, a centralized platform that allows users to select and play multiple simple games such as Breakout, Tic-Tac-Toe, and Trivia within a single application. The application will be accessible on desktops with minimal system requirements, ensuring compatibility with most operating systems. Additional functionality will include:

* Game Selection Menu: A central hub where users can choose from different games.
* Game Instructions: View the rules and guidelines for each game.
* High Score Tracking: Display high scores for each game to encourage competition.
* User Profiles: If integrated, users will have profiles that store game progress and high scores (SQLite database).
* Functional Requirements for UI: Define the exact functionality of the UI components (e.g., buttons, text input fields, score displays, etc.). You might mention specific components like a Start button, Exit button, Restart Game button, etc.
* Accessibility Features: Highlight any accessibility features that will make the platform user-friendly, such as keyboard navigation, screen reader compatibility, or color contrast for visually impaired users.
* Error Handling: Include how the UI handles errors or invalid inputs (e.g., if a user enters an invalid profile name or tries to play a game that isn’t loaded).

**Time and Resource Estimations:** To complete each task, it will go by weekly updates. It should take about 4-5 weeks to implement all of these, where each week there is 1 task. It will be completed by each role and the project manager will check on it.

# **Application Structure**

The Python Game Hub will consist of several key components developed using Python and the Pygame library for game logic and GUI creation. The application will include a main menu for game selection and separate modules for each game. The structure of the application will be broken as follows:

* Menu/UI System: A central interface where users can navigate between games, view instructions, and manage profiles.
* Game Modules: Each game (Tic-Tac-Toe, Breakout, Trivia) will be developed as a separate module to ensure modularity and ease of maintenance.
* [If possible] - Database: An SQLite database will be used to store data or make other changes.
* Backend Logic: The backend Python code will handle game logic, player input, and score management.
* **Note: Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process**

**Application Requirements:** These requirements define the core functionalities of the Python Game Hub application.

**Main Menu/UI:**

* Select a game to play
* View Game Instructions
* Exit the application
* Interaction: Tic-Tac-Toe, Breakout, and Trivia by invoking the game logic when the user selects a game.

**Tic-Tac Toe:**

* 2-player mode
* Display Win, Loss, or Draw Conditions
* Clear game board for a new game
* Interaction: connects with the UI module to update the screen and display the scores. If possible also include a database to connect with SQL.

**Breakout:**

* Ball movement with physics-based collision
* Player-controlled paddle movement
* Brick destruction mechanics and score tracking
* Interaction: connects with the UI module to update the screen and display the instructions and scores. If possible also include a database to connect with SQL.

**Trivia:**

* Display random trivia questions with multiple-choice answers
* Keep track of correct and incorrect answers
* Provide feedback for user responses
* Interaction: connects with the UI module to update the screen, questions, answers and display the scores. If possible also include a database to connect with SQL.

**Database Module (SQLite):**

* Stores user profiles, high scores, and game progress.
* This module will ensure that the data persists between sessions.
* Interaction: It interacts with all game modules to save and retrieve data (e.g., user scores, profiles, progress) as required by the game logic.

**Overall Interaction Between Components:**

* Main Menu & Game Modules: When the user selects a game from the menu, the Menu/UI module passes control to Tic-Tac-Toe, Breakout, or Trivia. Each game module runs independently, with its own UI and game logic. After the game ends, the module can either return to the main menu or restart, depending on the user’s choice.
* Game Module & Database (If possible and optional): As the user plays a game, the game module will interact with the Database module to store relevant data. For example, record the score and the winner in the database, and user profiles name.
* UI & Game Modules: The UI module is responsible for displaying the game’s interface to the user, which means that it communicates with the game modules to show the status of the game: scores/winner/loser). It also handles user inputs like button clicks and keyboard presses.

**Database Structure (If Possible):**

* Users Table:
  + Fields: user\_id, username
  + Relationship: One-to-many relationship with Scores
* Scores Table\*\*:
  + Fields: user\_id, game\_name, score
  + Relationship: Many-to-one relationship with Users. The game\_name field will specify which game the score belongs to: Tic-Tac-Toe, Breakout, Trivia.

The GUI will be designed using Pygame, a Python library that provides easy-to-use tools for building game interfaces. The project will follow a structured Model-View-Controller (MVC) pattern:

* Model: Handles the game data and user input (game logic).
* View: Displays the game’s visual interface and feedback (Pygame rendering).
* Controller: Manages user interactions and updates the model based on input.

**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.



**Technical Requirements:** For the Python Game Hub project, the development team will utilize the following tools and techniques:

* Language – Python
* Library – PyGame (for game development)
* IDE – Any Python IDE (PyCharm, Visual Studio Code, etc)
* Code repository/ Version Control – GitHub for Git (sync files)
* Documentation – Microsoft Word
* Communication – Discord (verbal/text communication), Google/Microsoft Teams or Drive (Sometimes for file sharing as well)
* **Note: Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process**

The development team will use a Python IDE (e.g., PyCharm or Visual Studio Code) to write and test the code. These IDEs provide excellent debugging tools and support for Python, making them ideal for development. Git will be used for version control, allowing the team to manage changes, track progress, and collaborate effectively. The code will be shared via a Git repository (such as GitHub or GitLab), ensuring that all team members have access to the latest updates and can work on different parts of the game concurrently. All project documentation will be maintained in Microsoft Word. This ensures the documentation is easily accessible and compatible for team collaboration and future reference. The development team will use Discord or other alternatives for team communication, discussions, and updates. By utilizing Python, Pygame, Git, and other tools listed above, the project will be developed efficiently.

**Breaking Down the Phases in the Waterfall Methodology for this Project:**

1. **Requirement Analysis (Unit 1) – 1 week:**
   * Gather all requirements from stakeholders (e.g., end-users, project team).
   * Define all game features, user interface elements, and database structures clearly.
   * Produce a Requirements Specification Document.
   * Begin the coding ideas and process to start the methods and functions.
     1. Ex: Write out the class and methods, get, setters, headers, imports, but don’t write code yet, leave it empty.
2. **Requirement Analysis (Unit 2) – 1 week:**
   * Gather all requirements from stakeholders (e.g., end-users, project team).
   * Define all game features, user interface elements, and database structures clearly.
   * Produce a Requirements Specification Document.
   * Begin with the project coding and initiation plans
3. **System and Software Design (Unit 3) – 1 week:**
   * Detailed design of the UI and Game Modules (Tic-Tac-Toe, Breakout, Trivia).
   * Design the Database Schema for user profiles and high scores.
   * Produce UML diagrams, including Use Case, Class, and Activity Diagrams.
   * Begin coding based on the design specifications.
   * Develop the individual modules and integrate them into the main menu
   * **Note: Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process**
4. **Implementation and Integration (Development) (Unit 4) – 1 week:**
   * Continue coding based on the design specifications
   * Team is allowed to add other features to implementations if tasks from week three are completed.
   * Test and modify individual game modules (e.g., Tic-Tac-Toe, Breakout) and integrate them into the main Python Game Hub.
   * Create a
   * Implement database connections and the game selection system.
   * Conduct unit testing and Git Version control changes.
   * **Note: Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process**
5. **Integration and Testing (Unit 5) – 1 week**:
   * Conduct unit testing, a continuation from week 4, for individual components.
   * Conduct **integration testing** to ensure that all parts of the system work together smoothly.
   * Conduct **user acceptance testing** to ensure the final product meets user expectations.
   * **Note: Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process**
6. **Deployment (Unit 7) – 1 week:**
   * Once all testing is complete and the application is stable, deploy the game hub for use.
   * Ensure all user documentation and manuals are provided.
   * **Note: Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process**
7. **Maintenance (Unit 8) – 1 week:**
   * After deployment, continue with minor bug fixes or enhancements as needed based on user feedback.
   * Ensure the final documentation is complete and submit the source code and project deliverables.

**Project Risks and Limitations:** The Python Game Hub Development project carries several potential risks that could impact its timeline, functionality, and overall success. Technical risks include possible bugs, crashes, or performance issues, particularly when integrating multiple games into a single platform. Ensuring smooth transitions between games, proper resource management, and avoiding memory leaks will be crucial. Development risks arise if any developer encounters difficulties in implementing PyGame logic or game mechanics, potentially leading to delays or inconsistencies in gameplay. Team collaboration risks include miscommunication or lack of coordination, which could result in integration challenges between different game modules and the UI. Additionally, scope creep is a potential risk if new features or games are added without proper planning, leading to extended development time and resource constraints. User experience risks include unintuitive navigation or poorly optimized controls, which could affect engagement and usability. To mitigate these risks, the team must maintain regular communication, follow a structured development plan, conduct thorough testing, and adhere to project scope and deadlines to ensure a smooth and functional final product.

**Detailed Boundaries**: Expand on what is not included in the project (e.g., multiplayer support, advanced AI for Tic-Tac-Toe, or online scoreboards).

**Limitations**: Any known limitations of the project, such as performance concerns on older hardware, absence of game audio or animations due to time or budget constraints, etc.

**Waterfall** **Approach Risks (GeeksforGeeks (n.d.) and UMGC (n.d.):**

Risks typically emerge early in the development cycle, so it’s important to address them as part of the planning phase:

1. **Risk: Delays in Requirements Gathering**
   * **Mitigation**: Allocate sufficient time for meetings and ensure that all requirements are documented early in the project. Review and confirm these requirements before moving to design.
2. **Risk: Overlap or Unclear Requirements Between Game Modules**
   * **Mitigation**: Conduct frequent design reviews to ensure that each game’s requirements are well-understood and documented before development starts.
3. **Risk: Integration Issues Between Game Modules**
   * **Mitigation**: Test modules separately and ensure a comprehensive testing phase after integration to ensure all parts work together. We will use version control (e.g., Git) to manage changes.
   * **Note: Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process**
4. **Risk: Testing Delays Due to Complexity or Bugs**
   * **Mitigation**: Plan for adequate time for both integration and user acceptance testing. Prepare detailed test cases to ensure thorough coverage. Plan for unexpected delays caused by bug fixes or complexity.
   * **Note: Refer to APPENDIX D for the Testing Unit Process and APPENDIX Efor the Git Version Control Process**
5. **Risk: Resource Constraints (e.g., limited team members or tools)**
   * **Mitigation**: Ensure that adequate resources (team members, tools, etc.) are available for each phase of development. Plan for contingencies in case resources become limited during the project and have backup plans in place.
   * **Note: Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process**

**Schedule:**

**Milestones:** Identify clear milestones (e.g., completion of each mini-game module, main menu, user profile system, etc.).

* **Completion of Mini-Game Modules:**
* Breakout
* Trivia
* Tic-Tac-Toe
* **Completion of the Main Menu and Main Interface:**
* Finalizing the navigation structure of the game hub.
* Ensuring that the user interface is intuitive and connects well with the game modules.
* **Completion of Debugging, Testing, Modifications, Git Commits/Merges, and Branch Pulls:**
* Finalizing all individual game module code and conducting debugging cycles.
* Keeping track of all code changes via version control (Git).
* Merging branches and resolving any conflicts.
  + **Note: Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process**
* **Integration and System Testing:**
* Merging all game modules into the main game hub.
* Testing the integration of all modules to ensure they function cohesively as a single system.
* **Score Tracking and System Testing:**
* Testing out the score-tracking system (e.g., high scores, win/loss/tie logic).
* Ensuring that all data is properly stored and retrieved from the database.

**APPENDIX A** is our team’s sample of the Gantt Chart in a large table of our plans, and we will be using this schedule to make sure we stay focused, however; plans are not set completely and therefore might be changed (UMGC, n.d.). This is a repeat of Project Plan for unit 2 which is why it is not here but on the Appendix. **Note:** Last day of CMSC 495 is on May 6, 2025, at 11:59pm ET, but we will turn it in on May 4, 2025, at 11:59pm ET in case we want to make some minor changes. (ex: reupload files, etc.). **REFER TO APPENDIX A** for an update of the Gantt Chart per each unit and deadlines to submit the documents or files, but the discussions will be due on Tuesday at 11:59 PM ET (UMGC, n.d.):

**Project Schedule (Waterfall Approach)**

1. **Unit 1 and 2: Team Formation and Project Plan (plan 1)**
   * Duration: **1 week per unit**
   * Tasks: Gather and document all requirements, finalize the feature list, and create a requirement specification document.
2. **Unit 3: Project Design Plan (plan 2)**
   * Duration: **1 week**
   * Tasks: Design game modules, create UI/UX designs, create database schema, draft UML diagrams.
   * Constantly updating the Git version control and the Unit Testing for debugging.
   * **Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process**
3. **Unit 4: Phase 1 (Implementation, Coding, and Integration)**
   * Duration: **5 weeks (starting from unit 2)**
   * Tasks: Develop core functionality for each game (Tic-Tac-Toe, Breakout, Trivia), integrate the database, develop the main menu, and implement user profiles and score tracking.
   * **Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process**
4. **Unit 5: Integration and Testing**
   * Duration: **2 weeks**
   * Tasks: Test individual game components, integrate the system, conduct integration testing, fix bugs, and perform user acceptance testing.
   * **Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process**
5. **Unit 6: Deployment (Phase 2 Code).**
   * Duration: **1 week**
   * Tasks: Finalize deployment, prepare user documentation, and deploy the application.
   * **Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process**
6. **Unit 7: Maintenance and User Guide**
   * Duration: Ongoing as needed
   * Tasks: Address user-reported bugs, and provide updates or enhancements. Write the document and finalize it. Turn in the source code and final document.
   * Make sure it matches with the user Guide
   * **Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process**
7. **Unit 7: Maintenance (Optional)**
   * Duration: Ongoing as needed
   * Tasks: Address user-reported bugs and provide updates or enhancements. Write the document and finalize it. Turn in the source code and final document.
   * Project Manager must manage and look to make sure all the reuqriments are met before turning it in as this is the final week.
   * **Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process**

**Work Breakdown/Task dependencies (aligned with the Waterfall Schedule):**

1. **Requirement Analysis (Unit 1 & 2) – 1 week:**
   * **Task 1:** Meet with the team and discuss project requirements in detail.
   * **Task 2:** Document both functional and non-functional requirements.
   * **Task 3:** Confirm the scope of the game hub (e.g., number of games, specific features).
2. **Design / Development (Unit 3) – 1 week:**
   * **Task 1:** Design the user interface (UI) layout for the game hub
   * **Task 2:** Design individual game logic (Tic-Tac-Toe, Breakout, Trivia).
   * **Task 3:** Design the database schema for storing scores and winner/loser/tie.
   * **Task 4:** Create UML diagrams (Use Case, Class, and Activity Diagrams) to represent system flow and structure.
   * **Task 5:** Implement Unit testing and the Git Control integration for th code:
   * **Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process**
3. **Development /Phase 1 (Unit 4 and 5) – 1 weeks:**
   * **Task 1:** Develop the main menu, where users can navigate and select different games and make sure the menu interface connects well with it.
   * **Task 2:** Code the individual game modules (Tic-Tac-Toe, Breakout, Trivia) ensuring all game rules and mechanics work as expected (testing and debugging).
   * **Task 3:** Implement high score tracking and lives, winner/loser/tie, and integrate the database if possible for other data management.
   * **Task 4:** Integrate the database and the game hub system. Then test out and debug.
   * **Task 5:** Conduct unit testing for each game module (Tic-Tac-Toe, Breakout, Trivia). Suggest or add other optional elements to enhance the game.
   * **Task 6:** Perform integration testing to ensure all game modules and features work together as a complete system. Debug other additional enhancements.
   * **Task 7:** Conduct user acceptance testing to ensure the game hub meets the expectations of end-users and the project manager and testers will check on it.
   * **Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process**
4. **User Guide (Unit 7) – 1 weeks:**
   * **Task 1:** Conduct unit testing for each game module (Tic-Tac-Toe, Breakout, Trivia). Suggest or add other optional elements to enhance the game.
   * **Task 2:** Perform integration testing to ensure all game modules and features work together as a complete system. Debug other additional enhancements.
   * **Task 3:** Conduct user acceptance testing to ensure the game hub meets the expectations of end-users and the project manager and testers will check on it.
   * **Task 4:**Prepare and finalize the game hub for release, ensuring all features are polished and functional.
   * **Task 5:** Write user documentation and provide installation guides.
   * **Task 6:** Deploy the game hub for use and ensure it is accessible to users.
   * **Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process**
5. **Deployment (Unit 7 and Unit 8) – 3 weeks:**
   * **Task 1:**Prepare and finalize the game hub for release, ensuring all features are polished and functional.
   * **Task 2:** Write user documentation and provide installation guides.
   * **Task 3:** Deploy the game hub for use and ensure it is accessible to users.
   * **Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process**

**Project Evaluation Plan**

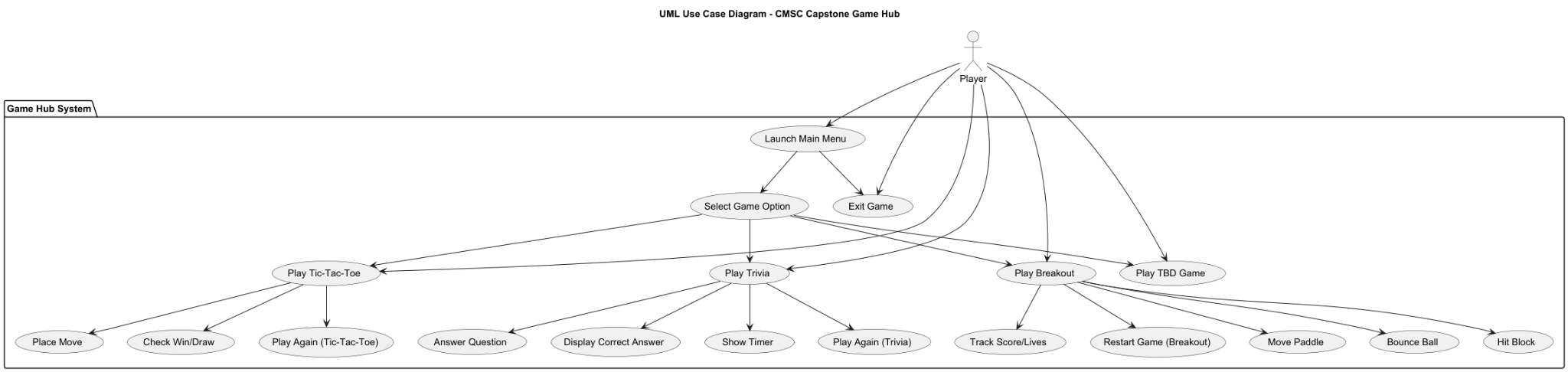
Since the project follows the Waterfall methodology, evaluation will take place at specific milestones:

1. **At the end of each phase**:
   * Review the completed deliverables against the requirements.
   * Conduct internal reviews with the team and stakeholders to ensure that the project is on track.
   * **Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process**
2. **User Acceptance Testing (UAT)**:
   * At the end of the testing phase, evaluations are:
     + Functionality: Does the game hub work as intended?
     + Usability: Is the user interface intuitive and easy to navigate?
     + Performance: Does the application run smoothly on all supported systems?
   * **Refer to APPENDIX D for the Testing Unit Process and APPENDIX E for the Git Version Control Process**

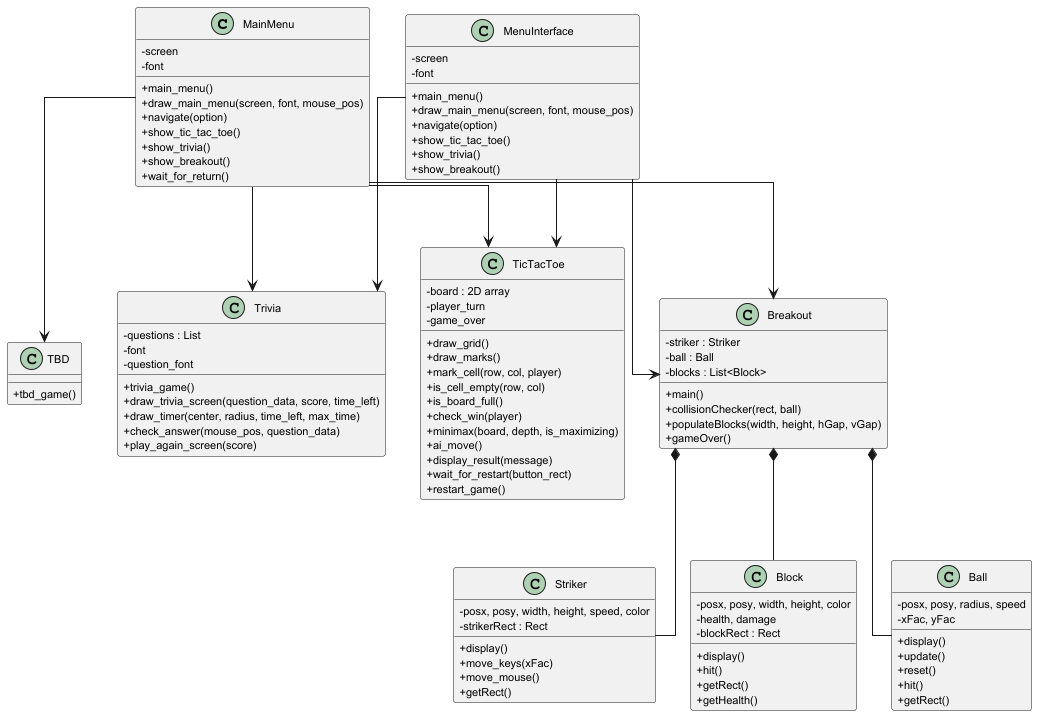
By structuring the Waterfall methodology clearly into sequential phases (Requirements Analysis, Design, Implementation, Testing, Deployment, Maintenance) and expanding on key areas like task breakdown, scheduling, and risk management, you ensure that the project will be executed in a structured, predictable manner.

# **UML Diagrams**

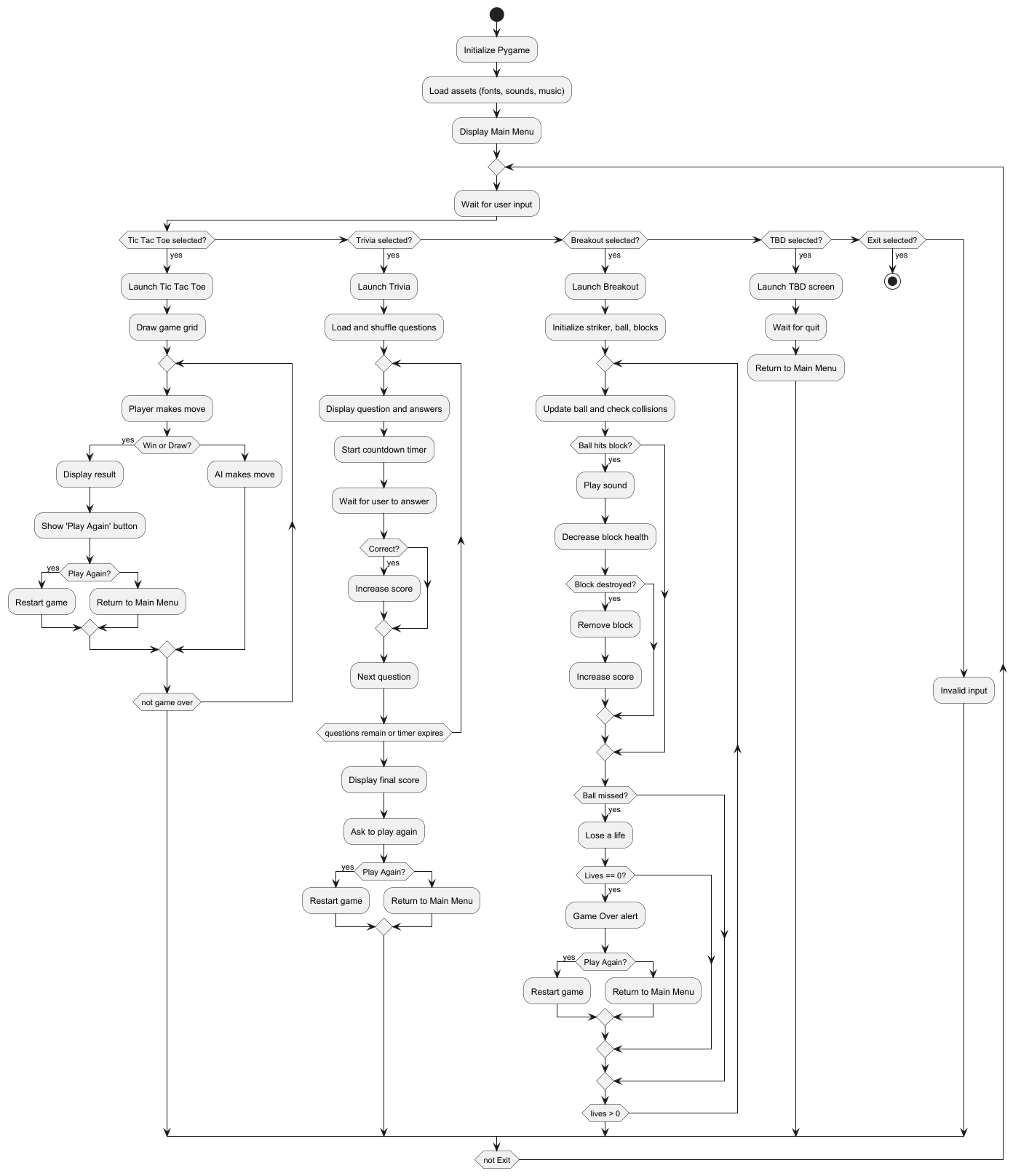
**UML Use Case Diagram**:



**Class Diagram**:



**Activity Diagrams**:



# **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 1 |

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.Below is a larger size of this:

A white background with black text

AI-generated content may be incorrect.

So far, units 1, 2, and 3 are completed. The date on unit 3 is slightly off, as we adjusted it and resubmitted the project design plan on 3/31/2025 or 4/1/2025.

# **APPENDIX B:**

CONTRIBUTION REPORT

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

Project: Python Game Hub

| TASKS COMPLETED / CONTRIBUTION LOG | | | |  |
| --- | --- | --- | --- | --- |
| DATE | TASKS/MILESTONES | NAME OF VOLUNTEER | HOURS | COSTS |
| 3/10/2025 to 3/14/2025 | Created a team for the project  Worked on plans and git version and included unit testing for week 3 prep. | 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 | ~ 4 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)  Worked on setting a plan for unit 3 testing and git version control | Lee, Victoria;  Mutry, James  Chung, Dajin | ~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 | ~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 |  |
| 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 | ~ 2-4 hour per each person | $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 | ~ 4 hours per person | $0 |
| 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 |

**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. We will update this in a different table in UNIT 4 weeks to ensure that it meets all the requirements. This is not the official version and may change the appearance as well. This is based on GeeksforGeeks (n.d.) and Tsui, F., Karam, O., & Bernal, B. (2014).

| **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 Manin 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. |

**Table Summary of the Unit Testing:**

Unit Testing ensures that each game component functions correctly, and debugging identifies and resolves any issues within each game's functionality, including the game mechanics, user interface, and data handling. Performance and Cross-Platform Testing validate that each game performs well across different devices, operating systems, and screen sizes, ensuring an optimal experience for the user. Software Engineering Techniques such as Test-Driven Development (TDD), Debugging, and Cross-Platform Testing are applied to create properly playing games. This table provides a detailed and systematic approach to testing and debugging in the Python Game Hub, incorporating best practices from software engineering to ensure that each aspect of the game functions as expected.

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: We will update this in a different table in UNIT 4 weeks to ensure that it meets all the requirements. This is not the official version and may change the appearance as well. This is based on GeeksforGeeks (n.d.) and Tsui, F., Karam, O., & Bernal, B. (2014).

| **Category** | **Game** | **Meets Requirements** |
| --- | --- | --- |
| **Unit Testing: Core Logic** | Break-Out | Yes |
|  | Tic-Tac-Toe | Yes |
|  | Trivia | Yes |
| **Unit Testing: UI Components** | Break-Out | Yes |
|  | Tic-Tac-Toe | Yes |
|  | Trivia | Yes |
| **Unit Testing: Data Handling** | Break-Out | Yes |
|  | Tic-Tac-Toe | Yes |
|  | Trivia | Yes |
| **Debugging Process** | Break-Out | Yes |
|  | Tic-Tac-Toe | Yes |
|  | Trivia | Yes |
| **Performance Testing** | Break-Out | Yes |
|  | Tic-Tac-Toe | Yes |
|  | Trivia | Yes |
| **Cross-Platform Testing** | Break-Out | Yes |
|  | Tic-Tac-Toe | Yes |
|  | Trivia | Yes |
| **Additional Features & Fixes** | Break-Out | No (Currently being fixed during Unit 4: Game ending and main menu transition) |
|  | Tic-Tac-Toe | No (Currently being fixed during Unit 4: Game ending and main menu transition) |
|  | Trivia | No (Currently being fixed during Unit 4: Game ending and main menu transition) |

# **APPENDIX E:**

**Git (GitHub Version Control) (Updates per Unit week):**

Below is the table for the Git Version Control Process and Steps that the team will utilize. We will update this in a different table in UNIT 4 weeks to ensure that it meets all the requirements. This is not the official version and may change the appearance as well. This is based on GeeksforGeeks (n.d.) and Tsui, F., Karam, O., & Bernal, B. (2014).

**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. |

This table outlines the essential Git version control steps used in a typical development workflow to manage your code changes, collaborate with team members, and track project progress over time. Each step is associated with the specific Git commands used and their related software engineering techniques, ensuring clear and efficient version control.

Below are the current updates. Note: save the image or use CTRL + to zoom in.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Below is the reasons why utilizing Unit Testing and Git Version Control are beneficial by GeeksforGeeks (n.d.) and Tsui, F., Karam, O., & Bernal, B. (2014). And ALL the Refrences. This demonstrates why Git Version Control Process correlates with Unit Testing throughout the development lifecycle.

**Git Version Control Process and Unit Testing Correlation**

| **Step Number** | **Git Version Control Step** | **Unit Testing Focus** | **Correlation between Git and Unit Testing** | **Goal/Outcome** |
| --- | --- | --- | --- | --- |
| 1 | **Initialize Git Repository** | **Test Initial Setup** | Before any code is written, initialize the repository and prepare for version control. | Ensures that the project is ready for version tracking and that the initial unit tests are planned. |
| 2 | **Add files to the staging area** | **Testable Code Implementation** | As features are implemented, files are added to Git, ensuring that each new piece of code can be tested. | Ensures that unit tests are added as features are written, allowing for proper tracking of changes. |
| 3 | **Commit changes** | **Committing Test Results** | Commit each feature with unit test coverage to track test results for every change. | Each committed change should be accompanied by corresponding unit tests to verify correctness. |
| 4 | **Verify commit history** | **Review Unit Test Changes** | Review commit history to ensure that new features or fixes are accompanied by unit tests. | Helps identify any missing tests and ensures that unit tests align with new changes made in the code. |
| 5 | **Branch creation** | **Unit Test Isolation** | Creating a new branch for a specific feature or bugfix allows unit tests to be isolated and worked on independently. | Isolates the testing of a specific feature to ensure it's properly verified before merging with the main branch. |
| 6 | **Switch between branches** | **Switching Test Focus** | Switch branches to focus on testing specific features. Different test cases are handled separately in different branches. | Allows developers to focus on unit testing for one feature while not interfering with others. |
| 7 | **Merge branches** | **Merge Test Results** | When merging branches, ensure unit tests for the feature being merged pass before merging into the main branch. | Ensures that unit tests from different branches are integrated and that no tests break during merging. |
| 8 | **Handle merge conflicts** | **Conflict Resolution in Test Cases** | Merge conflicts may require changes in unit tests. Developers must resolve conflicts while maintaining passing tests. | Ensures that any conflicts in the code are resolved without breaking existing unit tests. |
| 9 | **Push changes to remote repository** | **Pushing Test Results** | After committing, push changes to the remote repository with tests that pass and reflect the latest changes. | Shares the code and test results with the team to ensure consistency across the development environment. |
| 10 | **Pull latest changes from remote** | **Pull Latest Test Updates** | Pull the latest code and corresponding unit tests from the repository to keep local testing up to date. | Ensures that any changes or additions to tests from other developers are merged and properly tested. |
| 11 | **Clone a repository** | **Initial Unit Testing Setup** | When cloning a repository, set up the project environment to run unit tests on the existing codebase. | Provides a starting point for testing in a new development environment, ensuring tests can run out-of-the-box. |

This table highlights the correlation between the Git version control process and unit testing in the software development lifecycle. The steps for managing code changes through Git and running unit tests are deeply intertwined, ensuring that new features and fixes are tested early and often. By integrating unit tests at each stage of version control, we can maintain a high level of code quality and stability, while also making it easier to track changes and test results.

# **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>